Defining Cobalt Strike Components So You Can BEA-CONfident in Your Analysis

mandiant.com/resources/defining-cobalt-strike-components

Blog

Alyssa Rahman

Oct 12, 2021

23 mins read

Malware

Threat Research

Cobalt Strike is a commercial adversary simulation software that is marketed to red teams but is also stolen and actively used by a wide range of threat actors from ransomware operators to espionage-focused Advanced Persistent Threats (APTs). Many network defenders have seen Cobalt Strike payloads used in intrusions, but for those who have not had the opportunity to use Cobalt Strike as an operator, it can be challenging to understand the many components and features included in this framework.

In this blog post, we will walk through important definitions and concepts to help defenders understand Cobalt Strike and, hopefully, identify new ways to hunt for, respond to, and attribute malicious actors using this tool.

This content is informed by a combination of official documentation, public research, and Mandiant's own experience with both performing red team assessments and responding to intrusions where threat actors are using Cobalt Strike. Because Cobalt Strike is used for both legitimate testing and for intrusions, in this post we will refer to all Cobalt Strike users, regardless of motivation, as "operators" for simplicity.

While this blog post is focused on explaining Cobalt Strike and BEACON artifacts to aid defenders' analysis, we have also included references to several resources to learn more about Cobalt Strike and how to detect common BEACON activity at various stages of the attack lifecycle.

Important Components

Cobalt Strike, BEACON, Team Server. Oh My!

You may hear the names Cobalt Strike, BEACON, and even team server used interchangeably, but there are some important distinctions between all of them.

Cobalt Strike is the *command and control* (*C2) application* itself. This has two primary components: the team server and the client. These are both contained in the same Java executable (JAR file) and the only difference is what arguments an operator uses to execute it.

- **Team server** is the C2 server portion of Cobalt Strike. It can accept client connections, BEACON callbacks, and general web requests.
  - By default, it accepts client connections on TCP port 50050.
  - Team server only supports being run on Linux systems.
- **Client** is how operators connect to a team server.
  - Clients can run on the same system as a Team server or connect remotely.
  - Client can be run on Windows, macOS or Linux systems.

BEACON is the name for Cobalt Strike's default malware payload used to create a connection to the team server. Active callback sessions from a target are also called "beacons". (This is where the malware family got its name.) There are two types of BEACON:

- The **Stager** is an optional BEACON payload. Operators can "stage" their malware by sending an initial small BEACON shellcode payload that only does some basic checks and then queries the configured C2 for the fully featured backdoor.
- The **Full backdoor** can either be executed through a BEACON stager, by a "loader" malware family, or by directly executing the default DLL export "ReflectiveLoader". This backdoor runs in memory and can establish a connection to the team server through several methods.

Loaders are *not* BEACON. BEACON is the backdoor itself and is typically executed with some other loader, whether it is the staged or full backdoor. Cobalt Strike does come with default loaders, but operators can also create their own using PowerShell, .NET, C++, GoLang, or really anything capable of running shellcode.

It's All Connected

Listeners are the Cobalt Strike component that payloads, such as BEACON, use to connect to a team server. Cobalt Strike supports several protocols and supports a wide range of modifications within each listener type. Some changes to a listener require a "listener restart" and generating a new payload. Some changes require a full team server restart.

- **HTTP/HTTPS** is by far the most common listener type.
    - While Cobalt Strike includes a default TLS certificate, this is well known to defenders and blocked by many enterprise products ("signatured"). Usually operators will generate valid certificates, such as with LetsEncrypt, for their C2 domains to blend in.
    - Thanks to Malleable Profiles (discussed later in the post), operators can heavily configure how the BEACON network traffic will look and can masquerade as legitimate HTTP connections.
    - Operators can provide a list of domains/IPs when configuring a listener, and the team server will accept BEACON connections from all of them (see "Redirectors"). Operators can also specify Host header values (see "Domain Fronting").
- **Hybrid (DNS+HTTP)** is the default and uses DNS for a beacon channel and HTTP for a data channel.
- **Pure DNS** can also be enabled to use DNS for both beacon and data channels. This leverages regular A record requests to avoid using HTTPS and provide a stealthier, though slower method of communication.
- **SMB** is a bind style listener and is most often used for chaining beacons. Bind listeners open a local port on a targeted system and wait for an incoming connection from an operator. See "Important Concepts > Chaining Beacons" for more information.
- **Raw TCP** is a (newer) bind style listener and can also be used for chaining beacons. See "Important Concepts > Chaining Beacons" for more information.

The final two listeners are less common, but they provide compatibility with other payload types.

- **Foreign listeners** allow connections from Metasploit's Meterpreter backdoor to simplify passing sessions between the Metasploit framework and the Cobalt Strike framework.
- **External C2** listeners provide a specification that operators can use to connect to a team server with a reverse TCP listener. Reverse listeners connect back and establish an external connection to an operator, instead of waiting for an incoming connection such as with "bind" listeners.

