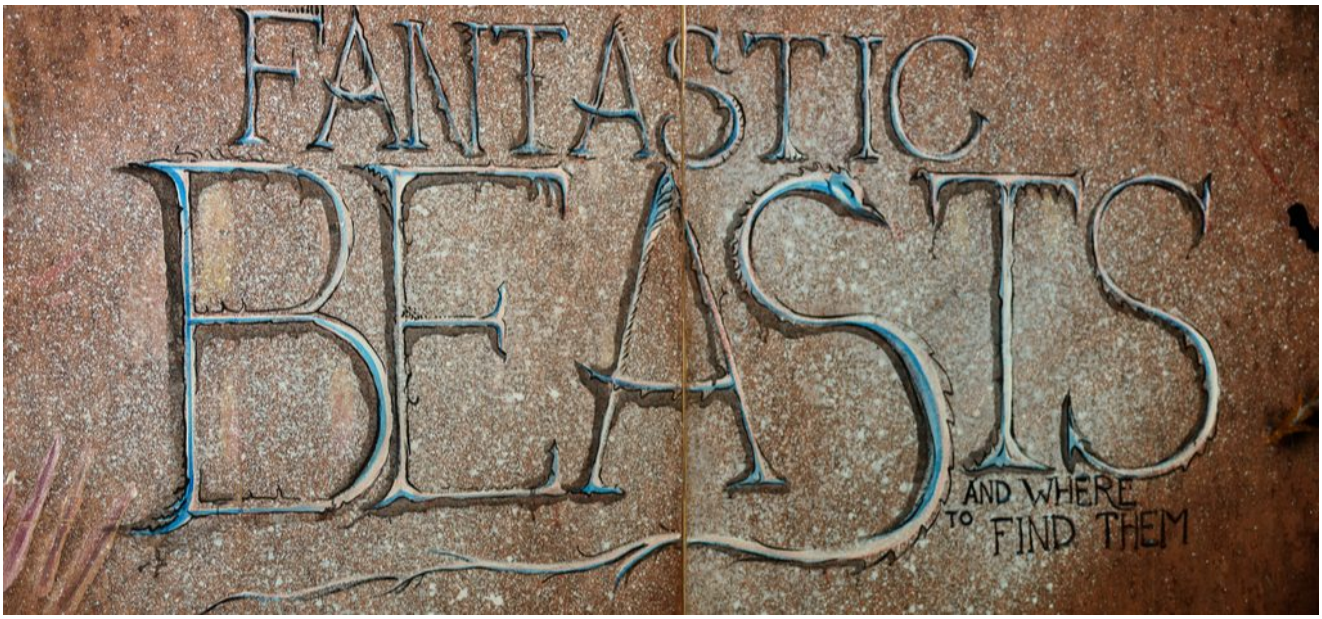
Fantastic payloads and where we find them

intezer.com/blog/intezer-analyze/fantastic-payloads-and-where-we-find-them

March 30, 2020

Written by Michael Kajiloti - 30 March 2020



Attackers have long used evasion features in their malware to avoid detection by security products and analysis systems. One of the most common anti-analysis tricks we have seen in today's Windows malware is the use of **packers**. Packers often complicate the analysis and detection of binary files by hiding the malware's real code and data; often referred to as the **payload**.

The most common solution to overcome packing is the use of **dynamic analysis** (sandboxes) and emulation systems. However, as these solutions have become more mainstream, packers have started to incorporate new **anti-analysis** methods to both detect and evade these systems.

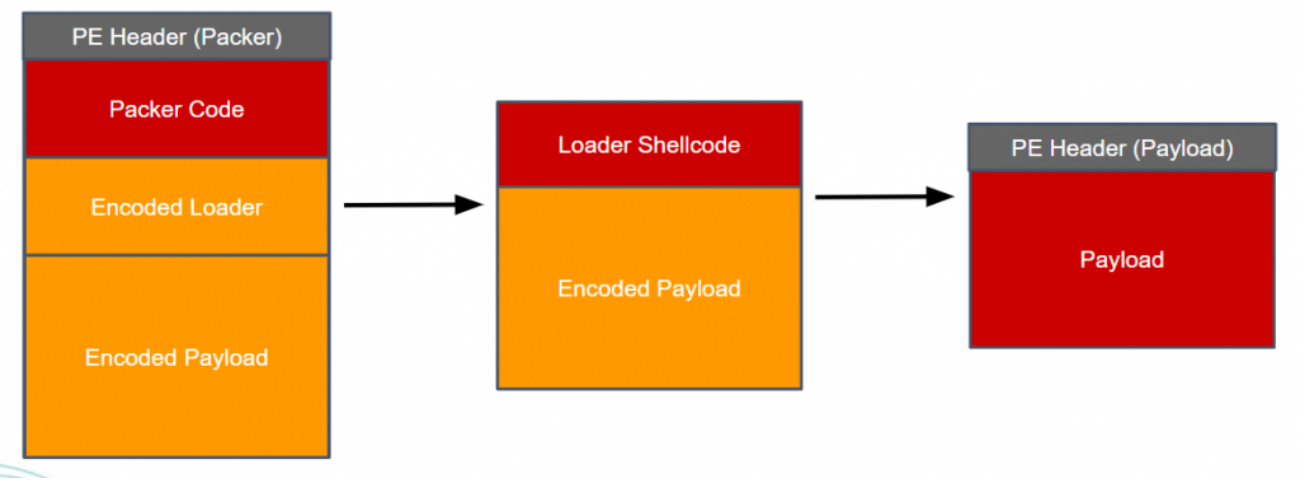
One such anti-analysis method which is becoming increasingly popular is the malformation or complete removal of the Windows PE header from unpacked malicious payloads. This can make both finding the malware payload and properly analyzing its code more challenging.

