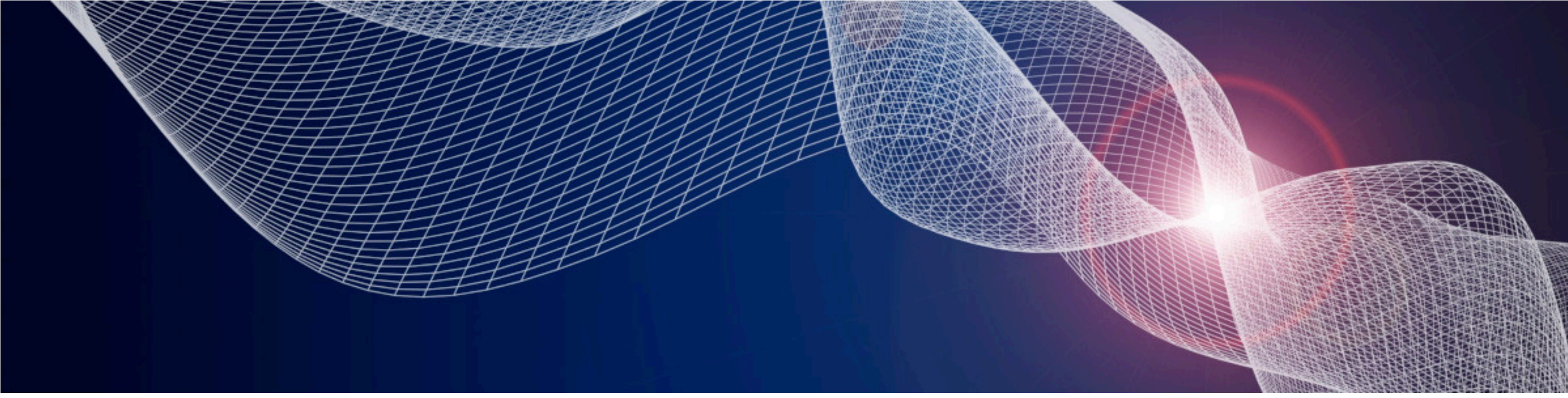

3. Static Analysis

3.1 Basics

3.1.1 File Hash (30’)

Check out the following SHA265 hash on the two sites mentioned and browse the information you can find there!

Sample Hash

```
98ca95d5888ed57532434861e0db91d5bf9b4b10b56bf5a1fb8b27fd7659fcf8
```

You don’t need to submit anything, just look around and get familiar with the sites. Also feel free to browse other samples for comparison.

There will be a lot of malware specific information - that’s what these sites focus on. We’re only really interested in the generic information for now, we will cover malware specifics in the lecture.

- We will particularly look at the *YARA Rules* you will find listed on *Malware Bazaar*.

3.2 Ghidra

3.2.1 Installation (15')

1. Make sure you have a Windows VM[1] set up.
   - We might be dealing with malicious code, so we should not work directly from our host system even if (or especially if) we do our normal work from a Windows machine.
   - Assure that your VM is allocated enough RAM (I would recommend at least 8GB) and CPU cores (make it at least 2).
   - Start up your Windows VM and do all the following work from within it.
2. Install Java 17[2]
   - download the *msi* package of the *JDK* and run it.
   - enable the "Set JAVA_HOME variable" option during installation.
3. Download the latest release[3] from GitHub.
   - We're going to assume you downloaded and unpacked to a folder `ghidra` in `%USERPROFILE%` (`$HOME` in *PowerShell*).
4. Open a *PowerShell* window and navigate to `$HOME\ghidra`.

That's it, you're ready to run your shiny new *Ghidra* by executing `ghidraRun.bat`!

[1] `https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/`
[2] `https://adoptium.net/temurin/releases/`
[3] `https://github.com/NationalSecurityAgency/ghidra/releases`

3.2.2 Introduction

3.2.2.1 Importing Programs (15')

Open up ghidra and import `ex1-win.exe` (you can download it from the book repository). Open it and run an analysis. Have a look around!