It’s (Almost) All Customizable

Arsenal Kits are available for download, with a valid license, and for use with licensed (or cracked) installations only. Arsenal kits are sometimes distributed with cracked copies of Cobalt Strike. The full list of kits (as of October 2021) is:

- **Applet/PowerApplet Kit** allows operators to modify Cobalt Strike’s built-in Java Applet payloads. This kit was the first to be added to Arsenal and is no longer widely used.
- **Artifact Kit** allows operators to modify the templates for all Cobalt Strike executables, DLLs, and shellcode. This kit was added in January 2014 and is still used.
- **Elevate Kit** allows operators to integrate their privilege escalation exploits as Cobalt Strike commands. This kit was released in December 2016. Unlike other kits, this is public, and it is essentially just an Aggressor Script to streamline loading your own PowerShell scripts, binaries, etc. into a BEACON session (see “Aggressor Scripts”).
- **Mimikatz Kit** allows operators to update their version of Mimikatz without waiting for a Cobalt Strike software update. This kit was added in July 2021.
- **Resource Kit** allows operators to modify the script templates Cobalt Strike uses (mostly as loaders). This kit was added in May 2017 and is still used.
-
- **User Defined Reflective Loaders Kit** allows operators to customize the reflective loader functionality used by BEACON. This kit was added in August 2021.

Malleable Profileis the final part of Arsenal Kit, and it allows operators to extensively modify how their Cobalt Strike installation works. It is the most common way operators customize Cobalt Strike and has thus been heavily documented.

- Changes to a Malleable Profile require a team server restart and, depending on the change, may require re-generating payloads and re-spawning beacon sessions.
- There are several robust open-source projects that generate randomized profiles which can make detection challenging. Still, operators will often reuse profiles (or only slightly modify them) allowing for easier detection and potentially attribution clustering.
- When analyzing samples, check GitHub and other public sources to see if the profile is open source.

Aggressor Scripts are macros that operators can write and load in their client to streamline their workflow. These are loaded and executed within the client context and don't create new BEACON functionality, so much as automate existing commands. They are written in a Perl-based language called "Sleep" which Raphael Mudge (the creator of Cobalt Strike) wrote. For an example, check out the "An Operator's View" section.

Aggressor scripts are *only* loaded into an operator's local Client. They are *not* loaded into other operators’ clients, the team server, or BEACON sessions (victim hosts). Theprimary detection opportunity for these scripts is reviewing them for static defaults that can be used in detection or hunting rules.

Execute-Assemblyis a BEACON command that allows operators to run a .NET executable in memory on a targeted host. BEACON runs these executables by spawning a temporary process and injecting the assembly into it. In contrast to Aggressor Scripts, execute-assembly does allow operators to extend BEACON functionality. Assemblies run in this way will still be scanned by Microsoft's AMSI if it is enabled.

Beacon Object Files (BOFs) are a fairly recent Cobalt Strike feature that allows operators to extend BEACON post-exploitation functionality. BOFs are compiled C programs that are executed in memory on a targeted host. In contrast to Aggressor Scripts, BOFs are loaded within a BEACON session and can create new BEACON capabilities. Additionally, compared to other BEACON post-exploitation commands like execute-assembly, BOFs are relatively stealthy as they run within a BEACON session and do not require a process creation or injection.

An Operator's View

Four of the aforementioned components—the Client, Aggressor Scripts, Beacon Object Files, and Malleable Profiles—are the primary ways operators interact with and customize their team server. An example of each is included here to show these components from an operator's perspective.

An operator accessing a team server through the Cobalt Strike **client** would see a view like the following. The top pane shows a list of active beacon sessions with basic metadata including the current user, process ID, internal and external IP addresses, and the last time the host checked in with the team server. The bottom pane includes a tab for each session where operators can send commands to the victim hosts and see a log of past commands and output. The client interface also allows operators to build payloads, execute plugins, and generate reports.