While removal of Windows PE headers from malicious payloads is not an entirely new method, we are seeing some of the most popular and sophisticated crimeware families fully incorporate this approach into their packers. Below we will present some examples that demonstrate this trend through our Genetic Malware Analysis platform, Intezer Analyze.

On packers and payloads

When we investigate a packed malware file, the malware (payload) itself is encrypted or encoded and therefore we can't access its real code and data. Typically, in order to analyze the malware's code, we must execute the file in a sandboxed environment, let it unpack itself, and then grab the payload from memory.

While various malware and packers operate in different ways, an abstraction of a typical unpacking flow looks like this:



In this flow we can see that the packer will decode and execute a shellcode component that will in turn decode and execute the real payload. This is just a common scenario; some packers will not use a shellcode loader and instead they will unpack the payload directly. It's worth noting that unlike the packer's code, which is designed to be obfuscated and appear random, we have observed that these shellcode components are often reused. This means that these components generally have more value than the packer itself when investigating them with code reuse analysis. Therefore, detecting and analyzing this component's code can be worthwhile for us.

With that said, finding and analyzing the malware module itself is by far the most effective way to properly detect and classify packed malware. The most common way to identify malware payloads in memory is to search for PE header artifacts. Knowing this, many

attackers try to stay undetected by removing these artifacts from the PE Header of the payload, or by using no PE header at all, as we'll see soon in the examples presented in this article.

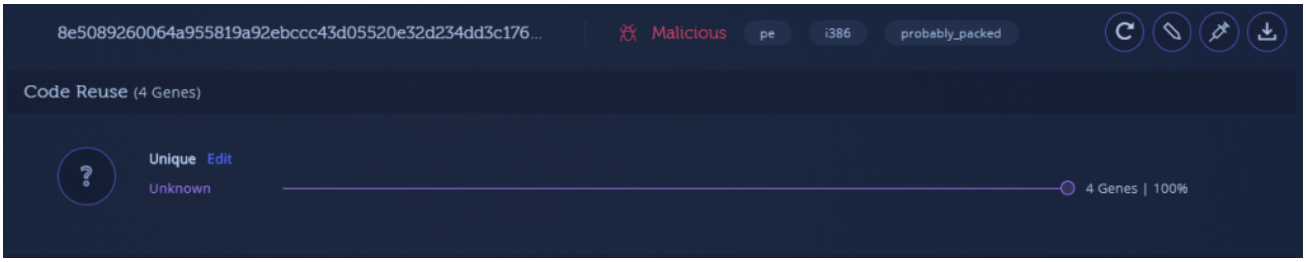
The many faces of Emotet

Emotet is one of the most widespread malware in the crimeware ecosystem which has undergone a number of updates since its inception. In particular, we have noticed that this malware is constantly evolving and adapting, both in the malware code itself, and in the packers it uses. While switching the packer is relatively inexpensive for the attacker and happens frequently, modifying the malware's code is much more costly and happens rarely.

