Fifty Shades of Malware Strings

medium.com/@tom_rock/fifty-shades-of-malware-strings-d33b0c7bee99

Thomas Roccia July 27, 2020

Thomas Roccia
Follow
Jul 24, 2020

.

9 min read

When analysing malware, string extraction is one of the first things to do to briefly extract useful information such as IP address, domains, functions, data, or any other information that has not been removed by the developer.

A string is a sequence of characters, it can be a constant or a variable. A string is a type of data and can be implemented as a byte (or word) data structure that stores a sequence of elements. Strings are inherent in computer programming, they are basically valuable data that is used to create software and, in turn, to get clues about functionality during reverse engineering.

In threat intelligence, using strings to detect a piece of malware with Yara is also a powerful method to scan for new malware and detect threats. This is also very useful when tracking a group of attackers.

In the previous post “Fifty Shades of Malware Hashing”, we discussed the hashing mechanisms used to identify and classify malware. In this article, we will explore the process of extracting strings in malware, understand how it works, and how you can leverage it in your daily analysis.

Character Encoding

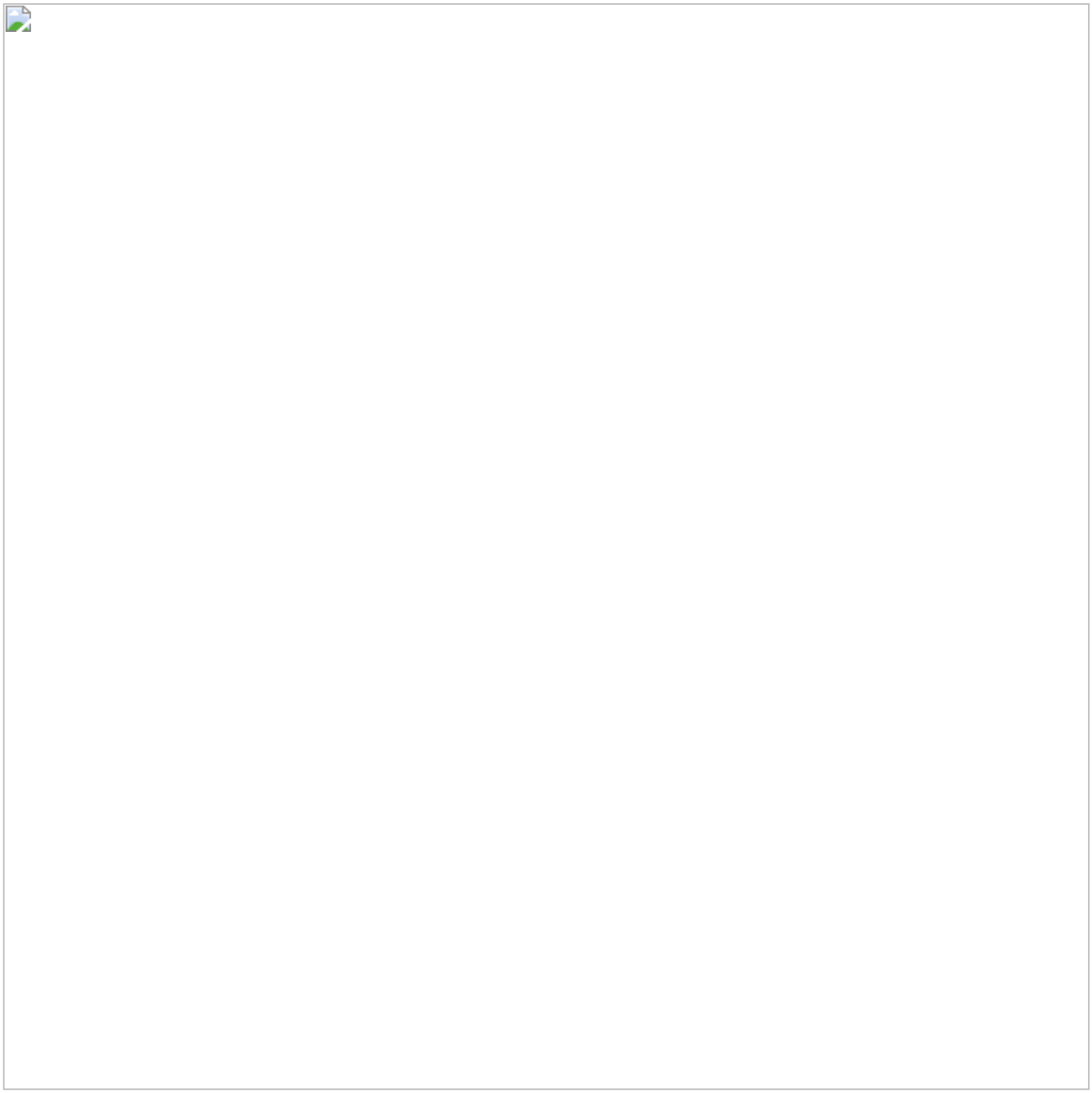
In real life there are several languages and they are written with different characters/alphabet. Computers were originally programmed to work with the English language without accents or symbols. But later, modern computer quickly spread and reached other countries and hence the need to use other languages with different characters such as “œ” or “é” in French, the “ñ” in Spanish or other characters used in Greek, Cyrillic, Arabic, Korea, Chinese, Japan and many others…

To handle this different semantics, computers must use a table that stores this information in order to have the correct encoding.

ASCII

American Standard Code for Information Interchange (ASCII) is a character-encoding scheme and was the first character encoding standard. It represents English character as numbers between 0 and 127.

The below table shows the ASCII table.



ASCII Table

Each character, number, or special character is represented by a 7-bit number. Most modern programs still use ASCII and support additional encoding.

ANSI later extended it to 8 bits and there are several different code pages for symbols 128 to 255. It is basically an extension of the ASCII character set in that it includes all ASCII characters with additional codes of 128 characters.

Unicode

To handle all the languages of the world, Unicode was created. It is a standard for character encoding. Unicode also supports ASCII and essentially extends the character encoding for each language.

