Malware Monday: Obfuscation

Matt B Follow
Dec 20, 2016 · 6 min read

For today's post, I'm going to look at a couple common obfuscation techniques. A large amount of malware utilize some sort of obfuscation to help make both automated and manual analysis a bit more difficult. As endpoint security products are getting "better" at detection, attackers and malware authors are equally getting better at hiding their code.

Another consideration for DFIR analysts is to also think about how attackers utilize malware to hide their *output* as well. I sometimes see a focus on malware obfuscation to the point that the attacker's goals are forgotten and data is removed from the environment before being detected.

Let's examine a couple of techniques that, while may seem dated, are still actively used.

XOR

Exclusive-OR, or "XOR", is an operation that returns a value dependent on only one bit being set. This is often easier explained with an example. Here's a simple example using Python:

```
>>> bin(5)
'0b101'
>>> bin(6)
'0b110'
```

For this XOR, we're comparing 101 vs 110 (0b is a prefix for a binary literal). Look at the two numbers and notice the positions where we have a one and a zero. Here's a step through, from left to right. We're comparing `101` and `110`.

- Position 1 is two ones. Our output is zero.

- Position two is a zero, and a one. Our output is one.
- Position three is a one and a zero. Our output is one.

Combined output: `0b011, or 3.` Congrats, you just did XOR in your head! The symbol ^ is often used to express XOR, and sure enough, utilizing Python to show this returns us our expected value:

```
>>> 5^6
3
```

Attackers like XOR because it is symmetric — the same key used to encode the data is also used to decode the data — and very easy to execute. They can create their own "key" of sorts, with minimal mathematical overhead, that may be tough for tools and/or analysts to guess. *Note that this theory generally applies to single-byte XOR encoding. There is a possibility of using larger "keys", which make it more difficult to decode the data via brute-forcing.*

XOR In Use

One of the reasons I brought XOR up is because I still see a lot of attackers utilizing it as their output encoder of choice, or as part of output encoding. Earlier this year, I was part of an investigation where FrameworkPOS, or TRINITY, was being used to scrape payment card data from memory and store it in a local file. Take a look at the encoding routine the malware goes through:

Screenshot of malware walkthrough from Trustwave's article on FrameworkPOS

Why use substitution + XOR to encode output data? Well, for starters, they need to evade any detection mechanisms on the infected system. Payment card data fits within a well-known regex, but obfuscated payment card data just looks like blobs of data. Second, by writing the encoding routine, attackers have also already written the decoding routine (remember symmetry).

Base64

Another well-known encoding technique utilized by attackers — as well as legitimate applications — is Base64. Without getting into too much history, Base64 is from the MIME standard, which recognized the need for converting binary to text for email attachments. Base64 has a set of only 64 characters (hence the name!), and a standard for translating data within this limited set. The MIME Base64 “alphabet” looks like this:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=
```

Note that due to Base64 being a smaller set of characters, encoded data is often "longer" than encoded data. Typically, we should expect an increase of about 33% (or 4 Base64 encoded characters for every three decoded characters), give or take. Furthermore, attackers can also define their own Base64 alphabets, which make standard conversion techniques useless.

Here's an example of Base64 in use:

```
$ echo http://www.evil.com/beacon+trouble | base64
aHR0cDovL3d3dy5ldmlsLmNvbS9iZWFjb24rdHJvdWJsZQo=
```

If you've written an indicator to look for `evil.com`, you'll have a hit. But searching for a Base64 string would be a silly indicator to construct, unless the value was static. Furthermore, due to the way that Base64 is translated, `evil.com` and `a.evil.com` are starkly different. Have a look:

```
$ echo evil.com | base64
ZXZpbC5jb20K
```

```
$ echo a.evil.com | base64
YS5ldmlsLmNvbQo=
```

Base64 In Use

I see a LOT of malware authors and attackers utilize Base64, primarily because, again, it's often a built-in encoding/decoding routine and evades detection. It's getting a LOT of use in PowerShell right now. For example, if you run `powershell.exe` and specify `-encodedcommand` (or one of its many shortened versions), this is actually a parameter to indicate that the following Base64-encoded string is being passed to PowerShell for execution. PowerShell also has a built-in `FromBase64String` decoding method, if you need to decode a string and manipulate it further.

To help illustrate the point, take a look at the following example payload from a Recorded Future threat intel report, that analyzes PowerShell scripts being stored in Base64 on code repository and pasting sites. Here's an example from the beginning of the report:

Screenshot of encoded PowerShell command from a Recorded Future Threat Intel report

Note that we have some obviously plaintext (such as powershell.exe, creating a New-Object, and IO.MemoryStream), the overall encoded text is not readable by humans. The decoded text actually contains *additional* Base64-encoded strings, establishing Inception-level obfuscation.

I'd recommend digging into that report to understand a bit more about how the threat actors are abusing multiple scripting languages and Base64 encoding to infect their targets and evade detection. Additionally, there are some excellent links that contain malicious Base64 text still live on Pastebin.

The use of encoded PowerShell has grown exponentially in recent years. Attackers and researchers have developed multi-layer techniques obfuscation. I'd highly recommend checking out Daniel Bohannon's talk at

DerbyCon 2016 from this past year, where he released the first version of his awesome obfuscation tool. I'll let the video do the job of displaying the power of the tool, and just how scary it can get.

Looking for Obfuscation

Looking for obfuscation is itself an analysis paradox as the point of obfuscation is to make something harder to identify. However, there are some helpful hints that DFIR analysts and threat hunters can keep in mind:

1. Be cognizant of how Base64 text is handled by various platforms. If you have the ability to do host introspection and/or collect command-line parameters, look for key words such as "FromBase64String", "encodedcommand", or "enc". Popular penetration testing tools such as MetaSploit and PowerSploit utilize encoded commands that, when examined in the context of an enterprise, often stick out as highly suspicious.
2. Trying to detect Base64 on a network or within a file can be tough — especially if you expect to write a string for *any combination of characters within an alphabet of 64 characters*. Instead, look for signs that malware

may utilize Base64 (such as an alphabet within the binary's strings) or actual encoded strings.

3. See if you can find static translations in Base64-encoded strings. Here's an example:

```
$ echo http://www.evil.com | base64
aHR0cDovL3d3dy5ldmlsLmNvbQo=

$ echo http://www.malware.com | base64
aHR0cDovL3d3dy5tYWx3YXJlLmNvbQo=
```

Note that we have two different sites, but can pivot off of the static translation of `http://www.`

1. In the XOR section, I mentioned attackers saving stolen data in a substituted, XOR'd format. When thinking of how to detect such a file (without any previous indicators), instead focus on whether a file of seemingly-random data belongs in a particular location. For example, RawPOS threat actors often like to store stolen XOR'd data in C:\Windows\system32 with a DLL file extension. While a simple file

listing may show a group of DLLs, a `file` command would identify a file as a blob of data, which does not fit in. Blobs of data are not irregular on a system; they are irregular in locations such as C:\Users\Public or Temp folders or system locations. For example:

See the malware?

See the malware now?

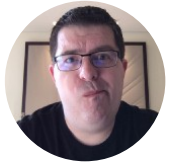
Wrap-Up

Today's post took a look at some simple obfuscation techniques that are still widely-used by malware authors and attackers. I often get questions from students about these types of obfuscation found within malware and used by attackers. While it is important for analysts to understand the capabilities, I think it is more important to think of ways to identify obfuscated data.

Until tomorrow, Happy Forensicating!

Malware Monday | Zeltser Challenge | Dfir | Malware | Obfuscation

17 claps



WRITTEN BY

Matt B

You don't know my mind, You don't know my kind. Digital forensics is part of my design.

Follow

Write the first response

More From Medium

Related reads

Why Taking a Technological Step Backwards May Better Protect America's Critical Infrastructure...

Ian Barwise
Jul 30, 2018 · 8 min read...

41

Related reads

Trickbot — a concise treatise

Vishal Thakur
Apr 4 · 12 min read ★

6

Also tagged Obfuscation

Creating an Obfuscator for Javascript files

Uddhav Navneeth
Jun 14 · 5 min read ★

36