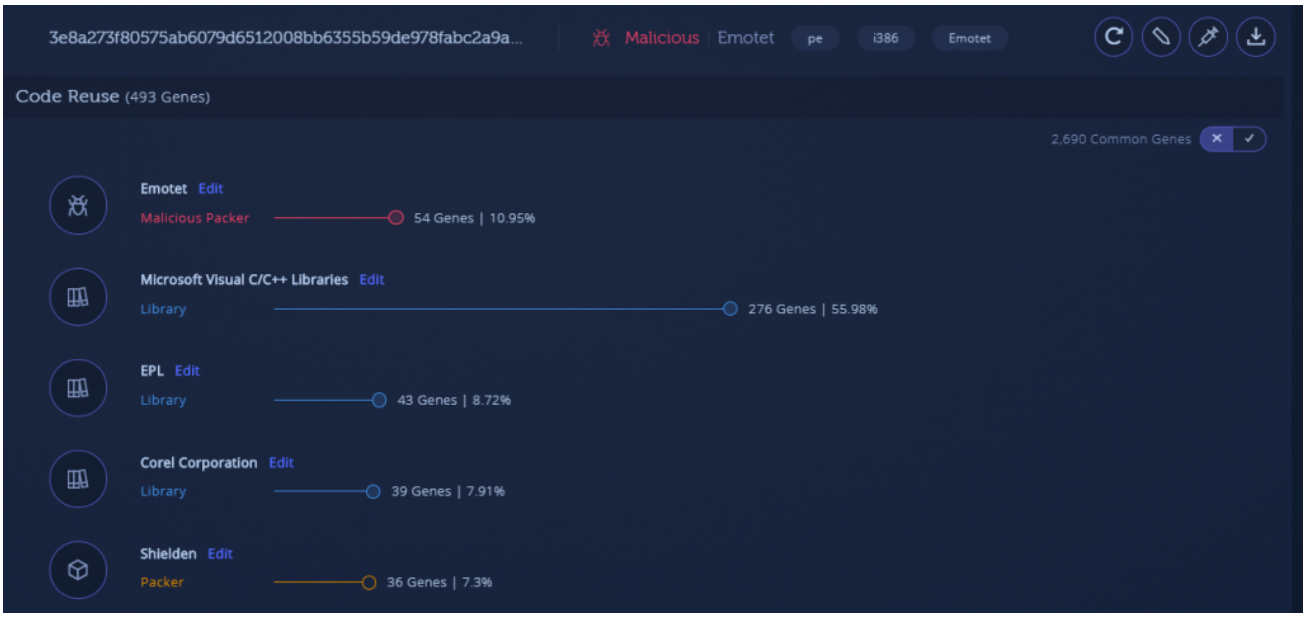
An interesting example of this can be seen when we compare the following Emotet samples:

- Emotet sample from June 2019: 8e5089260064a955819a92ebccc43d05520e32d234dd3c176bed5f6d0665ebdb
- Emotet sample from December 2019: 3e8a273f80575ab6079d6512008bb6355b59de978fabc2a9a66c0ed148b94fef

When looking at the code of the packed files themselves, we can see just how different these samples are, showing no apparent connections to one another:

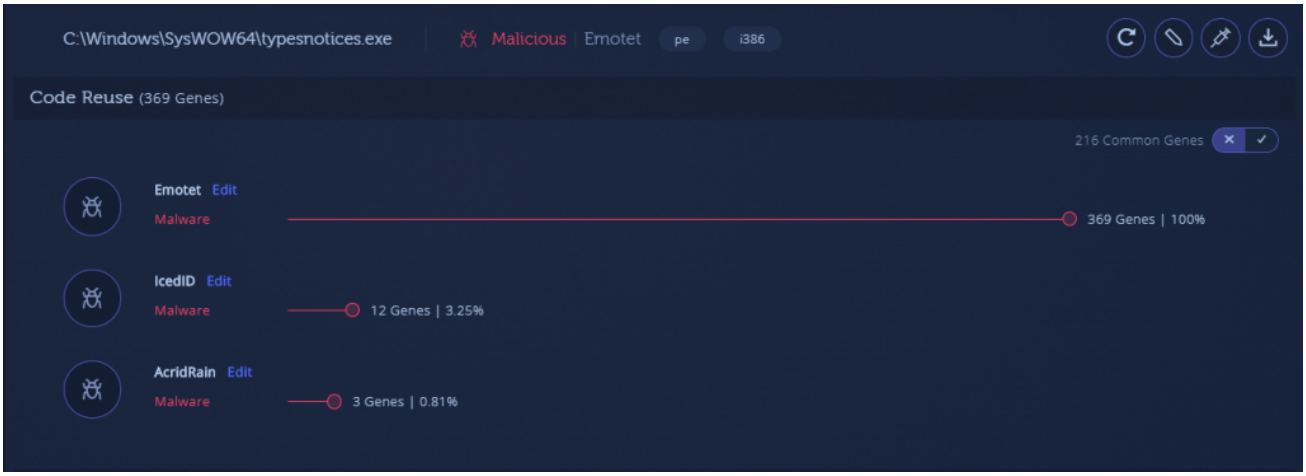


Packer (June 2019) – 8e5089260064a955819a92ebccc43d05520e32d234dd3c176bed5f6d0665ebdb