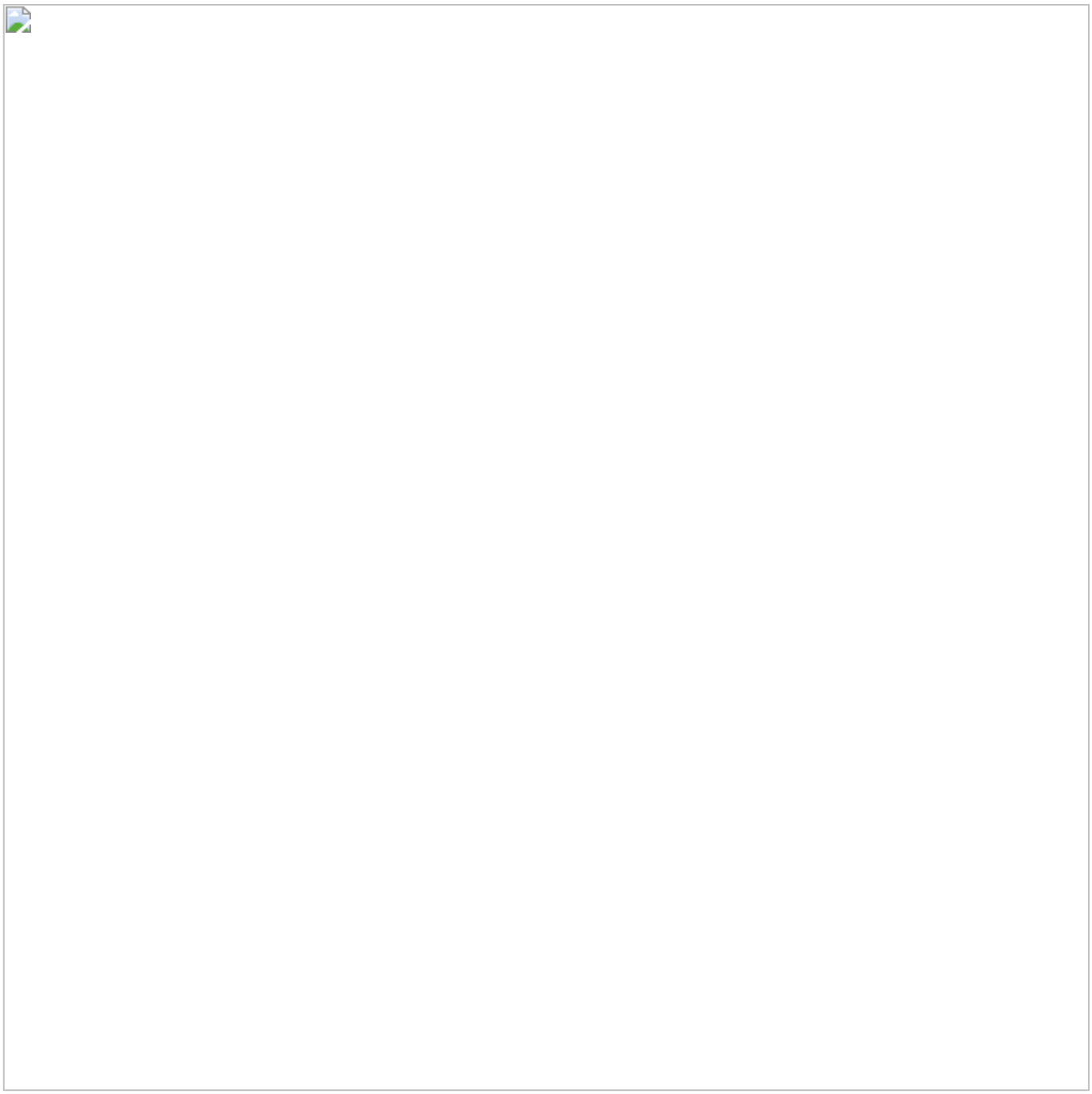
It basically defines a huge table of 1 114 112 code point, which can be used for all kinds of letters and symbols. All languages are currently references using code pages.

The full table of Unicode can be found here: https://home.unicode.org/.

Unicode Transformation Format (UTF)

Unicode assigns a unique number (code point) to each character. To map characters to a code point the Unicode standard includes UTF-8, UTF-16, and UTF-32.

The three methods are simply different ways of representing Unicode codes with different byte sizes.



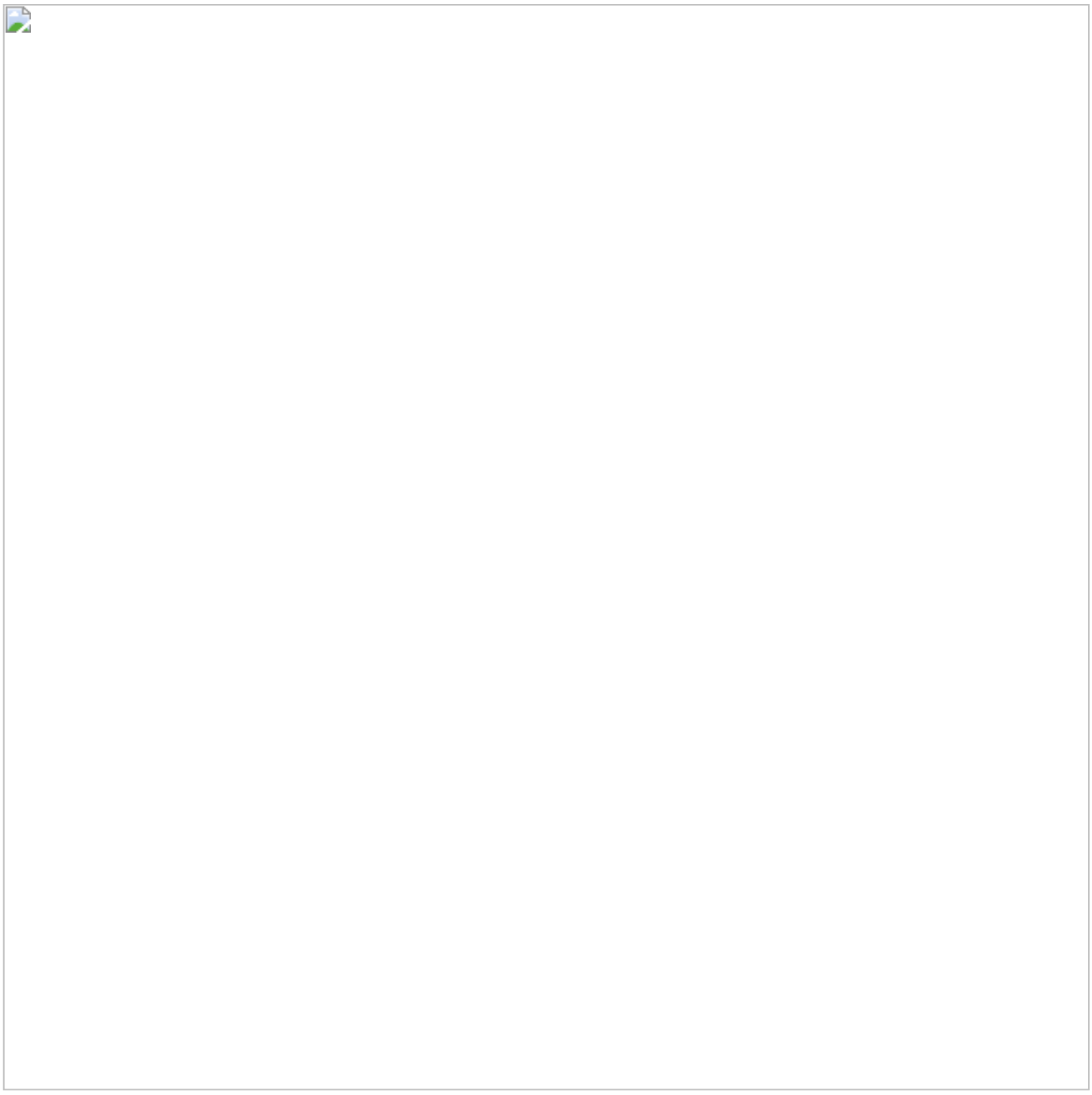
Example of Code Point

There are other encodings, but this will be left to the reader for personal research.

