# Talk to your children about Payload Staging

*Wednesday 22 June, 2016*

Time to time, I find myself in an email exchange about payload security and payload staging. The payload security discussion revolves around Beacon's security features. Once it is running on target, Beacon takes steps to authenticate its controller and establish a session-specific key to decrypt tasks and encrypt output. I discuss these security features at the end of the Infrastructure lecture in Advanced Threat Tactics. Questions on this topic are usually easy to field.

Payload Staging is a different animal though. Payload Stagers are tiny programs that connect to a controller, download a payload, and run it. Payload Staging is helpful to pair large payloads (e.g., Beacon, Meterpreter, etc.) with attacks that have size constraints. The payload ***stagers*** in Cobalt Strike do not authenticate the controller or verify the payload they download. Questions on this topic usually spawn discussion.

In this post, I'll explain why Cobalt Strike's stagers are the way they are. I'll also discuss ways you can adapt your use of Cobalt Strike to limit payload staging over a hostile network.

## The Threat Model

People who ask questions about staging and payload security have a threat model in mind. They assume that there is an actor present in the communication path between their targets and their Cobalt Strike controller. They also assume that this actor has the ability to observe and manipulate data that traverses this communication path.

This is a fair assumption. A traceroute between a target system and an externally hosted Cobalt Strike team server will yield many systems that you and your customer do not control.

A malicious actor, present in your payload communication path, could man-in-the-middle the staging process and deliver their payload stage to your target. This would give the malicious actor access to your target. That's no good!

A payload stager could mitigate this problem in one of two ways. The stager could authenticate the server that hosts the stage. Or, the stager might take steps to verify that the stage it receives is the one you intended to send.

Simple enough. Why don't Cobalt Strike's stagers authenticate their controllers or verify the payload stage after staging completes? Let's discuss that.

## Size Matters

My first excuse to deflect this discussion is size. The larger a payload stager becomes, the fewer attacks you can use it with. This limits our ability to stick security features into a payload stager.

The above statement is true, but the excuse is somewhat thin with my product. Why? Cobalt Strike isn't an exploit development framework. Cobalt Strike does have size constrained attack vectors, but the size constraints I deal with are nothing like what you'd see in an unforgiving memory corruption exploit.

Ok, if that's all true, then why does Cobalt Strike use small stagers with no security features? Cobalt Strike uses these stagers to stay compatible with the Metasploit Framework. Remember, until 3.0, Cobalt Strike was a collection of features built on top of the Metasploit Framework. The easiest way to make Beacon work with the Metasploit Framework's exploits and modules was to stay compatible with the Metasploit Framework's staging process.

Even with Cobalt Strike as a separate platform, this compatibility with the Metasploit Framework has its benefits. For example, it's quite easy to use a Metasploit Framework exploit to deliver a Cobalt Strike Beacon payload. Just set PAYLOAD to a Meterpreter payload that uses the same stager, point LHOST/LPORT at Cobalt Strike, and fire away!

## The Chicken and the Egg

You may not buy the size excuse, that's fine. Earlier, I mentioned that the purpose of payload staging is to pair a large payload with a size-constrained attack. Let's assume that we have a stager with amazing security features. Let's also assume that this attack is a file that the target downloads from your controller.

To reach your target, the attack with your secure stager must travel over a potentially hostile network. If we apply the same threat model to the attack delivery process, we find ourselves in a hopeless position. The attacker might choose to replace our secure stager in the attack with another stager that acts according to their wishes. We lose anyways.

This is the chicken and the egg problem. If we can't securely deliver the stager that securely downloads our payload, then how can we trust the stager?

## Sometimes there's a Chicken. Let's protect the Egg.

If you've read this far, you might feel the discussion is getting a little academic. Sure, if an attacker is at the vantage point where they can manipulate the stager, then all bets are off. There are situations where the "chicken and the egg" assumptions don't apply.

Many of my customers work assume breach engagements. These engagements focus on the target's detection and response capability, not their patch management or phishing awareness. In an assume breach engagement, the red team is often given a foothold to work from. It's quite feasible to deliver an attack package over a channel that bypasses our hostile network described above.

If the red team's foothold was setup in a secure way AND the red team's payload is up to the task, then we can assume the red team has a safe channel to work with. In this case, the chicken and the egg problem doesn't apply. The open question is, if these assumptions are in play, what are the best ways to operate with less staging risk? Let's dig into that.

## HOWTO: Reduce Network Staging in Your Operations

Last week, I wrote a blog post about stageless payloads and discussed why you might find this feature valuable. I mentioned that stageless payloads are attractive when the risks of payload staging are not acceptable to your organization. Here's why: A stageless payload artifact contains Cobalt Strike's Beacon payload and its configuration in one file. Stageless payloads don't use stagers. If you can securely deliver and run a stageless payload artifact on your target, you benefit from Beacon's security features right away.

### Access

In assume breach engagements, use a stageless payload artifact to seed your foothold. **Attacks** -> **Packages** -> **Windows EXE (S)** exports a stageless payload artifact. Cobalt Strike has several stageless payload artifact options. You can export Beacon as an executable, a DLL, a service executable, a PowerShell script, or shellcode. One of these options is bound to work for your target. Once a stageless Beacon is on target, you have a (presumably) secure channel to work with.

Staging isn't just initial access though. Staging is a very convenient tactic and it's present in many post-exploitation workflows. Multiple toolsets use staging in their session passing, privilege escalation, and lateral movement workflows. With some adjustments, it's possible to perform these actions without staging a payload over a hostile network.