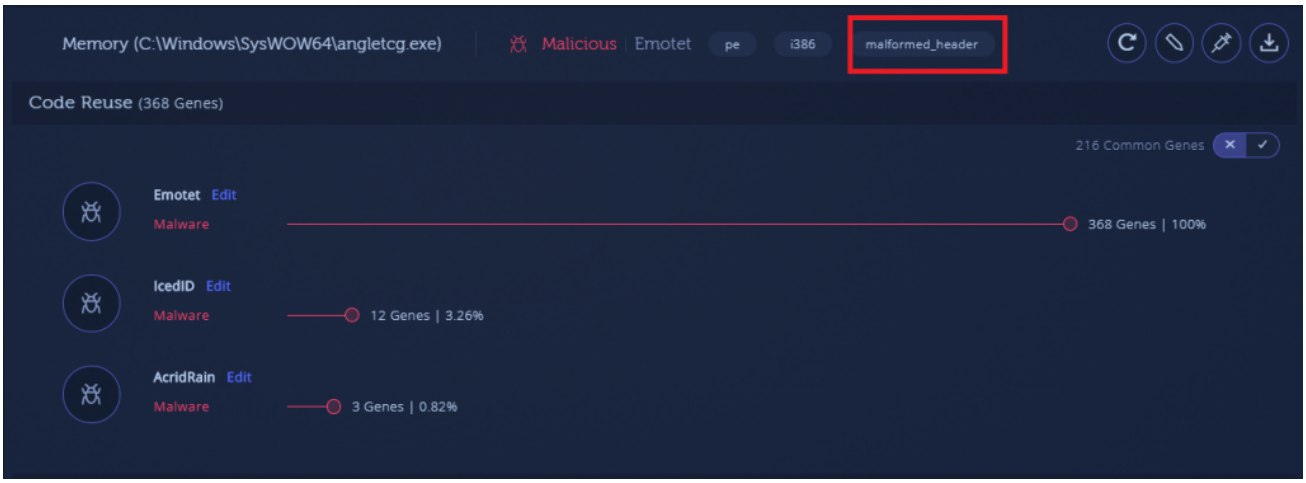


Packer (December 2019) – 3e8a273f80575ab6079d6512008bb6355b59de978fabc2a9a66c0ed148b94fef

But if we look at the unpacked memory module payloads after undergoing dynamic analysis, we can see that they are nearly identical from a binary code perspective:



Payload (June 2019) – 8e5089260064a955819a92ebccc43d05520e32d234dd3c176bed5f6d0665ebdb



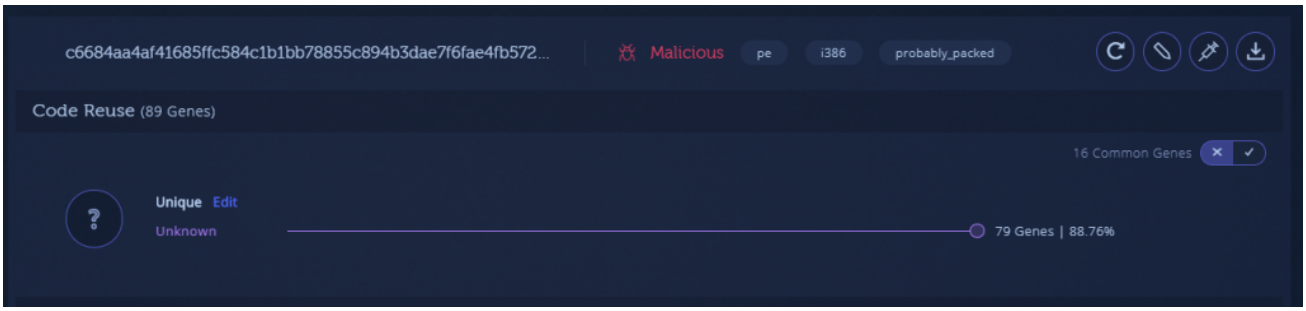
Payload (December 2019) – 3e8a273f80575ab6079d6512008bb6355b59de978fabc2a9a66c0ed148b94fef

There is one notable exception: in the December 2019 sample, the payload memory module has a **"malformed header"** tag. In this case, this indicates that the packer loaded the module with a PE Header that's missing certain PE artifacts, such as the "MZ" magic or other PE Header constants. This is usually done in an attempt to make it more difficult to find these modules in memory, since the most common method of finding PE memory modules is by scanning for these artifacts.

It is worth mentioning that in Intezer Analyze we repair the malformed header as part of our analysis process. Downloading and examining the memory module will reveal that it has a full PE header.