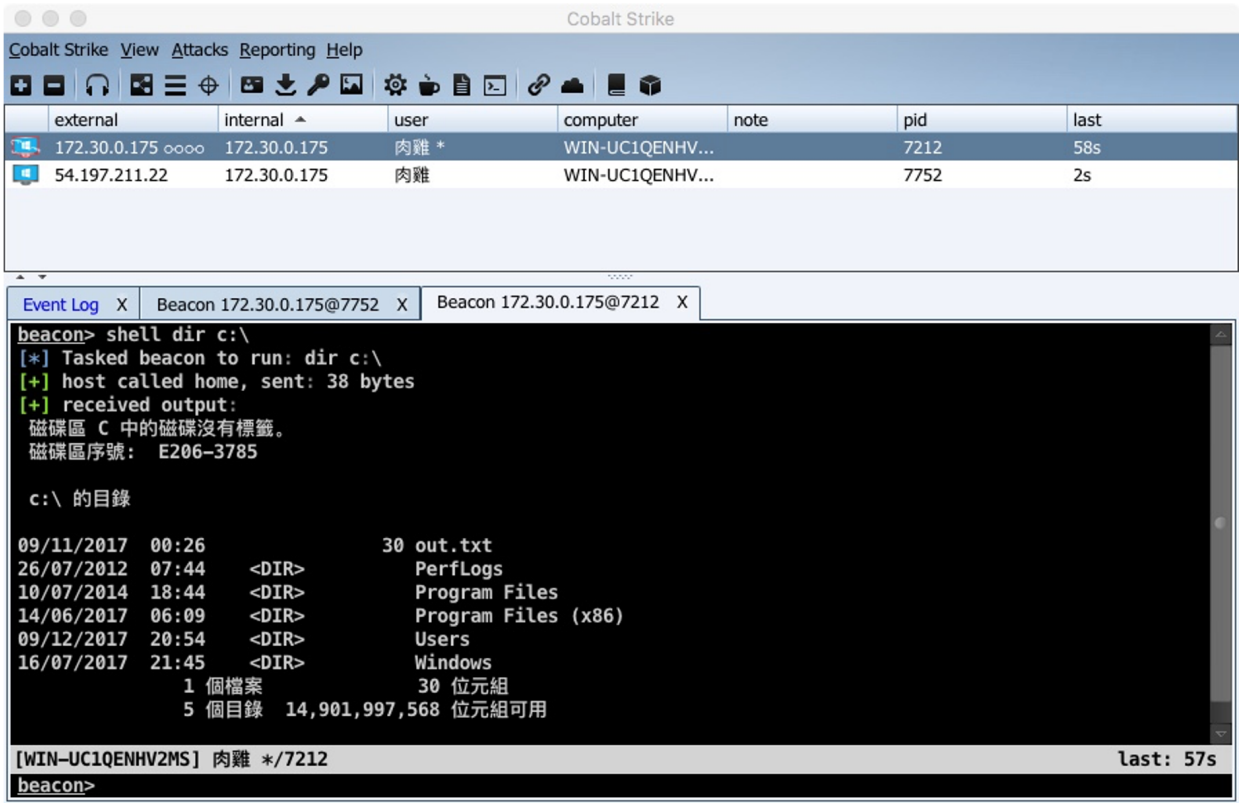

Figure 1: Cobalt Strike 3.10 client view (source)

Within the client, operators can import **Aggressor Scripts** to customize their commands, menu options, and interface. Aggressor Scripts vary in complexity, from adding a new menu shortcut to chaining multiple attack steps. The following is an excerpt from credpocalypse.cna, an Aggressor Script that checks active beacon sessions on a schedule and runs Mimikatz, an open-source credential dumper, if a new user logs in.

Note that this is not adding functionality to Cobalt Strike. In this case, the script uses built-in BEACON functionality to list processes and execute Mimikatz, and it uses Cobalt Strike APIs to run on a schedule. That means existing detections for BEACON's Mimikatz module will also detect this.

```
sub steal_creds {
    #Log what you're doing (this output shows in Script Console)
    @pids = map({ return beacon_info($1, "pid"); }, @watchlist);
    println("Looking for new users in PIDs: " . join(", ", @pids));

    #Update creds list
    clear(@captured_creds);
    foreach %cred (credentials()) { #Add all the options
        push(@captured_creds, uc(%cred['realm']) . '\\' . uc(%cred['user']));
    }

    #Check each beacon for new users
    foreach $bid (@watchlist) {
        get_users($bid, {
            if ( $2 ) {
                #Log to beacon
                btask($1, "[" . formatDate('yyyy-MM-dd HH:mm:ss z') . "] New
user! Running logonpasswords.");

                #Log to script console
                println("[" . formatDate('yyyy-MM-dd HH:mm:ss z') . "] BID: "
. $1 . " PID: " . beacon_info($1, "pid") . " New user! Running
logonpasswords.");

                #Run Mimikatz "logonpasswords" function
                blogonpasswords($1);
            }
            else {
                println("No new users in BID: " . $1 . " PID: " .
beacon_info($1, "pid"));
            }
        });
    }

}

on heartbeat_60m {
    if ( size(@watchlist) > 0 && ($interval cmp "60m") == 0) {
        steal_creds();
    }
}
```