Punycode

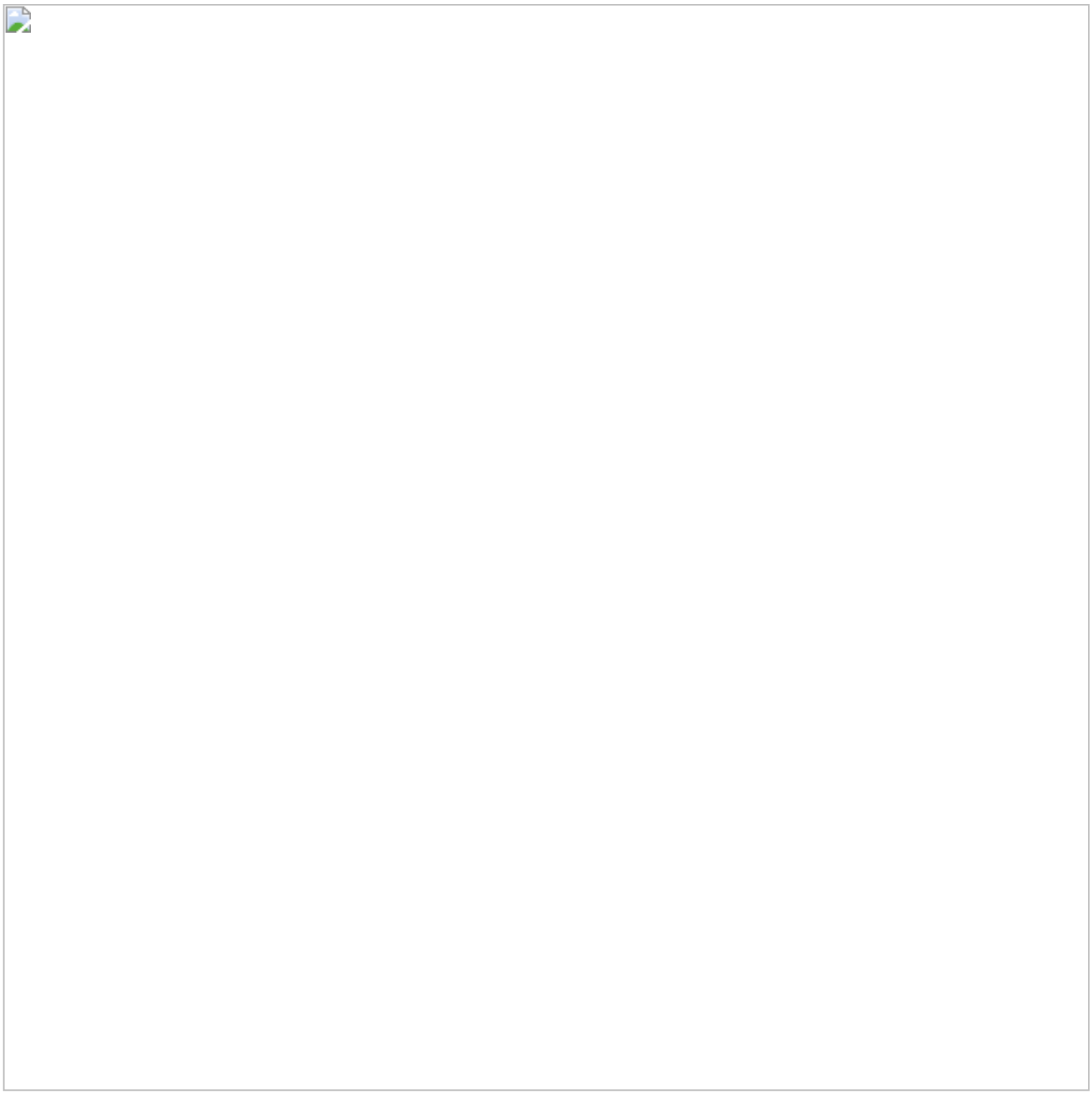
Interestingly, in 2020 we don't just communicate with letters and words, we also use emoji. For example, we regularly use this one when we are happy: 🙂 or this one when we are sad: 😭 and even this one: 👍 to show that we agree… In fact, there is a whole list of emojis that can be used to extend and express our feelings with computers.

The below diagram shows the most used emoji.



Even domain names can use emoji today using punycode. Punycode is a way to convert encoded strings containing Unicode characters for Internet hostnames.

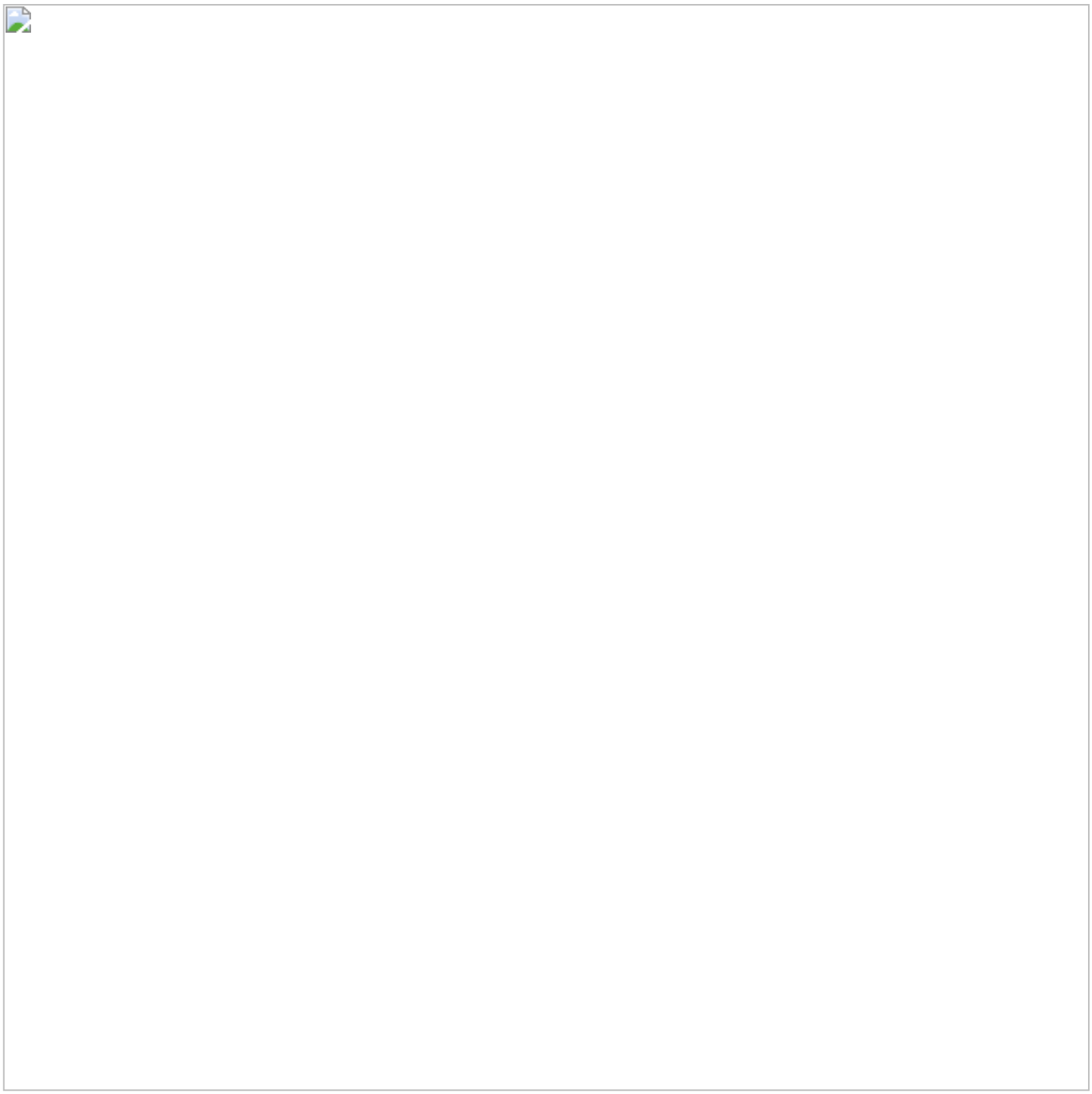
The example below is an example of a punycode using emoji for a hostname.



