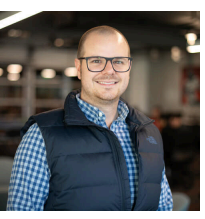
red canary

RESOURCES • BLOG

THREAT DETECTION



Detecting attacks leveraging the .NET Framework

The .NET framework includes rich offensive capabilities that adversaries aren't yet using, but we've been thinking about detection anyway.

ZAC BROWN • SHANE WELCHER

Originally published January 22, 2020.
Last modified April 30, 2024.

The .NET framework is increasingly being used for offensive purposes in the post-exploitation world. Unless you have the visibility into memory that some EDR solutions provide, detecting this activity can prove to be quite a challenge. Even with a solution that provides visibility into memory, sifting through large volumes of data and identifying what API calls are considered malicious can be a daunting task.

A lot of the .NET-related presentations out there revolve around the capabilities of these programming languages, but few address the possible avenues of execution associated with them. This blog is focused on the latter.

If you aren't familiar with C# or F#, both of the development kits for these languages can be obtained from Microsoft via Visual Studio. Within these development kits are two interactive console utilities, better known as `csi.exe` and `fsi.exe`. These utilities allow for full scripting capabilities comparable to a tool like PowerShell. Additionally, you can create script files for both of these languages and execute them from a command line like the one below:

Just so you can see both binaries in action, here are similar executions leveraging csi.exe:

This is especially dangerous, as these binaries are signed by Microsoft and can easily be brought onto a host if not already installed. This "bringing your own land" technique has continued to grow in popularity as we've started seeing it with utilities like wscript. Adversaries copy a renamed instance of wscript over to a compromised host to execute their payloads. Sidenote: if your environment is not tracking internal names within a binary's metadata, then you may be missing additional activity.

As of right now, we haven't seen any malicious usage of these scripting utilities, but the offensive capabilities are there and will almost certainly be exploited eventually. Just to show how easy abusing this functionality can be, here is a screenshot of `fsi.exe` loading SharpSploit within an interactive session and leveraging the `GetHostname` function from the Enumeration module:

For what it's worth, we used SharpSploit to show how quickly this can be abused.

For a competent adversary, these utilities don't require any additional frameworks for exploitation. We used SharpSploit as an example, but the point we're trying to drive home is that an adversary can easily use this offensive technique is to load a random DLL when you have interactive access to the .NET Framework. As simple as it is to load a DLL, interactively adding this code to one of these sessions gives access to the Windows API without having to bring any additional binaries to disk, which is why it is important to better understand how and if these binaries are being utilized within your environment.

Instead of spending too much time going over these languages, over the past two years we've given a presentation on some of **the basics and capabilities of C#**, which we highly recommend you check out.

We also recommend reading the following:

Bypassing application whitelisting by using rcsi.exe
Bypassing application whitelisting by using dnx.exe

Detection Methods

With the intention of bringing this technique to light, we wanted to focus more on detection methodology than on potential offensive capabilities, so that blue teams can better prepare for potential abuse. The following are a few detection techniques we have rolled out into our environments:

- csi.exe or fsi.exe usage without command-line arguments for an indication of interactive sessions
- csi.exe or fsi.exe usage with F#/C# script files being passed as arguments
- Renamed instances of both of these binaries
- Network connections originating from these binaries
- csi.exe or fsi.exe usage from user paths

Conclusion

Of course, there are other detection techniques you can implement but the ones mentioned above are a great start. There are additional scripting environments that are capable of doing the same, but most of the others are not signed by and easily obtainable via Microsoft.

It's also worth noting that the most interesting detection techniques are going to be those looking for malicious .NET behaviors. While you can construct some basic heuristics around csi.exe and fsi.exe usage, the real nefarious stuff will usually be found hunting for malicious .NET behaviours, which isn't dependent on C#, VB .NET, F#, or other .NET-based languages.

RELATED ARTICLES

THREAT DETECTION

Artificial authentication: Understanding and observing Azure OpenAI abuse

THREAT DETECTION

Apple picking: Bobbing for Atomic Stealer & other macOS malware

THREAT DETECTION

Keep track of AWS user activity with SourceIdentity attribute