Beacon Object Files are single file C programs that are run within a BEACON session. BOFs are expected to be small and run for a short time. Since BEACON sessions are single threaded, BOFs will block any other BEACON commands while they are executing. The following is an example from the Cobalt Strike documentation that uses Dynamic Function Resolution to look up the current domain.

```
#include <windows.h>
#include <stdio.h>
#include <dsgetdc.h>
#include "beacon.h"

DECLSPEC_IMPORT DWORD WINAPI NETAPI32$DsGetDcNameA(LPVOID, LPVOID, LPVOID,
LPVOID, ULONG, LPVOID);
DECLSPEC_IMPORT DWORD WINAPI NETAPI32$NetApiBufferFree(LPVOID);

void go(char * args, int alen) {

    DWORD dwRet;PDOMAIN_CONTROLLER_INFO pdcInfo;

    dwRet = NETAPI32$DsGetDcNameA(NULL, NULL, NULL, NULL, 0, &pdcInfo);

    if (ERROR_SUCCESS == dwRet) {

        BeaconPrintf(CALLBACK_OUTPUT, "%s", pdcInfo->DomainName);

    }

    NETAPI32$NetApiBufferFree(pdcInfo);

}
```

Finally, **Malleable Profiles** allow operators to customize a wide range of settings when they first launch their team server. The snippet that follows from a public profile is an example of how an operator could make BEACON traffic look like it's related to Amazon. The portions in blue (the set uri line and the client block), define how a BEACON payload behaves. Some of these values can be extracted from a BEACON sample. See “Interpreting Artifacts > GET and POST Requests” for more information.

```
http-get {
    set uri "/s/ref=nb_sb_noss_1/167-3294888-0262949/field-keywords=books";
    client {
        header "Accept" "*/*";
        header "Host" "www.amazon.com";
        metadata {
            base64;
            prepend "session-token=";
            prepend "skin=noskin;";
            append "csm-hit=s-24KU11BB82RZSYGJ3BDK|1419899012996";
            header "Cookie";
        }
    }
    server {
        header "Server" "Server";
        header "x-amz-id-1" "THKUYEZKCKPGY5T42PZT";
        header "x-amz-id-2"
"a21yZ2xrNDNtdGRsa212bGV3YW85amZuZW9ydG5rZmRuZ2tmZGl4aHRvNDVpbgo=";
        header "X-Frame-Options" "SAMEORIGIN";
        header "Content-Encoding" "gzip";
        output {
            print;
        }
    }
}
```

Important Concepts

Stagers

Earlier, I mentioned there are "two types of BEACON", one of them being a stager. Operators can have stagers for multiple listener types (e.g. a DNS stager, an SMB stager, an HTTPS stager). In those cases, when the stager shellcode is executed, it will pull the final BEACON payload over the relevant protocol and execute it, establishing a connection using the defined listener method.

An important note for defenders is that, *by default,* defenders can download a Cobalt Strike HTTP/S stager payload from a team server even if the operator is not using staged payloads in their operations. This will allow defenders to 1. confirm something is hosting a team server with a listener on that port and 2. extract additional configuration artifacts from the payload.

This works because Cobalt Strike was designed to be compatible with Metasploit's Meterpreter payload. Metasploit (and thus Cobalt Strike) will serve an HTTPS stager when a valid URL request is received. A valid URL is any 4-character alphanumeric value with a valid 8 bit checksum calculated by adding the ASCII values of the 4 characters.

Operators can prevent defenders from retrieving stagers by setting the host_stage Malleable Profile value to "false". More commonly, they may use reverse proxies to filter out unwanted traffic like stager requests. As a protection feature, Cobalt Strike will ignore web requests with blacklisted User-Agents, such as curl or wget. Starting in Cobalt Strike 4.4, operators can also *whitelist* user agents with the .http-config.allow_useragents Malleable Profile option. These caveats are important to remember, since a team server may not always function as expected by scanners that automate stager requests.

As an operational security note, operators can also detect any web request to a team server, as it will be visible to the operator in their logs. They will also be able to see in the "Web Log" view if a stager has been pulled, along with all HTTP request details like source IP.

Trial vs Licensed vs Cracked

Cobalt Strike is not *legitimately* freely available. Copies of the team server/client cannot be downloaded as a trial or licensed copy from Help Systems—the company that owns Cobalt Strike—unless the operator applies and has been approved. Unfortunately, trials and cracked copies (including most, if not all, licensed features) have been and continue to be leaked and distributed publicly for nearly all recent versions.

- **Trial** versions of Cobalt Strike are heavily signatured and include lots of obvious defaults intended to be caught in a production environment. (For example, it embeds the EICAR string in all payloads.) This is to ensure that the operator is really using it as a trial and will eventually pay if using it for professional purposes.
- **Licensed** versions of Cobalt Strike include more features (e.g. Arsenal Kits) and fewer embedded artifacts (no more EICAR!). A watermark related to the associated Cobalt Strike license is still embedded in payloads and can be extracted using most BEACON configuration parsers. There are a lot of caveats with that value, so please see "Interpreting Artifacts > Watermarks" later in the post.
    Licenses can be stolen, however if a license is revoked operators will no longer be able to use it to update an installation. If operators keep the "authorization file" (See "Interpreting Artifacts > Watermarks"), the existing installation will still work until expiration.
- **Cracked** versions of Cobalt Strike are distributed in various forums. Typically, these are the result of someone modifying a trial JAR file to bypass the license check and rebuilding the JAR, or by crafting an authorization file with a fake license ID and distributing that with the JAR.