Punycode Example

Punycode can also be used for phishing, because some characters may have a similarity to the usual English alphabet, attackers can take advantage of using Unicode characters to appear as a legitimate website. This attack is known as IDN homograph attack.

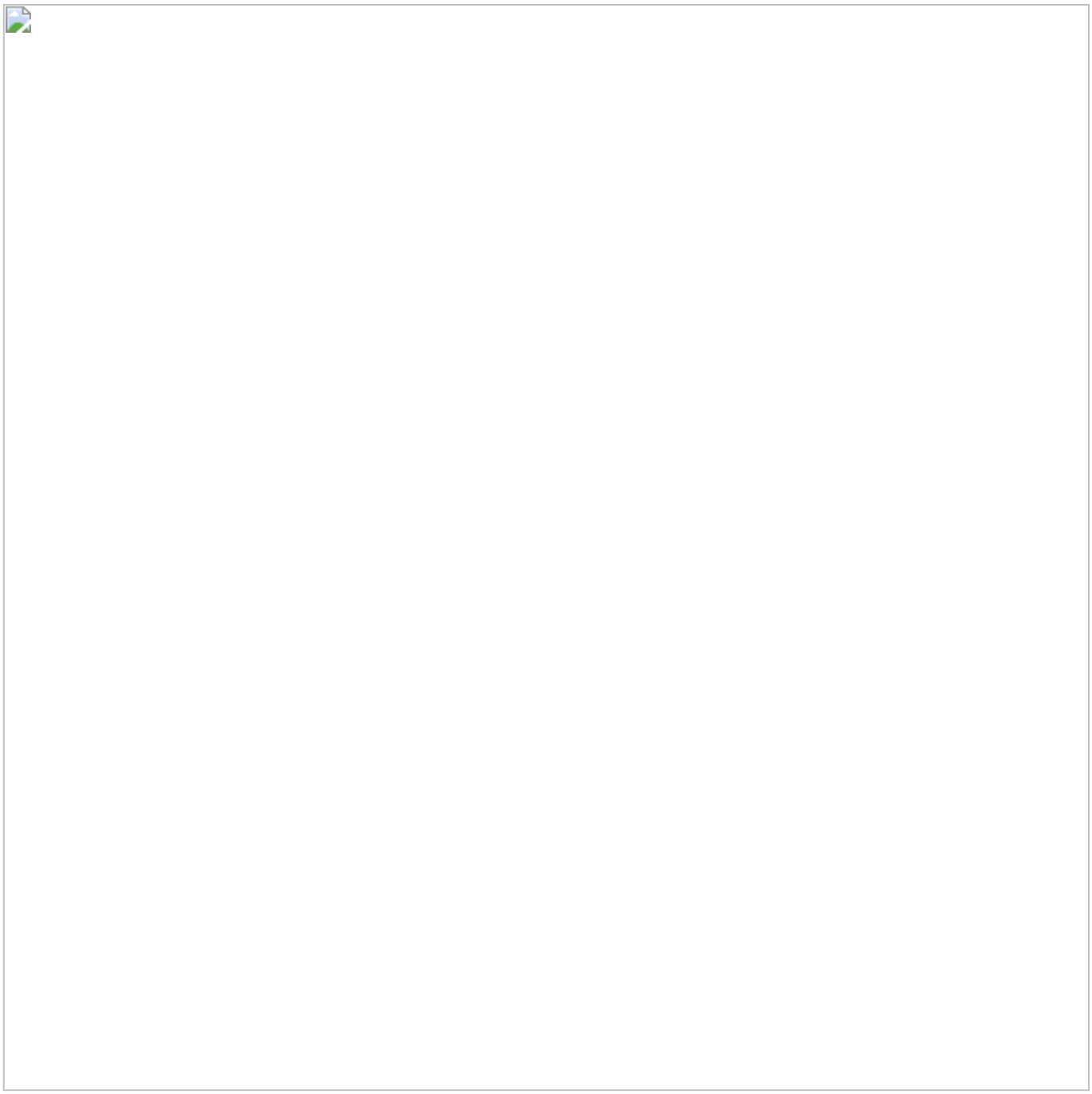
The below example shows an example with the “ạ” instead of the “a”.



IDN Homograph Attack

This type of attack can be easily set up and users who are not careful can fall into the trap.

To detect IDN homograph attack, the python script MyLittlePuny can check domain names that can be used. In the screenshot below we can see the use of the script for facebook.com with the letter “A”.