### Session Passing

Let's start with session passing. This tactic is a way to use an active payload to spawn another payload. Beacon's **spawn** and **inject** commands are designed to pass sessions via stagers. It's possible to pass sessions in Cobalt Strike without staging. Go to **Attacks** -> **Packages** -> **Windows EXE (S)** and export a **raw** stageless payload artifact. This file is essentially a large-blob of shellcode that contains the Beacon payload. Use the **shinject** command in Beacon to inject this shellcode into a process and run it. You can export and use shinject with the x86 and x64 raw stageless payload output.

### Privilege Escalation

What about privilege escalation? Beacon's **elevate**, **spawnas**, and **bypassuac** commands target a listener. These commands do not have alternatives to work with a stageless payload artifact. How do we avoid the risks of staging an elevated payload over a hostile network? Create a listener for Cobalt Strike's SMB Beacon payload. For actions on the local target, this payload will use a stager that sets up a listening socket bound to localhost. The Beacon, running on the target, will then connect to that stager's listening socket, send the payload stage, and the stager will clean itself up and run the stage. The stager and the stage communicate through Beacon's existing communication channel. If you're worried about the risks of staging an elevated payload over a hostile network, this is one way to work. Bonus: the SMB Beacon is the preferred payload to use with Beacon's privilege escalation features anyways.

Be aware, there is a potential race if an adversary is co-habitating with you. An adversary, on the same system, might connect to the stager socket first and elevate their payload instead of yours. This isn't part of the original threat model that provoked this discussion. 🙂

What about privilege escalation based on misconfigurations? For example, you might find that you have an opportunity to elevate via a service with weak permissions. One way to take advantage of this is to drop a service executable to disk, reconfigure the service, and start it with your executable. How do you avoid staging in this case? Use a stageless windows service exe artifact.

### Lateral Movement

How about lateral movement? You have options here too. Cobalt Strike's **psexec**, **psexec_psh**, **winrm**, and **wmi** commands each depend on payload stagers. That said, you do not have to use this built-in automation for lateral movement. I do most of my lateral movement by exporting a stageless payload artifact, dropping it to an intermediate session, and using built-in Windows capability to copy the artifact to my target and run it. This plays very well with Cobalt Strike's ability to steal tokens and create tokens from various types of credential material.

What are options to run a payload on a remote system? Look at PowerShell's Invoke-Command, wmic, sc, schtasks, and at. There's more, but this is a good starter set of options. I wrote a blog post with various lateral movement recipes a few years ago. The material is still relevant. If you use stageless payload artifacts for lateral movement, you can avoid staging a payload over a hostile network.

### Update 8 December 2016

Cobalt Strike 3.5.1 added a **host_stage** option to Malleable C2. Set this option to false and Cobalt Strike will not host a public HTTP/HTTPS/DNS stage. This option will also remove HTTP/HTTPS/DNS stagers from Cobalt Strike's workflows.

## The Talk

Some of you will read this post and scratch your head. "What is Raphael talking about? I stage payloads all day long, and I like it". The point of this post is twofold.

First, I hope to get you thinking about your tools, how they work, and the potential risk associated with that. This will allow you to evaluate the risk and make the best decision for your situation. You may decide that the one-off risks of payload staging are worth the benefits. That's fine.

You may decide otherwise though. This gets to the second point. I have customers who care a great deal about this particular risk. They adjust their operations to work without it. This post is my opportunity to disseminate best practices for future and existing customers with this particular need. If this is you, I hope you found this post useful.

## Epilogue: What about TLS Certificate Pinning?

This section doesn't fit into the narrative of this post, but I field questions on this too:

Earlier, I mentioned that one way to add protection to the staging process is to authenticate the staging server. Last year, the Metasploit Framework gained an optional HTTPS stager that does this. This stager ships with the expected hash of the staging server's SSL certificate. When the stager connects to the staging server, it checks the server's SSL cert hash against the value it expects. If they don't match, it doesn't download and act on the payload. If they do, it assumes things are good. Pretty neat, right? Ignoring the chicken and the egg problem, this is a way to solve this problem for one protocol.

Occasionally, I get asked, "Raphael, why don't you add this to Cobalt Strike?" While I think this technique is interesting, I don't feel this is the right approach for Cobalt Strike. Here's why:

This technique applies to only one protocol: HTTPS. The HTTPS Beacon isn't as heavily used as other Beacon options. The HTTPS Beacon's default self-signed certificate is likely to stick out like a sore thumb. It's possible to bring a valid certificate into Cobalt Strike, but this is a barrier to fully benefiting from the HTTPS Beacon payload. My customers tend to rely on the HTTP Beacon much more. Thanks to Malleable C2 it's easy to disguise your Cobalt Strike HTTP traffic to look like something innocuous. The HTTP Beacon isn't "less secure" than the HTTPS Beacon either. After staging, Beacon's payload security features are in effect, no matter which data transport you use. If I were to tackle the problem of secure staging, I'd rather focus on an implementation that is transport agnostic.

This verification technique relies on an API specific to WinHTTP. Windows has two APIs for easily making HTTP requests: WinHTTP and WinINet. The Beacon payload and its stagers use WinINet for communication. This is the preferred option for desktop applications. The Metasploit Framework offers the WinHTTP stager (with the verification option) and WinINet stagers with no verification built-in. The Metasploit Framework offers both options because there are certain types of proxy servers that the WinHTTP APIs don't do well with. Consult The ins and outs of HTTP and HTTPS communications in Meterpreter and Metasploit Stagers for more on this. I don't believe the benefits of this choose-the-right-stager-API approach outweigh the complexity it would add to Cobalt Strike. Again, this is an opinion specific to my product and user community.