What is a stageless payload artifact?

Wednesday 15 June, 2016

I've had a few questions about Cobalt Strike's stageless payloads and how these compare to other payload varieties. In this blog post, I'll explain stageless payloads and why you might prefer stageless payload artifacts in different situations.

What is payload staging?

A stageless payload artifact is an artifact [think executable, DLL, etc.] that runs a payload without staging. To understand stageless payloads, it helps to understand payload staging first.

Many of Cobalt Strike's attacks and workflows deliver a payload as multiple stages. The first stage is called a stager. The stager is a very tiny program, often written in hand-optimized assembly, that: connects to Cobalt Strike, downloads the Beacon payload (also called the stage), and executes it.

The payload staging process exists for a reason. Staging makes it possible to deliver Beacon [or another payload] in an attack or artifact that has a size constraint. For example, several of Beacon's lateral movement commands run a PowerShell one-liner to kick off the payload you specify. This PowerShell one-liner is limited to the maximum length of a Windows command + arguments. I can't stuff a ~200KB payload stage into this space. A payload stager that is a few hundred bytes works just fine though.

What is a stageless payload artifact?

A stageless payload artifact is an artifact that contains the payload stage and its configuration in a self-contained package. Cobalt Strike has had the option to export stageless Beacon artifacts since its March 2014 release. I used to call these "staged" artifacts, but I adopted Metasploit's nomenclature when the framework gained this capability. Stageless Beacon artifacts include: an executable, a service executable, DLLs, PowerShell, and a blob of shellcode that initializes and runs the Beacon payload.

Why would I use stageless payload artifacts?

Stageless payloads are beneficial in many circumstances. Stageless payloads are a way to work without the staging process. Payload staging is a fragile process and some defenses mitigate it. Stageless payloads allow you to benefit from Beacon's security features, right away. Beacon authenticates its team server and encrypts communication to and from the team server. Beacon can do this because it does not have the same size constraints a stager has. Cobalt Strike's stagers do not have any security features. If an attacker can intercept and manipulate the staging process, they can replace your stage with theirs. This is a concern in some situations.

What's inside of a stageless payload artifact?

Cobalt Strike's stageless payload executables and DLLs are not much different from its stager-delivering counterparts. Cobalt Strike's Artifact Kit builds artifacts for stageless payloads and payload stagers from the same source code. The difference is the executables and DLL templates for stageless payloads have more space to hold the entire Beacon payload.

The common denominator for Cobalt Strike's stageless payload artifacts is the raw output. Think of this as a big blob of shellcode that contains and runs Beacon. When you export a stageless payload artifact, Cobalt Strike patches this big blob of shellcode into the desired artifact template (PowerShell, executable, DLL, etc.).

What's inside of the raw stageless payload artifact?

The raw stageless artifact is a self-bootstrapping Reflective DLL. A Reflective DLL is a Windows DLL compiled with a Reflective Loader function. The Reflective Loader function is like LoadLibrary, except it can load a DLL that resides somewhere in memory. Stephen Fewer developed the Reflective DLL loader that Cobalt Strike and other toolsets use.

Cobalt Strike's Beacon is compiled as a Reflective DLL. This allows various payload stagers and the stageless artifacts to inject Beacon into memory.

Now, let's discuss the self-bootstrapping part. If you're with me so far, you understand that a Reflective DLL is a DLL with this loader function built into it. This does not make the Reflective DLL self-bootstrapping.

The way the Reflective DLL becomes self-bootstrapping is to overwrite the beginning of the DLL with a valid program that does the following things:

(1) Resolve the memory address where the (currently running) bootstrap program/Reflective DLL resides

(2) Call the Reflective Loader with (1) as an argument. The Reflective Loader lives at a predictable offset from (1). When you see “Offset is: 4432” in the Cobalt Strike console, that’s Cobalt Strike resolving the offset to the Reflective Loader within a DLL.

(3) After the Reflective Loader initializes a DLL, it calls the DLL’s DllMain function. This is when control is passed to Beacon.

In summary, the raw output of Cobalt Strike’s stageless payload is a Reflective DLL with a valid program patched in over the PE header. The patched in program performs steps (1), (2), and (3). This valid program is what makes the Reflective DLL blob self-bootstrapping. This self-bootstrapping blob is a stageless Cobalt Strike payload.