MyLitlePuny on Facebook Domain

Strings for Threat Intelligence

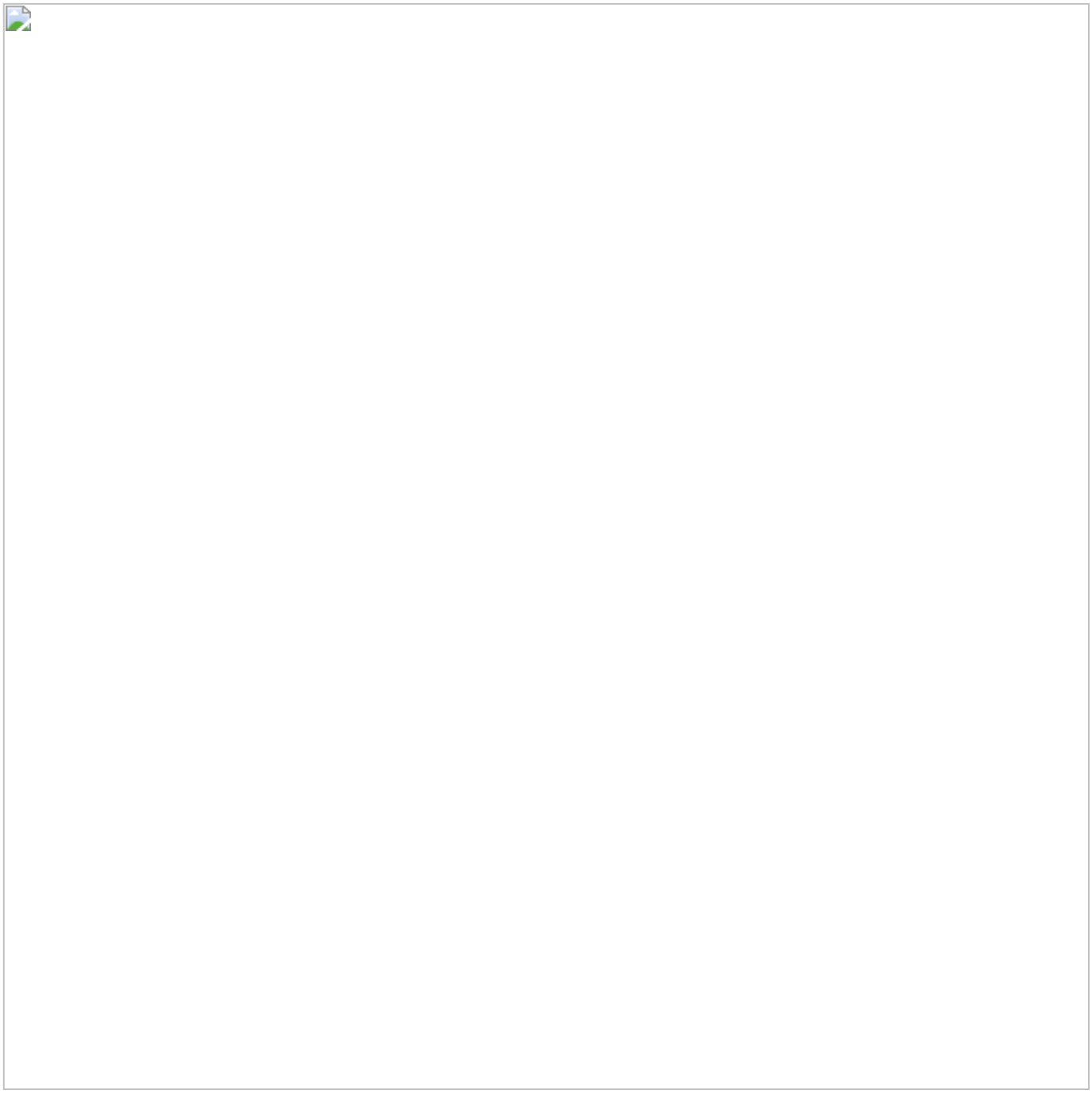
Now that we know a little more about how character encoding works in modern computers, we will discuss in this section how we can take advantage of it for threat research.

Strings

The tool Strings is one of the most used tool when analysing malware. It allows the analyst to quickly identify the sequence of characters that can be useful in identifying features, or any other variable used by the malware.

Strings is a native tool built into any Linux system. A Windows version is available and on Mac OSX, strings is available in the Xcode Command Line Tools package.

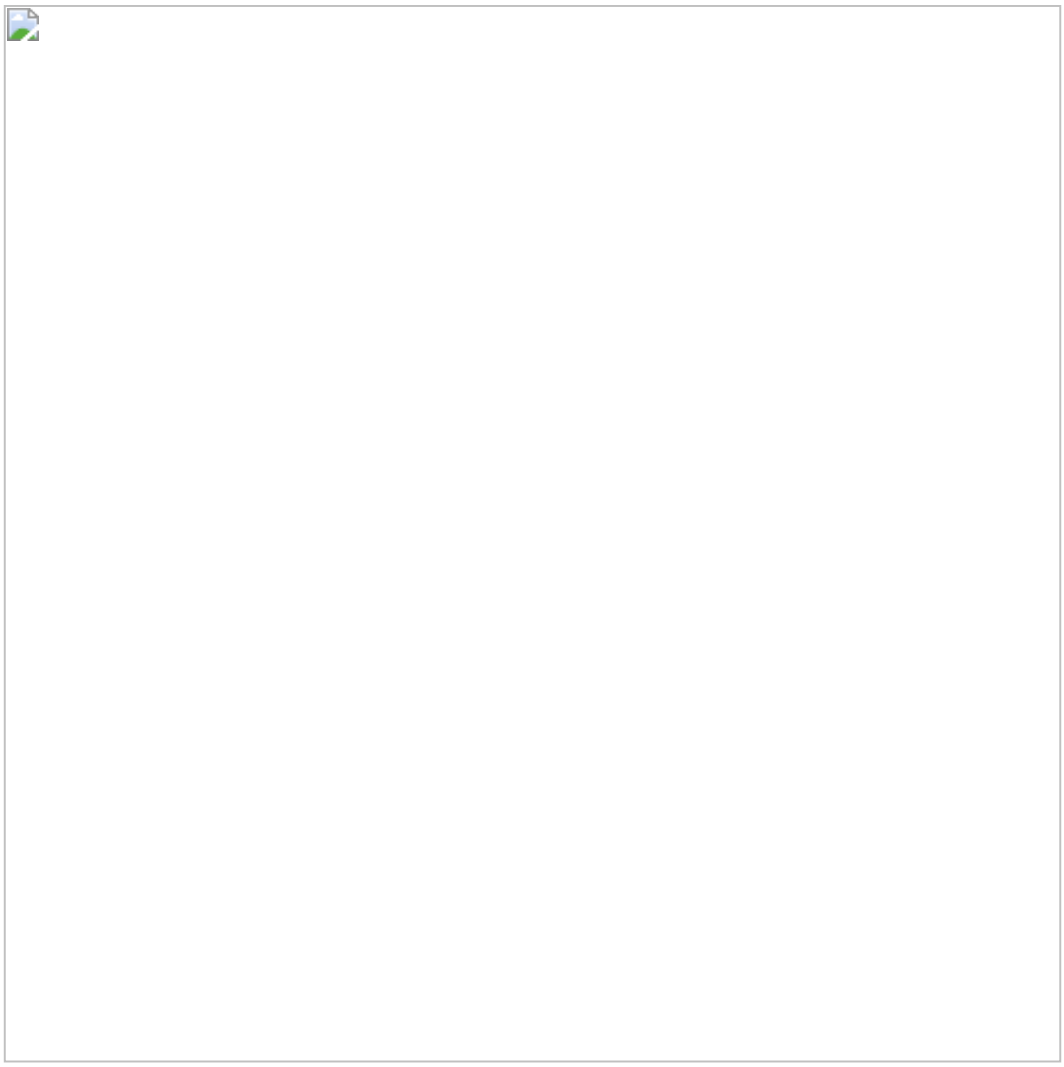
Below is the default output of strings.