Differentiating cracked versions of Cobalt Strike from legitimately licensed versions can be difficult. If the watermark was statically defined as part of the cracking process, it may be possible to tie that to a shared/distributed copy. Some cracked copies of Cobalt Strike also contain backdoors to provide third-party access to a team server. In these cases, malware analysts may be able to identify illegitimate copies through static analysis.

Redirectors

Instead of having beacons connect directly to a team server, operators will sometimes use a redirector (or several) that accepts connections and forwards them to the team server. This has several advantages for operators, including being able to:

- Cycle through multiple domains for a single BEACON connection
- Replace detected/blocked redirectors without having to replace the underlying team server
- Use high(er) reputation domains that help BEACON traffic blend in and avoid detection

Operators can also use redirectors to filter out "suspicious" traffic, like scanners or hunting tools, to protect their team server, however there are typically still easy wins to track down team servers and redirectors. See the "Hunting for Team Servers" section for more details.

What About Domain Fronting?

Sometimes "redirectors" are as simple as a cloud instance with an nginx proxy, but another highly effective redirector method is "domain fronting". Domain fronting is a technique by which an operator may hide the true destination of a network connection by redirecting it through the infrastructure of a Content Delivery Network (CDN). The technique was first documented as a means of bypassing internet censorship and has also been used by threat actors, such as APT 29, to disguise C2 traffic.

When an HTTPS request is fronted, the connection is established directly with a reputable domain hosted by the CDN. This is the “fronted” domain. The encrypted request will contain a unique identifier, often contained in the HTTP “Host” header, that the CDN uses to route the request to an operator-controlled server. Because domain fronted traffic is initially sent to the CDN, it will use the legitimate SSL/TLS certificate of the CDN when observed by a defender.

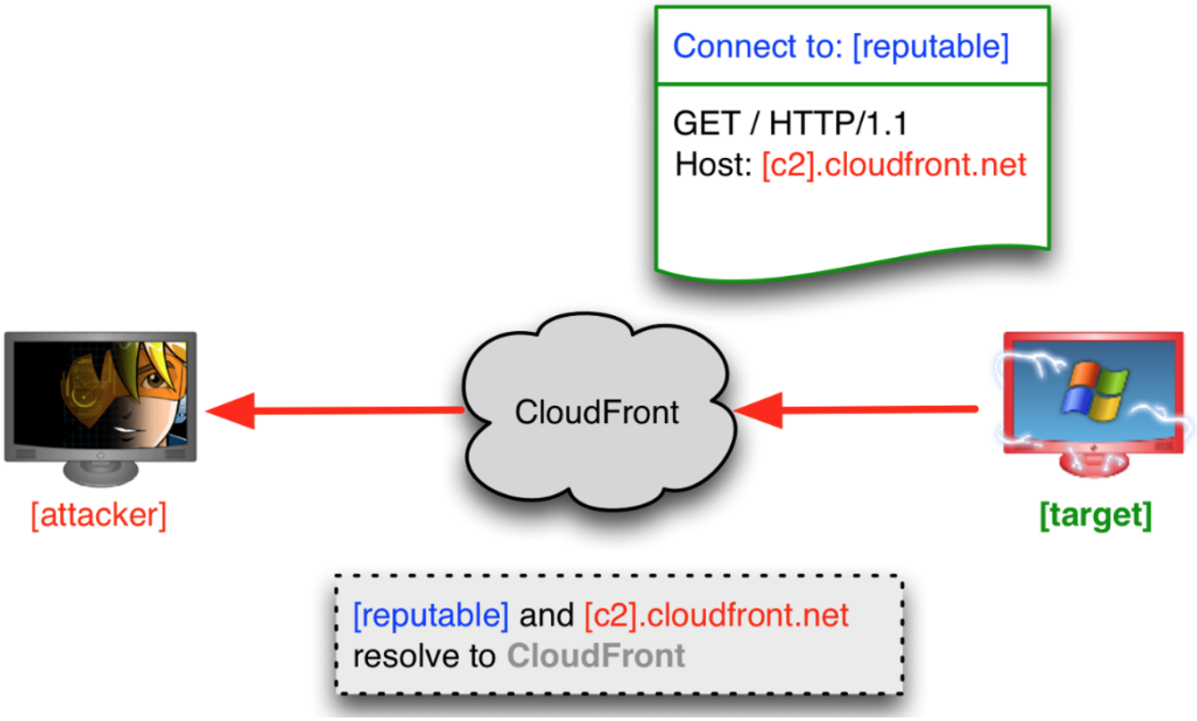

Figure 2: Domain fronted C2 connection (source)

In this situation, to block traffic to the operator's domain, a defender would need to decrypt the traffic to discover the true destination within an HTTPS connection, but this is not always practical. In addition to being resource intensive, decrypting the traffic for some CDNs may not be possible. For example, some CDNs enforce certificate pinning on their SSL/TLS certificates, to prevent interception and decryption of the traffic using an organization-provided trusted root certificate.