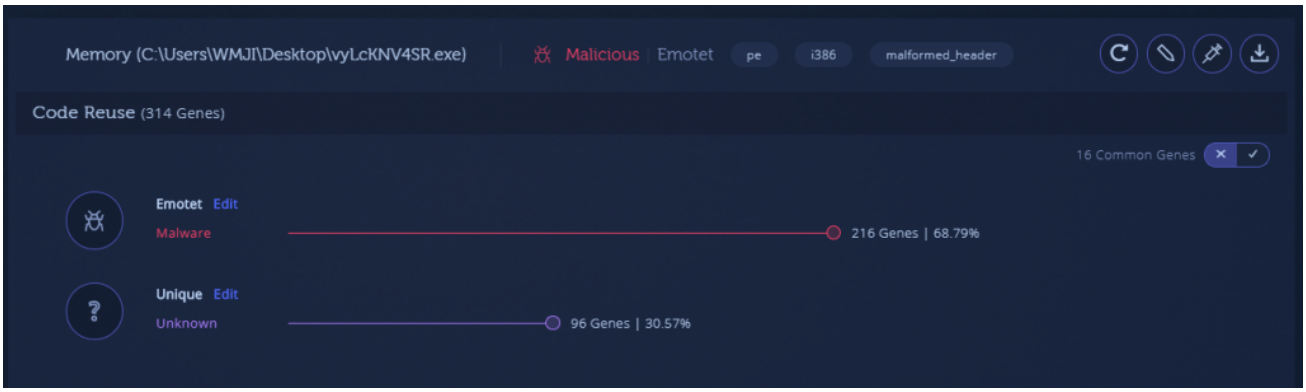
It is also important to note that even in the instance of a constantly evolving threat such as Emotet, when the attacker modifies the code of the malware itself, there are still parts that remain unchanged. Therefore, when Intezer Analyze executes the malware, finds its payload, and then analyzes its code, we can still detect those traces to Emotet's code that have not been modified.

This can be observed when we investigate a recent sample of Emotet from March 2020. Looking at the packer's code, we can see that it is different yet again, and has no relations to any code we have seen previously:



Packer (March 2020) – C6684aa4af41685ffc584c1b1bb78855c894b3dae7f6fae4fb572bc02525102f

However, when we look at the memory module payload itself, we can see that some of the code is new and has been modified from previous versions. Most importantly, a substantial part of the code has been reused from previous Emotet variants that we have seen before:



Payload (March 2020) – C6684aa4af41685ffc584c1b1bb78855c894b3dae7f6fae4fb572bc02525102f

When less is more

While using a malformed PE Header is a relatively simple trick, using no PE header at all is a more sophisticated and complicated affair. It takes more than just removing a few unused magic constants from the header in order to conceal it. This method requires loading and executing the payload just like shellcode.

When malware authors write and compile the malware payload itself, it is usually compiled and built into the form of standard PE files, not position independent shellcode. Directly converting a sophisticated malware's codebase such as Dridex or TrickBot to run like shellcode can be complicated.

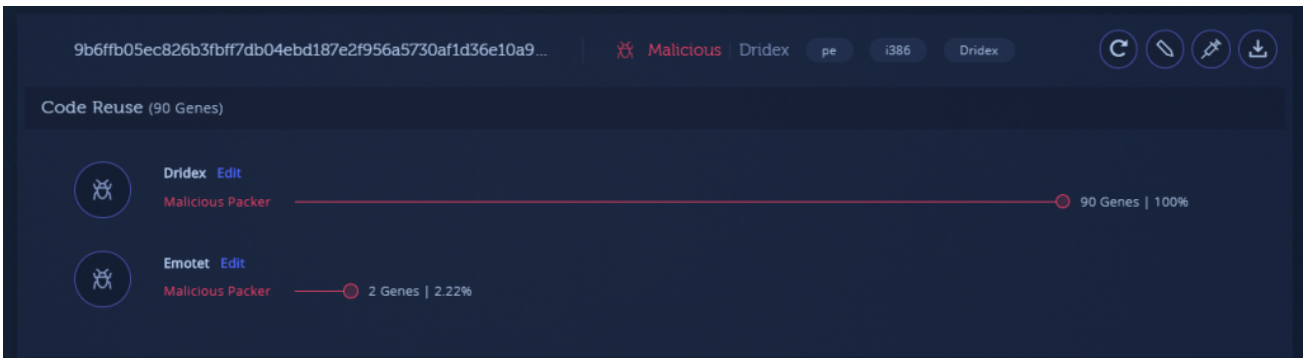
Most malware authors choose a simpler option, using a packer that unpacks and loads the payload module without using its PE Header at load time. These packers typically store the relevant information from the PE header, such as API import and relocations data, in a custom structure during packing time. The packer then uses that structure during the unpacking process in order to properly load the malware code.