Strings Help

To extract printable characters from a given file, you can use strings without any options.

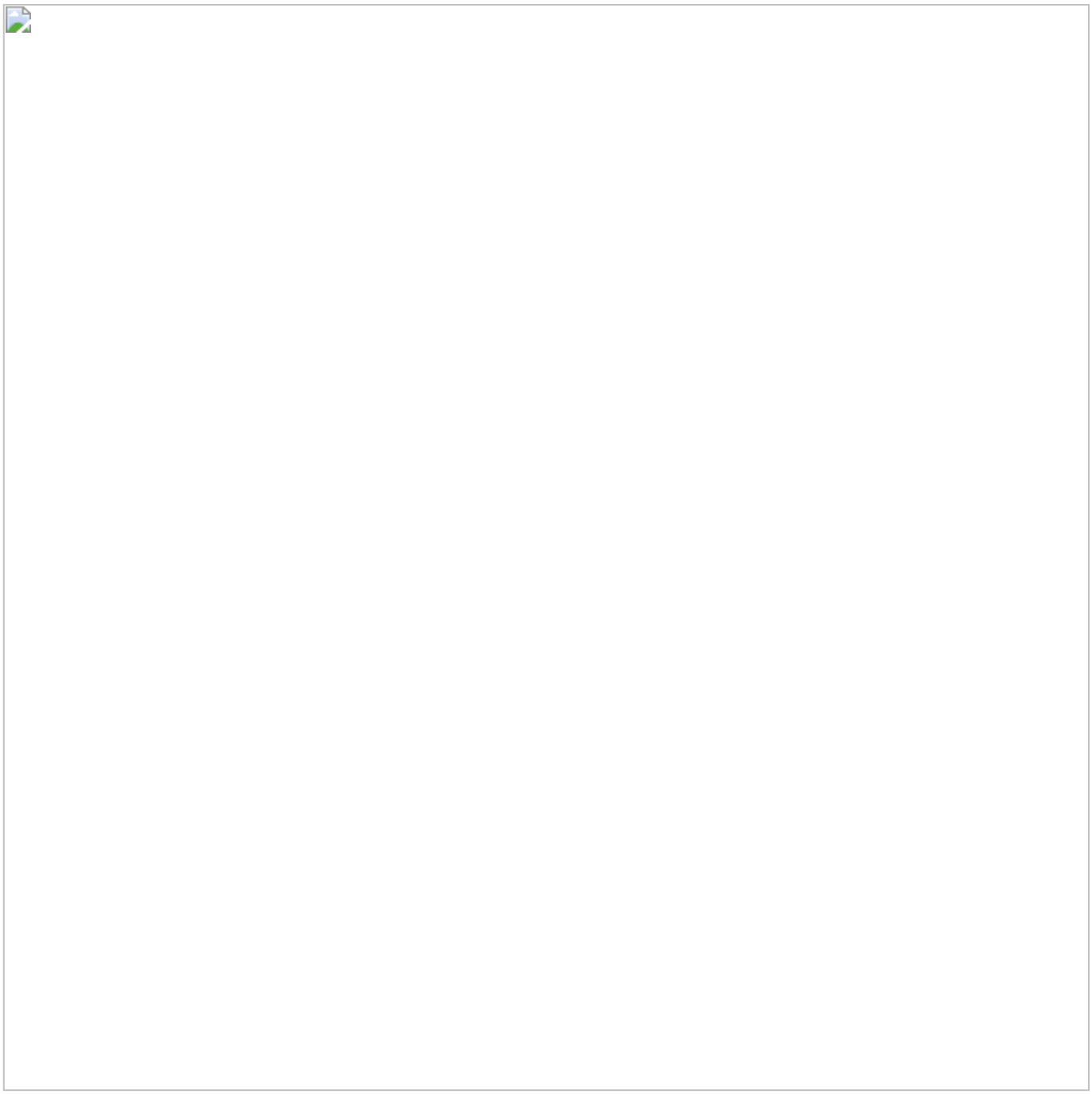
This will provide the following output:



Strings Extraction

You will notice a lot of junk bytes among the output with interesting results. Not all strings found by the tool are valid strings, they can be a memory address, CPU instructions, or data used by the program. In this case, we can spot the information that refers to a keylogger.

It is also possible to extract strings with a specific encoding with the “-e” option.



Strings -e b

In this case, we extract the information in UTF-16 which prints additional information not present in the default execution of the strings.