Domain Fronting vs Masquerading

Masqueraded traffic is designed to look like a legitimate service but is actually a **direct connection** to an operator's server, while **fronted traffic** is **sent to a legitimate service** (CDN) and forwarded from there to the operator.

If the Host header value and destination domain are the same, this is just a normal, *direct HTTP connection.*

If the Host header and destination domain are different and the Host header is

- A legitimate domain (e.g. in the Alexa Top 1M), then it is likely *domain masquerading*
- A domain used for CDN endpoints (e.g. *.azureedge.net), then it may be *domain fronting.*

There are multiple public guides to test if a domain is frontable. This repo also includes a pre-compiled list of frontable domains by CDN provider. Please note, public lists may be out of date and manual validation will likely still be necessary.

Chaining Beacons

Another way operators can establish connections is by chaining beacons. Once an operator has a single compromised system, for example beaconing back to an HTTPS listener, they can pivot internally to other systems.

The most common reason an operator will chain beacons is to **bypass network segmentation** and restrictions. If they target a server that doesn't have outbound internet access, they can proxy their connection through another beacon with network connectivity to the targeted system and receive the callback on their team server. If the operator loses access to the parent system, they will also lose access to chained beacons.

SMB listeners and SMB staged beacons were the original method for chaining. Cobalt Strike now also supports a raw TCP listener. In environments where the SMB stager is detected (increasingly common for Endpoint Detection and Response products), an operator can use a raw TCP chain over port 445. This will allow them to still leverage the fact that SMB is rarely blocked between internal hosts but avoid using the more heavily signatured SMB listener.

In the screenshot from the "An Operator's View" section, the first session's external IP matches the internal IP for session 2. This indicates it is being chained through the second session. A double infinity/chain icon next to the IP also shows that it is chained. If the system loses connection to its parent host, that chain icon will be disconnected.

The following screenshot shows the "Pivot Graph" view from a Cobalt Strike client. In this graphic, each BEACON is represented by a computer icon. Chained sessions are identified by arrows to their child sessions, and the firewall icon represents an external connection to the operator's C2.

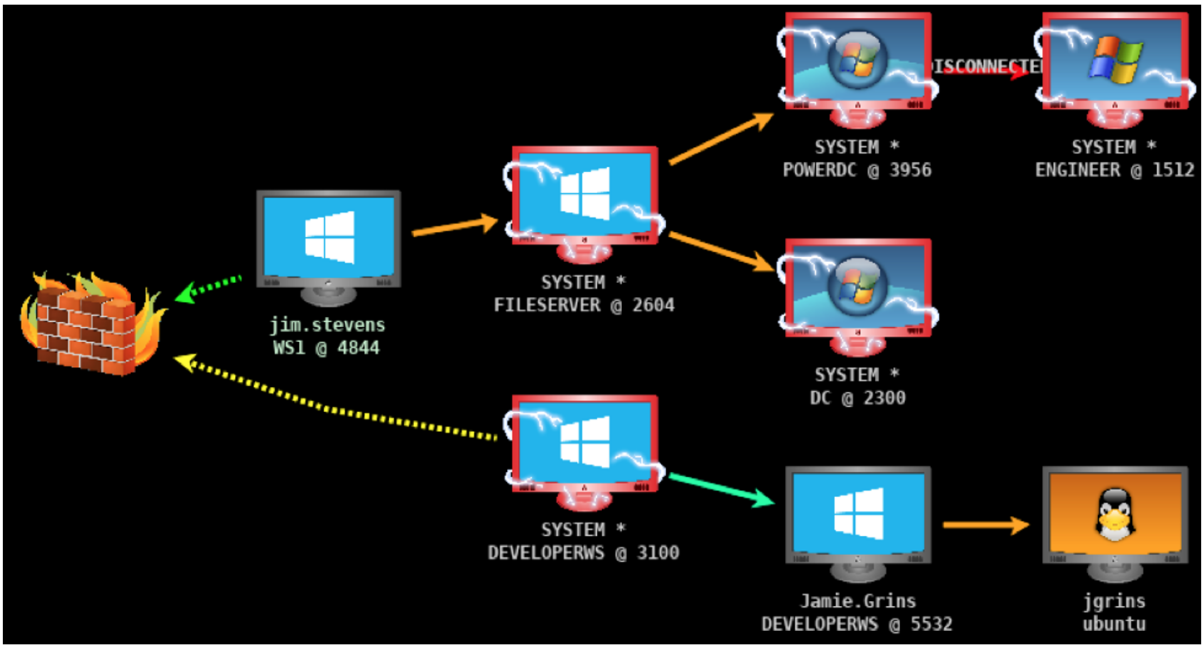

Figure 3: Client "graph view" of chained BEACONs (source)