Same Dridex, new tricks

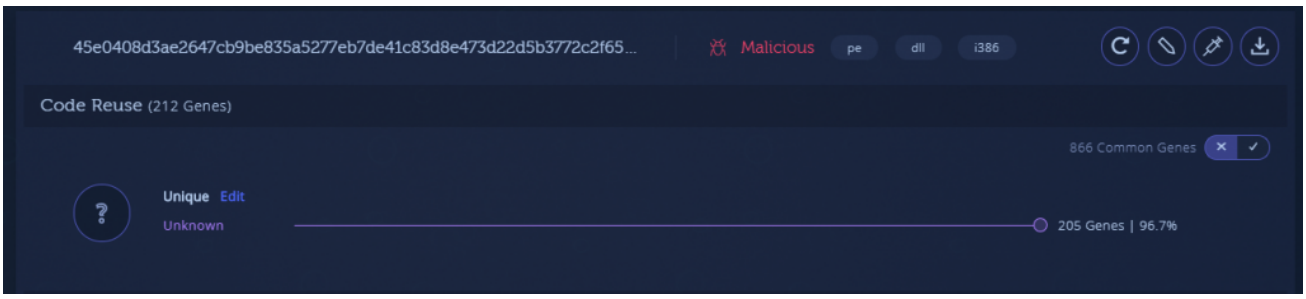
An example of a popular malware that has recently adopted such a packer is the Dridex banking trojan. Let's compare two recent Dridex samples from February 2020:

- 9b6ffb05ec826b3fbff7db04ebd187e2f956a5730af1d36e10a9f56ee1bb767a
- 45e0408d3ae2647cb9be835a5277eb7de41c83d8e473d22d5b3772c2f65305fb

Similar to Emotet, these packed files are very different from one another at the code level, showing no resemblance:

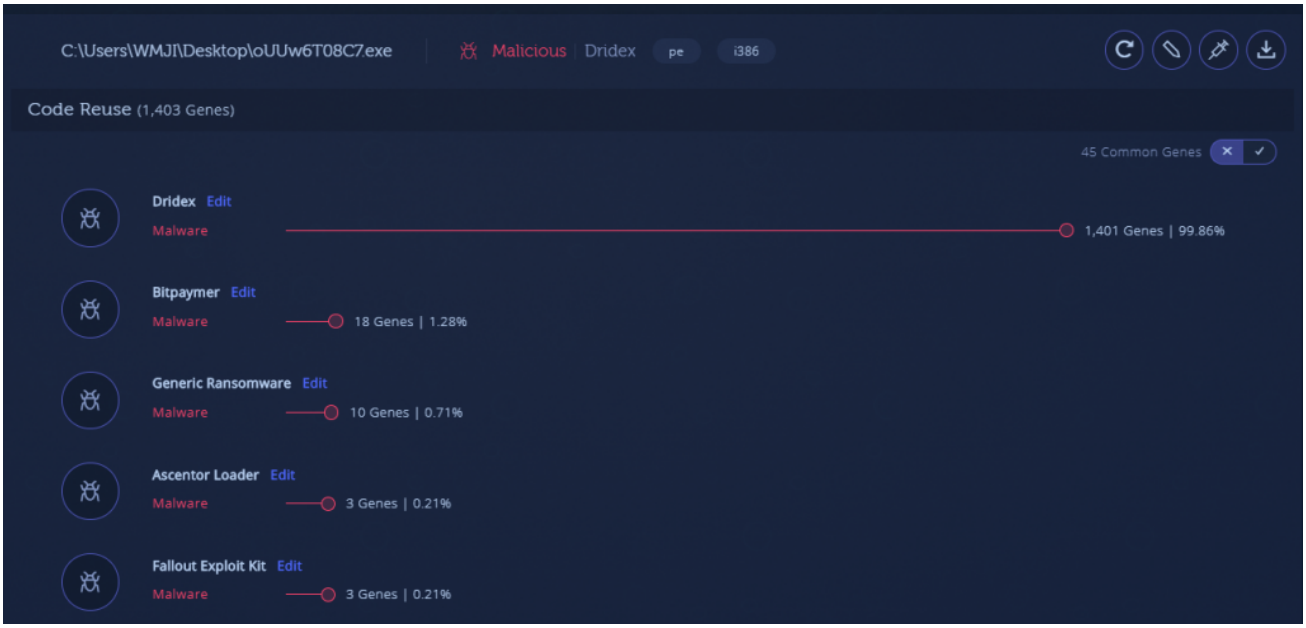


Packer – 9b6ffb05ec826b3fbff7db04ebd187e2f956a5730af1d36e10a9f56ee1bb767a

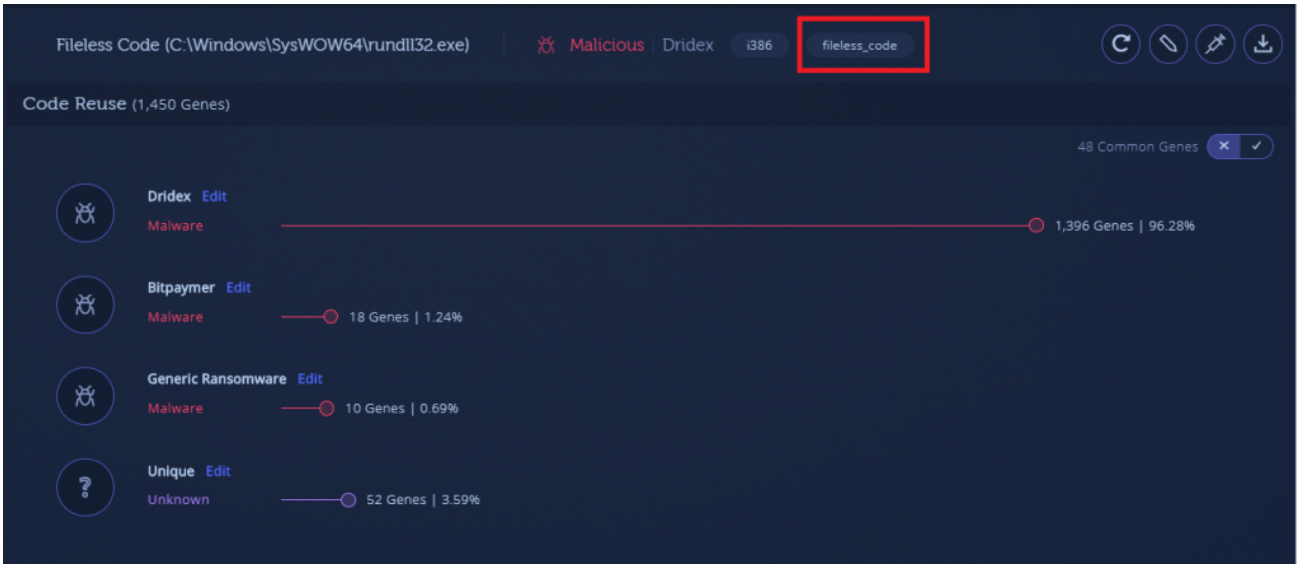


Packer – 45e0408d3ae2647cb9be835a5277eb7de41c83d8e473d22d5b3772c2f65305fb

However, when we look at the payloads themselves, we can see again that they are actually very similar:



Payload – 9b6ffb05ec826b3fbff7db04ebd187e2f956a5730af1d36e10a9f56ee1bb767a



Payload – 45e0408d3ae2647cb9be835a5277eb7de41c83d8e473d22d5b3772c2f65305fb

There is one notable exception here as well, but in this case it is much more substantial. In the second payload we can see the **"fileless_code"** tag, which indicates this module has no PE header. Downloading and examining this module will make it known that it is indeed a binary file with no structure.

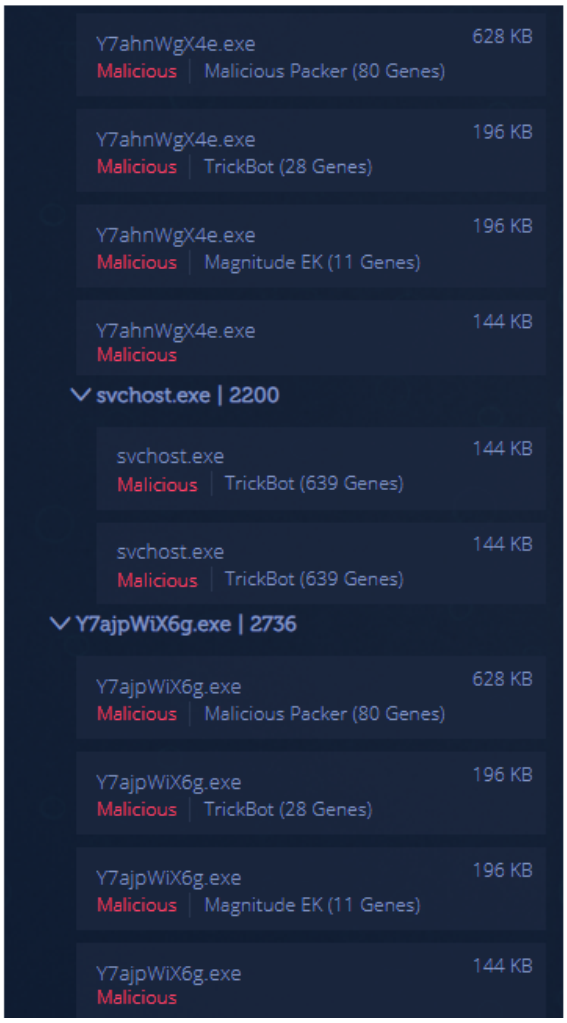
Intezer Analyze detects these modules during dynamic analysis and analyzes their code even though no PE Header is present. It will also detect any other shellcode pieces that are used by the malware.