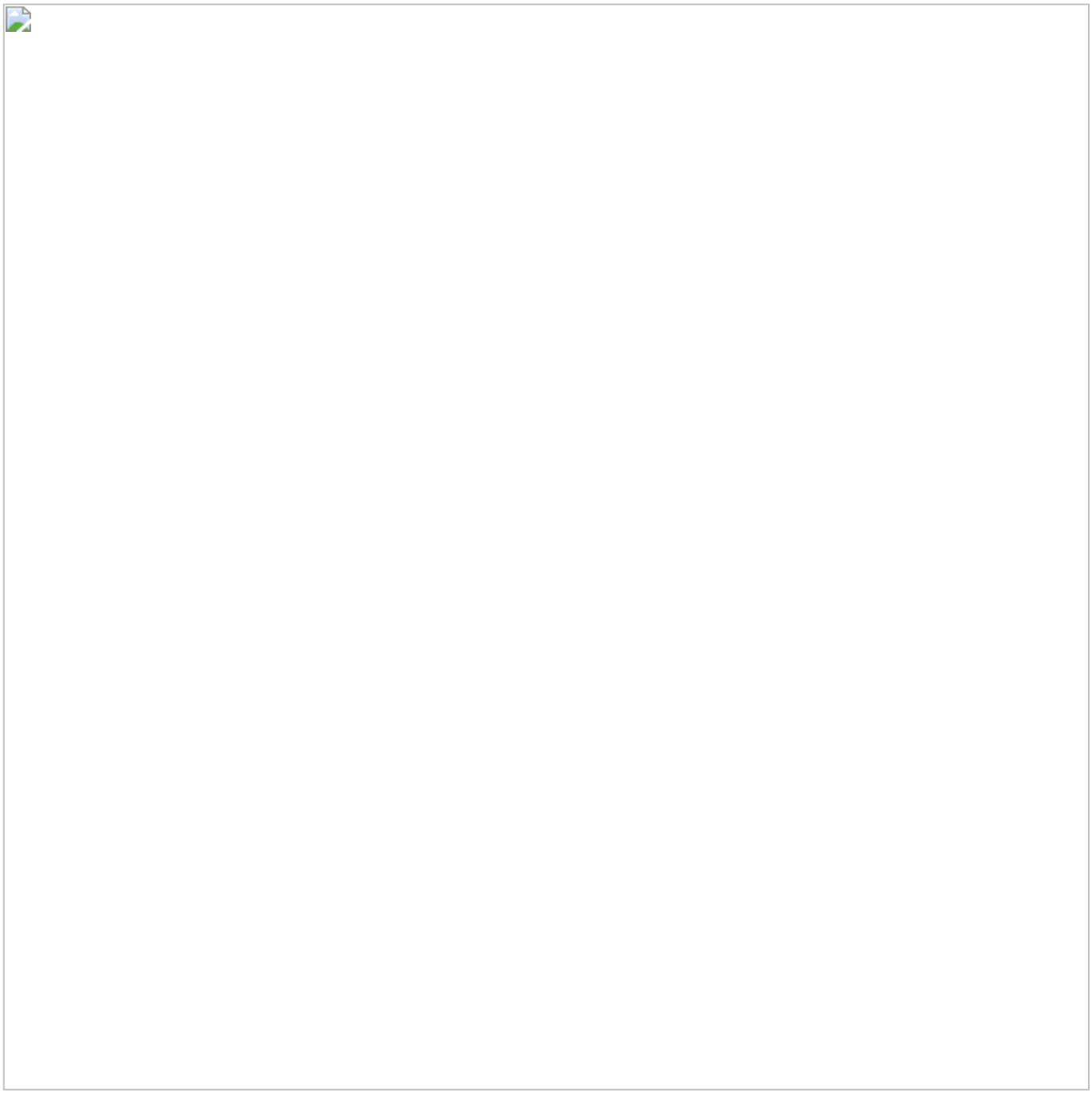
Strings Obfuscation

Malware writers know that analysts look at strings. To avoid rapid identification, they sometimes use obfuscation to hide data in a binary. XOR encryption, base64, or some other mechanisms are used in most samples to hide content, making the output of strings useless. Analysing obfuscated strings can take a long time.

The tool FLOSS, provides a handy tool to quickly identify obfuscated strings in a binary.

Basically the tool will try to detect the decoding routines and emulate the code to extract the decoded strings. Although it does not always detect interesting strings, in some cases the tool is able to extract very relevant information.

In the screenshot below we can see the strings decoded using FLOSS in the sample 304cceff9d29e8f879124f183337b28ffd7c28e2.



FLOSS Decoded Function

Here you may notice some useful information like cmd.exe commands to run a dll, registry key change and also HTTP request which may indicate behaviour of the sample.

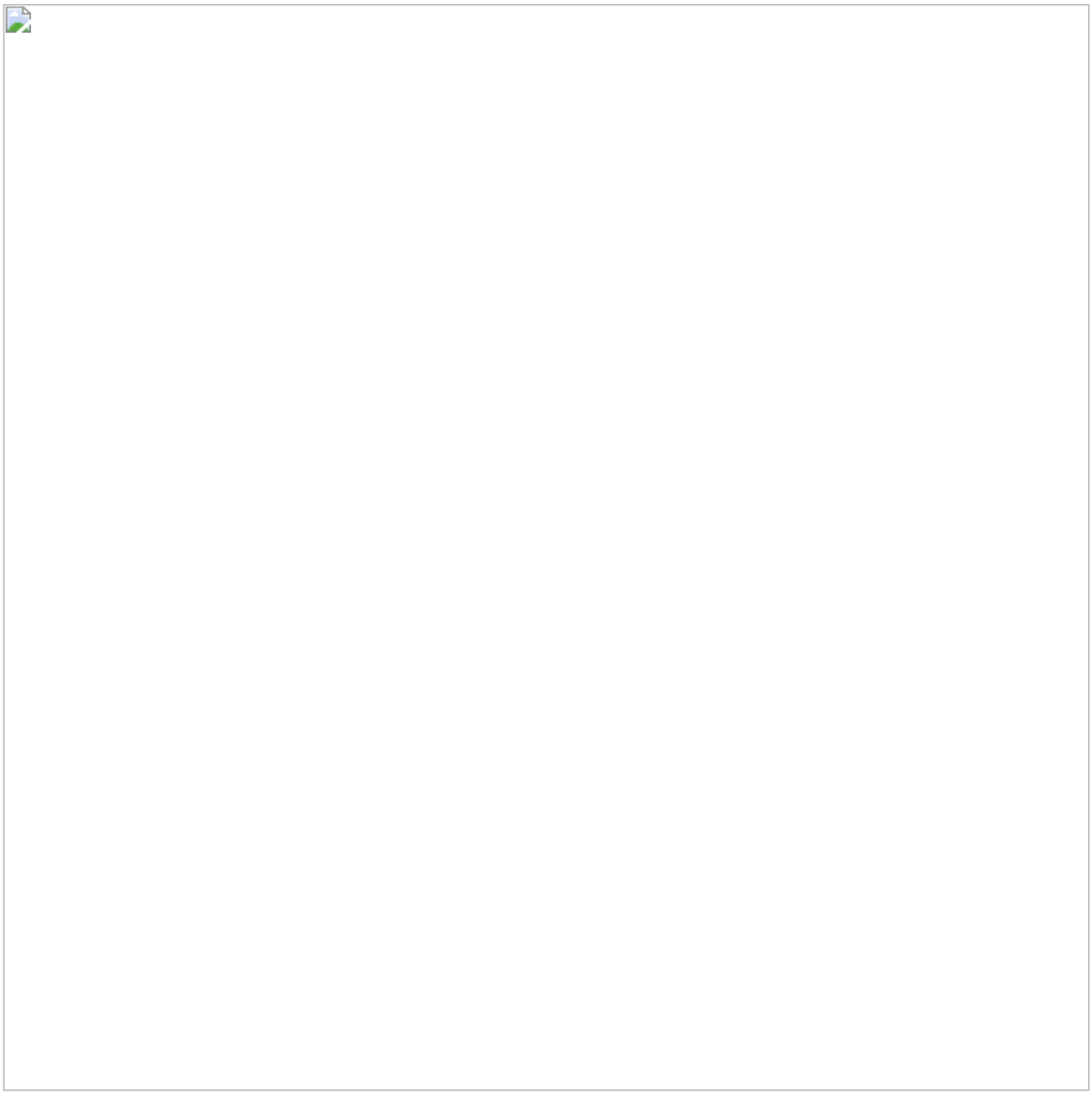
Using Strings with Graph Theory

In malware research, it can be useful to extract strings and compare them in a large dataset to identify similar samples or similar variables used. Graph theory will make it possible to use this data and put it in a graph to show the connection between a list of samples.

To compare the similarity within the samples, it is possible to use a similarity algorithm such as the Jaccard index. The Jaccard index is a statistic used to understand the similarities between sets of samples.

Using string extraction with multiple malware and creating a set that will be compared with the Jaccard distance and written in a graph can quickly help research to understand the connection between multiple samples.