If responding to an intrusion with SMB or TCP staged BEACON payloads, look for other compromised hosts starting with the IP addresses configured as the “C2” for the staged payloads.

Hunting for Team Servers

Mandiant’s Advanced Practices team regularly scans the internet for C2s, including Cobalt Strike team servers. If an operator does not properly protect their team server, analysts can identify malicious infrastructure without prior knowledge of if or where it is being used and provide advanced warning and coverage for customers. For more details on how we identify C2s like Cobalt Strike, check out our blog post, “SCANdalous! (External Detection Using Network Scan Data and Automation).”

Interpreting Artifacts

Team Server and Client Logs

Cobalt Strike logs are invaluable for understanding the activities conducted using a team server. Defenders may not frequently have access to this log data, but in the event that logs are available, it is useful to understand what data can be extracted from them.

Cobalt Strike stores logs in two primary formats: **full, plaintext beacon logs** and **Java serialized bins**. These are stored in the team server working directory and duplicated in client working directories when a client is connected to a team server. They are also updated when items change, such as new credentials being captured.

The **cobaltstrike/logs/** directory includes a directory structure of the format **[date]/[internal ip of beaconed host]/[beacon id].log**. Each file is a plaintext log of every beacon command and the associated output. This mirrors what an operator would see in the Cobalt Strike beacon console.

The **cobaltstrike/screenshots/**and **cobaltstrike/downloads/** folders each respectively contain all screenshots or files an operator has downloaded from beacons.

- An individual operator/client working directory will only contain downloads which that operator synchronized to their local system, and these could be from multiple team server connections.
- A team server working directory, will contain screenshots/downloads from any operator on that team server.

Under each **cobaltstrike/logs/[date]** directory, there is also an **events.log** and a **web.log** file.

- Events.log is a mirror of the "events" console for the team server which, at a minimum, will include the date/time of each operator connecting to the server and each initial beacon callback.

- Web.log will show a log of every web request to the team server, including requests for staged payloads or team server-hosted files.

Files hosted on a team server and served through the Web feature of Cobalt Strike are saved in the **cobaltstrike/uploads/** directory. An individual operator/client working directory, will only contain files which that operator uploaded.

The **cobaltstrike/data** directory includes several .bin files which are serialized Java objects of different data models used by Cobalt Strike to track its state. The data from Cobalt Strike serialized .bin logs can be extracted as CSVs using this script.

For most analysts, sessions.bin, listeners.bin, and credentials.bin will be of the highest interest. The .bin logs contain the most concentrated and simple to parse data for identifying which systems and users were compromised using a given team server.

- Sessions.bin lists all active and historical beacon sessions from the given team server along with some basic metadata (e.g. user, internal and external IP, hostname, architecture).
- Listeners.bin details all active listeners for the given team server.
- Credentials.bin includes any credentials (cleartext, hashed, etc.) that were either automatically dumped with a built-in credential module (such as Mimikatz) or manually added (such as after cracking a password offline).

Watermarks

Cobalt Strike **watermarks** are a unique value generated from and tied to a given "CobaltStrike.auth" file. This value is embedded as the last 4 bytes for all BEACON stagers and in the embedded configuration for full backdoor BEACON samples.

The **CobaltStrike.auth** file is a config file used by Cobalt Strike to determine license ID and expiration. When launched, Cobalt Strike will check that the license is valid and unexpired. The CobaltStrike.auth file is required to launch modern versions of Cobalt Strike, and it is updated when updating Cobalt Strike and when entering a license (whether for the first time or as a re-entry).

A **matching watermark means** that two payloads came from a team server(s) using the same CobaltStrike.auth file. This **does not necessarily mean** it came from the same operator. Someone can copy the whole Cobalt Strike directory, including the auth file, and install it on another server which would then have the same watermark until the license expired.

Public Keys

Cobalt Strike uses 4 different types of keystores:

- **Listeners** can have custom keystores specified in Malleable Profiles.
- **Code signing** can be configured with custom keystores specified in Malleable Profiles.

- **Client Connections** use the cobaltstrike.store file to encrypt client communications.
- **Beacon Connections** use a team server generated keystore to encrypt BEACON data and handle other security features.