3.2.3 Component Overview

3.2.3.1 Strings (15')

Have a look at the defined strings in `ex1-win.exe`. Which ones seem relevant?

3.2.3.2 Symbol References (15')

Find the strings you decided were relevant in the previous exercise and check where they're referenced from. What do you notice?

3.2.3.3 Decompile & AST Control Flow (75')

Have a look at `ex1-win.exe`'s entry point's code in the "Decompile" component.

1. Select some code in the "Decompile" component
   - notice how the matching parts get selected in the "Listing" view at the same time. Try it the other way around as well. Can you match the two representations?
   - Open the "AST Control Flow" component for the entry function and see how it interacts with both the "Listing" and the "Decompile" view.
2. This code looks pretty complicated for just an entry point. The original program has much less code than this altogether - and this is just the entry point!
   - Any idea where this code came from?
   - How did it get there?
3. Try to identify the `main` function.
   - What's the function name / label?
   - How did you find it?

3.2.4 Program Analysis

3.2.4.1 Editing Function Signatures (30')

If you managed to identify the `main` function above, work with this one. If you didn't manage, import the MacOS version of the same program (`ex1-macOS`) and find the `entry` function. That's directly `main` (so much simpler, isn't it?)!

1. The type of the second parameter seems strange, doesn't it?
   - What should it actually be?
   - Why doesn't it make a difference on a lower level?
2. Edit the function signature to correctly define the function.
3. What changes happen in the function code due to the adjustment of the signature?
4. If you have time left (or you find this interesting ;-)), compare the two versions (*Windows* and *MacOS*).
   - What differences do you notice?
   - By comparing the two, you'll easily see what the currently unknown function is.
     – Change its signature as well to clean up a bit more.
     – Note that it's a function which accepts a variable number of parameters. Check the "varargs" function attribute.
   - After editing, what differences are still left?

3.2.4.2 Overriding Call Signatures (15')

Make sure you go back to the windows version of the program (`ex1-win`). If you didn't manage to find the `main` function, navigate there following the references to the strings you found in `main` of `ex1-macOS`.

You may have guessed that some of the methods passing these strings as arguments are calls to `printf` (well, a wrapper of the function, but that doesn't matter for our purposes). But wait, what are all those extra parameters? Override the function calls to only pass what's required.

> Ghidra does this for you automatically if you edit the signature of the function called to accept a `Varargs` parameter in the second position. That's not what this exercise is about, though.

3.2.4.3 Renaming & Retyping Variables (15')

Go back to your preferred version of the program and navigate to the `main` function.

1. Go ahead and assign better names (and, if necessary, types) to the variables.
2. Observe what happens in the "Listing" component.

3.2.4.4 Retyping Data & Renaming Labels (15')

- If you've been working with the *Windows* version, you'll notice that the first parameter for the strangely named function within `main` is a pointer to data. Navigate to that location.
- If you're working in the *MacOS* version, double click on the string "%s" in the "Decompile" component. When you get to the target in the "Listing" component, click on "%s" there and press lower case "C".

Now everyone should have roughly the same setting:

- A strange label starting with "DAT_"
- (At least) three lines with one byte of data on each.

Your task: turn this into a string.

- How many bytes do you have to select? Why?
- Change the label to something more recognizable.

3.2.4.5 Putting it all together (75')

1. Now that you've dealt with `ex1-win.exe` or `ex1-macOS` for a while, can you run it from a console and get it to be happy with your input?
   - Remember, they're the same program, so no matter which one you reverse engineer, you can apply your findings to any of them (also to `ex1-linux`).
   - Regardless of how much you trust me, you should only run a program on your host if you're absolutely certain you've understood what it does. If you're not, run it in a disconnected VM!
2. To end with, please open and analyze `ex2-*` (of whatever flavor you prefer).
   - Which version did you choose? Why?
   - What are the steps the program takes?
   - What functions can you discover? What would you name them?
   - What would be good variable names?
   - What does the program do?
   - Can you make the program happy?