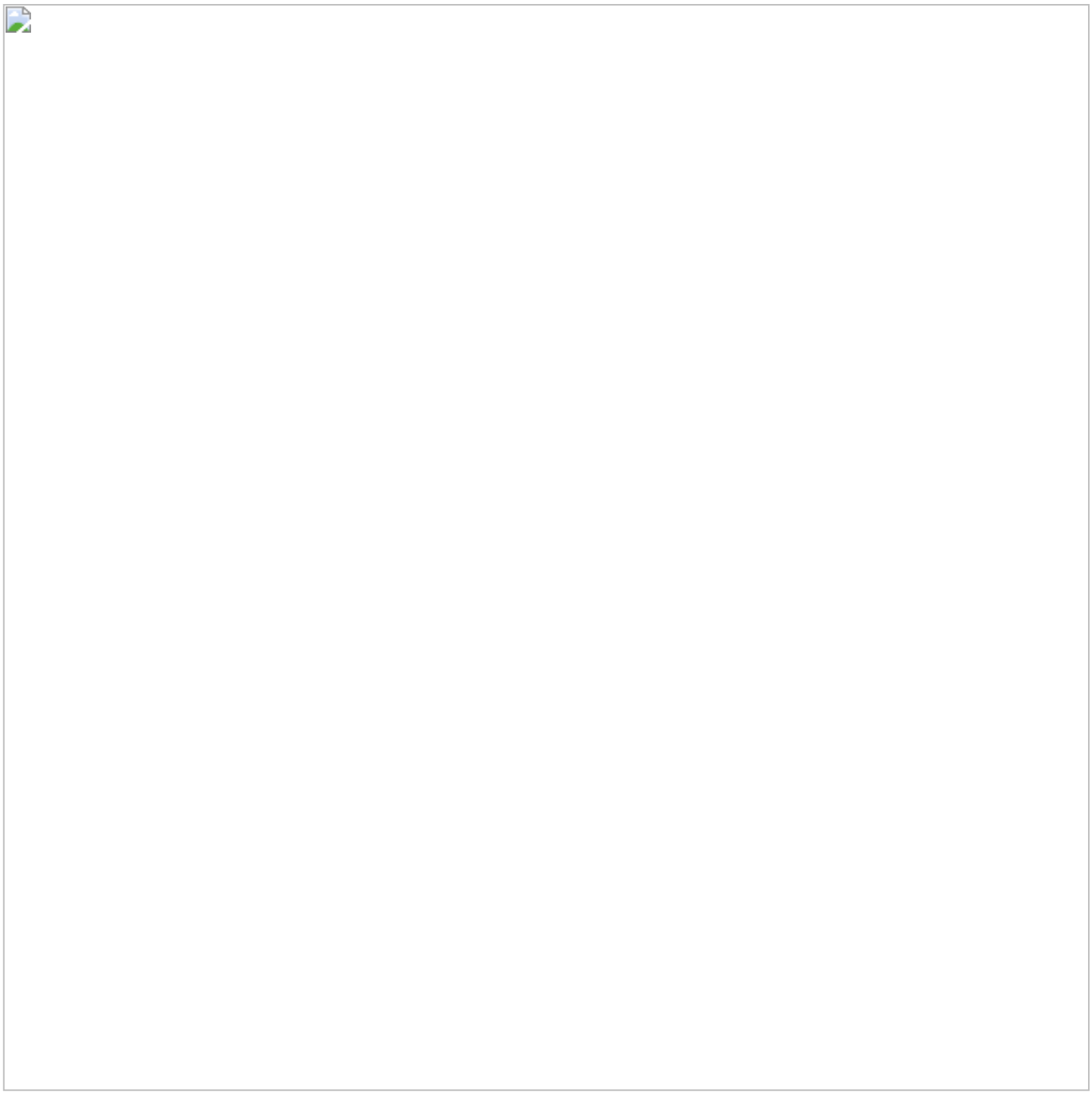
The example below demonstrates this concept for the Dtrack malware family.



Strings Correlation

In this screenshot, we can directly identify malware using the same strings and thus spot the similarities between the different variants.

It is also possible to apply this same concept to extract a single strings and identify the samples that use it. In the example below, we apply this concept by extracting a well-known string used in several samples as a password: “dkwero38oerA^t@#”.



Single String Extraction

In no time we were able to expose samples using the same strings. To generate these graph I created a Jupyter Notebook that explains and demonstrates the process https://github.com/fr0gger/strings_similarity.

Extracting strings and comparing them to plot them in a graph is very powerful for sorting malware and identifying strings sharing. Of course, this method has limitations and packed samples will not provide any useful information. This is why it is very important to clean your dataset first.

Leveraging Strings for Powerful Yara Rules

Yara is one of the most powerful tools to hunt for malware using strings. The goal of yara is basically to create a rule that will detect the strings previously identified when scanning for malware.

To write a good yara rule, the first step is to identify unique strings in the binary. Meaning strings that are not common or that are specific to the binary you are currently analysing.

To create your own Yara rule, you can for example use a registry key with specific values related to a sample, PDB path, certificates, mutex, filename, specific encoder habits or even encrypted strings. It is also worth adding condition corresponding to the file format such as size, entropy or even sections.

The example below demonstrates some use.

```
rule Rule_Example {    meta:        description = "Rule example"                author =
"@fr0gger_"                date = "2020"                rule_version = "v_"
actor_type = ""                hash1 = ""    strings:        $s1 = “dkwero38oerA” wide
ascii        $s2 = “CCS_” nocase        $s3 = “%02X:%02X:%02X:%02X:%02X:%02X” ascii
fullword        $hex1 = { E2 34 A1 C8 23 FB }        $hex2 = { F4 23 [4-6] 62 B4 }
$xor1 = "This program cannot" xor        $xor2 = "This program cannot" xor(0x01-0xff)
$b64 = "This program cannot" base64        $re = /md5: [0-9a-fA-F]{32}/    condition:
uint16(0) == 0x5a4d and filesize < 2000KB and (3 of $s* or          any of $hex* or any
of $xor* and $b64 and $re)}
```

Limitations

The use of string extraction is very powerful, but there are some limitations. For example, if your binary is packed or obfuscated, the string extraction won’t really reflect the binary you are analysing but rather the packing or obfuscation routines used to hide sensitive data.

Even if samples are packed, it is very useful to check the contents for a first look. Analysing packed samples doesn’t mean you can’t leverage strings for malware hunting, but you should consider this before you continue your research. It will of course be possible to search for these specific packing routines, and this can be powerful if several samples use the same algorithm.

Additionally, some malware authors will bind the binary to legitimate software, resulting in more strings that will generate false positives when using it with Yara rules.

Conclusion

In this blog post, we explored the string extraction process used when analysing malware. We have seen that multiple encoding can be used in a modern computer for different purposes. We have also seen that extracting strings from a malware can be very powerful for quickly identifying its capabilities as well as for further research. It is also possible to combine string extraction with graph theory to generate diagrams and expose proofs of similarities between multiple samples.

Even though extracting strings is one of the first steps in malware analysis, it is crucial to know the limits and understand that malware authors will obfuscate, hide, or trick the strings inside the binary to counter the analysis. As a malware researcher, you should keep this in mind during your analysing and research process.

I hope you enjoyed this blog, if you like this content you can follow me on Twitter @fr0gger_ for more stuffs such as this one. ❤