Look ma! No heads!

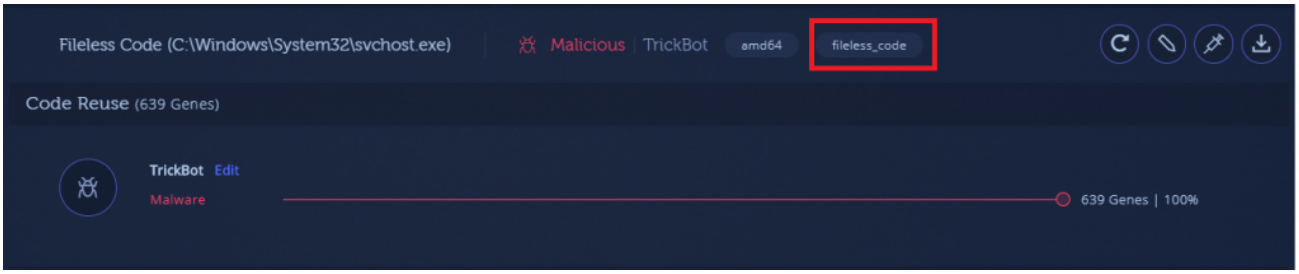
To demonstrate how prevalent this trend is, let's look at the analyses of recent samples of the Ursnif and TrickBot malware. When looking at the dynamic analysis modules, we see several different malicious modules that are detected and analyzed:

Ursnif

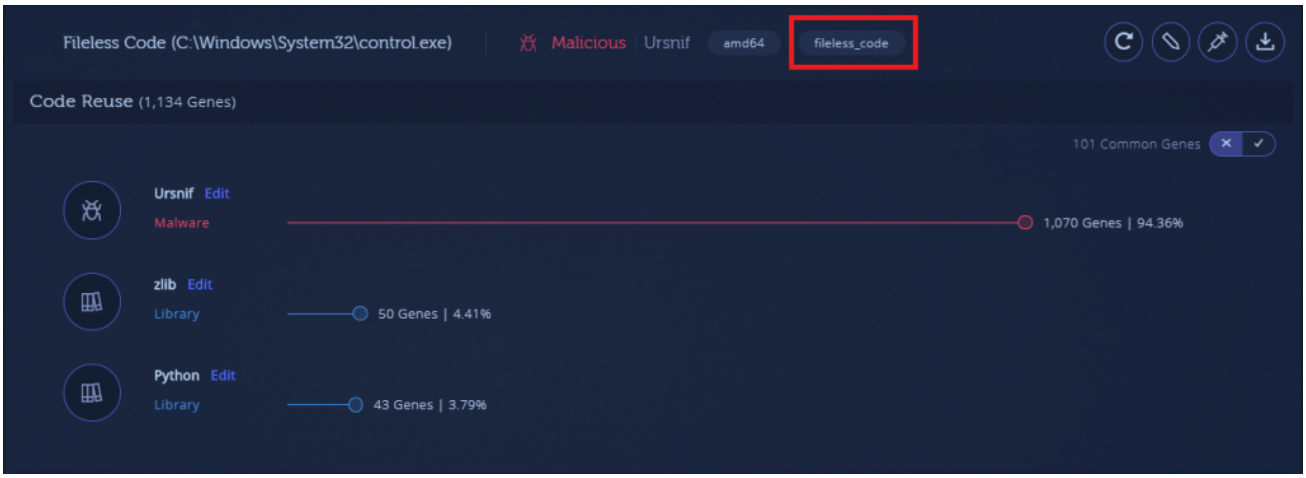
Trickbot



If we review them more closely, we will see that all of them — minus copies of the original packer file — have the **"fileless_code"** tag, indicating that they are all shellcode-style modules with no PE header:



TrickBot – 26d61225f07e39d9e35880631c5043626eb8e3846a526a44015d3a5d03642d55



Conclusion

The use of pure shellcode and shellcode-style components in modern malware seems to be a growing trend, one that is being adopted by some of the most successful malware developers. At the same time, when we identify and analyze these components using code reuse analysis, we can see that they are mostly just a different version of the same thing. We encourage you to analyze your files using the free Intezer Analyze community edition. You might detect malicious payloads that you didn't expect to find!

Michael Kajiloti

Michael is a security researcher turned product manager, who previously led the Intezer Analyze product lifecycle. He has presented his malware research at conferences such as REcon and the Chaos Communications Congress (CCC).