The BEACON keys and other security features are discussed in the Raphael Mudge's training "Advanced Threat Tactics: Infrastructure". These keys are only used in the full BEACON backdoor, not stagers. As identified by NCCGroup, these keys are stored in a serialized file called ".cobaltstrike.beacon_keys" in the team server working directory.

A **matching public key means** that two payloads came from a team server(s) using the same .cobaltstrike.beacon_keys keystore. This **does NOT NECESSARILY mean** it came from the same team server. Again, someone could copy the whole Cobalt Strike directory, including the keystore, as is sometimes done with distributed or cracked copies.

C2 IPs and URLs

BEACON parsers will also typically extract C2 IPs, domains, and URLs from samples. Some of these may be legitimate domains, for example if the operator is using domain fronting or masquerading.

There are some **gotchas** to be aware of when attempting to cluster BEACON activity for attribution purposes. For example, the following cases do **not necessarily** mean you're looking at different team servers.

- *Two samples connect to different IPs* - This could be the result of changing redirectors or multiple domains/hosts being defined in a listener.
- *Two samples refer to different URL paths* - Operators can specify multiple URL paths when defining Malleable Profiles. Each time an operator generates BEACON shellcode, Cobalt Strike will select a random value from this list. If two samples have different URLs, it's possible they are still referring to the same server.

GET and POST Requests

BEACON configurations may include custom HTTP headers, Cookies, etc. for HTTP GET and POST requests. Those fields are representations of how an operator has configured their team server (using Malleable Profile or default values) to construct and disguise the BEACON network traffic. This is **not** a direct 1-to-1 for what will be in a payload. Malleable Profiles accept multiple values for several fields, and only one of them may appear in a given payload. If the GET or POST requests don't match exactly, that is not necessarily an indication that it's from a different server.

Spawn To

Each BEACON payload will be configured with two "spawn to" processes, one for 32-bit tasks and one for 64-bit tasks. These values are what BEACON will spawn as temporary processes for various post-exploitation commands. BEACON launches the process, injects into it (with whatever technique the operator has specified), executes the post-

exploitation task, and terminates the process. As an example of what commands this may affect, check out this older guide: Opsec Considerations for Beacon Commands. The default spawn to process is rundll32.exe, but this can be modified in two ways.

- **Malleable Profile** allows operators to modify the default spawn_to processes (if using a licensed or cracked version). Changing these values will require restarting the team server and regenerating new BEACON payloads to use the new values.
- **Client** allows operators to interactively modify the spawn_to processes once a BEACON session is established. These can be changed as many times as desired and don't require a restart, although it will only affect the current BEACON session.

Named Pipes

Named pipes are a Windows feature used for interprocess communication (IPC). Cobalt Strike uses named pipes in several ways:

- **Payloads** - Used to load the backdoor into memory and are modifiable via Malleable profile and/or Artifact Kit.
- **Post-exploitation Jobs** - Used for a variety of Cobalt Strike commands that need to spawn and inject into a process.
- **Staging** - Used for SSH/SMB BEACON staging and chaining

Raphael Mudge's post, "Learn Pipe Fitting for all of your Offense Projects," has some helpful additional details on this feature.

Sleep Time

The "sleeptime" value in a BEACON configuration is the base time used for callback intervals. BEACON will randomize callbacks within a range determined by the "jitter" percentage. For example, a sleeptime of 10 seconds and a jitter of 50%, will produce callbacks every 5 to 15 seconds. This value is not constant to make network detection more challenging.

In Malleable Profile, operators can specify a sleeptime in **milliseconds**. After a session is created, operators can interactively change the sleeptime and jitter values, but both must be integers and sleeptime must be defined in **seconds**.

Kill Date

The "killdate" value in a BEACON sample dictates whether it should connect to a team server after a given date. This is used by red team operators to ensure payloads are not accidentally executed after an engagement is complete. It can also be used by threat actors to limit successful sandbox execution and frustrate retroactive analysis efforts.

This value is defined by a command line argument when the team server is started, and it is not modifiable through Malleable Profile or interactive BEACON commands. By default, no killdate is specified.

Conclusion

Cobalt Strike continues to be the C2 framework of choice for both legitimate security testers and threat actors alike. Our hope is that with a more thorough understanding of the framework's capabilities and common usage, defenders will be more equipped to find new ways to hunt, respond to, and attribute malicious actors using this tool.

References

Detection Guides

Analysis Resources

Walkthroughs and Examples

The creator of Cobalt Strike, Raphael Mudge, created a nine-part YouTube course, Red Team Ops with Cobalt Strike, showing how to use Cobalt Strike. This is worth watching to see what the product looks like and how to use the various features discussed in this blog post (+ some).

Acknowledgements

Special thanks to Tyler Dean, Jae Young Kim, Matthew Dunwoody, and Chris King for their technical expertise and review.