Final Assignment

Micah Flack

CSC-804-DT1 – Cybersecurity Research Methodologies

Dakota State University

Dr. Bhaskar Rimal

05/05/2023

Chapter III: METHODOLOGY

The following chapter presents the research methodology and system design used for the analysis of cross-compiled fat binaries against single-platform binaries. Fat binaries, which contain code for multiple architectures, are increasingly used to bridge generational gaps in device architecture and operating system versions. However, comparing fat binaries poses several challenges, such as identifying the architecture-specific code and handling differences in the alignment of code and data. This chapter presents the methodology for comparing fat binaries in the context of malware attribution. The chapter then discusses the research questions, objectives, and methods used, including the selection and gathering of the samples, the measurement of the binaries' data, and the methods of analysis for the data. Any limitations of the study and validation of the methodology is also discussed.

Research Questions

This study seeks to identify shared patterns and/or commonalities to the following research questions:

RQ1: Structurally, how do the binaries differ?

RQ2: Do these differences impede attribution? E.g., packed executables.

RQ3: Can fat binaries still be identified using the attribution methods of slim binaries?

RQ4: Does the data gathered present any trends or patterns that could be used to modify the previous attribution methods?

Methodology Selected

This research follows the quantitative methodology and focuses on the use of comparative case studies. Robert Yin [1], in his book " Case Study Research and Applications: Design and Methods," describes a case study as an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident. Case studies are often used to explore complex phenomena in-depth and from multiple perspectives, and they typically involve the collection of qualitative data through a range of methods, such as interviews, observations, and document analysis.

However, for this study quantitative data will be collected instead from multiple fat binaries generated by cross-compilers. Although case studies are typically performed using qualitative data, because they can supposedly better capture the richness and complexity of a phenomenon, the use of quantitative data is understood and well-studied [2]. Case studies are also a fairly common method used for establishing or introducing new data on malware to aid future development [3]–[5].

Comparative Case Studies

Comparative case studies [1] are a research method where a researcher examines multiple cases to find similarities and differences between them. The goal is to understand how different factors may have influenced the outcomes. The cases being studied can be similar or different in terms of their characteristics, settings, or outcomes. By comparing different cases, the researcher hopes to gain insights that can be used to inform theory development, policy decisions, or practice recommendations.

The use of comparative case studies in this research study intends to explore the efficacy of current malware attribution methods across several types of fat binaries, as well as identify their limitations through an analysis of the data collected. Using a combination of the python libraries angr [6], pandas [7], and LIEF (Library to Instrument Executable Formats) [8] to extract the structured data about the executables. We can then draw comparisons in an attempt to identify the similar databufs, structures, control flows, and functions shared between the cases. Because of the varied nature of fat binaries, be they multi-platform or multi-architecture, there are certain limitations expected.

Assumptions, Scope, and Limitations

The ability to ingest binaries and perform an analysis is dependent upon their targeted platform and architecture. Since this study analyzes a few samples at a time, the hardware specifications described in the Procedures Followed section are more than sufficient. There are no inherent limitations with using the Pandas library for representation of executable data and statistics, however, whether the data can even be produced is dependent upon LIEF and angr support. For instance, the following libraries are limited by the following platforms and executable formats:

1. LIEF [8]
    a. ELF (Linux), PE (Windows), Mach-O (MacOS)
2. Angr [6]
    a. ELF (Linux), PE (Windows), Mach-O (MacOS), memory blob
3. Cosmopolitan [9]

a. Linux, Mac, Windows, FreeBSD, OpenBSD, NetBSD, and BIOS

For the purposes of keeping the objectives clearly defined and limiting scope creep, it is presumed that all binaries analyzed are x86 (any platform) or ARM (only MacOS) architecture. Unless indicated, it is also assumed that any binary used for analysis was compiled using the Cosmopolitan compiler [9] through leaked source code of the targeted malware. Any binaries submitted to the executable analysis framework are also submitted as their executable format and not as binary, hex, or human readable code.

Data Collection

There are two phases to the data collection of this research study. First, the researcher gathered samples from various malware repositories located at the following:

1. VirusTotal (VT) [10]
2. HybridAnalysis (HA) [11]
3. Any.run (AR) [12]
4. Misc. GitHub Repositories (GR) [13]

The Malpedia Library by Fraunhofer (FIKE) [14] is a great source for the historical data of malware families or advanced persistent threats (APTs). This is where the researchers will identify the malware families using fat compilers (multi-platform/architectural cross-compilers). The sources held by this library contain deep technical reports on the behavior of the malware, indicators of compromise (IOCs), and the static format, structures, resources, and functionality of the malware.

Using the information from the reports, the individual binaries from each of the malware families is then pulled from the specified repositories, and categorized by their malware family, the compiler used, and their md5 hash. For example, an executable from the SilverSparrow [15] chain of MacOS malware would appear as the following:

silversparrow_lipo_30c9bc7d40454e501c358f77449071aa.bin

The second part of the data collection phase begins by using a combination of the previously mentioned python libraries: LIEF, pandas, and angr. The libraries will be used to extract the following types of data:

1. LIEF
    a. Imports/Exports
    b. Embeded Resources
    c. Manifest
    d. Symbol/Abstract layer
    e. Header
2. Angr
    a. Function endpoints
    b. Function entrypoints
    c. Control flow graphs (CFGs)
    d. Intermediary block data
3. Pandas
    a. LIEF dataframe
        i. Statistical data
    b. Angr dataframe

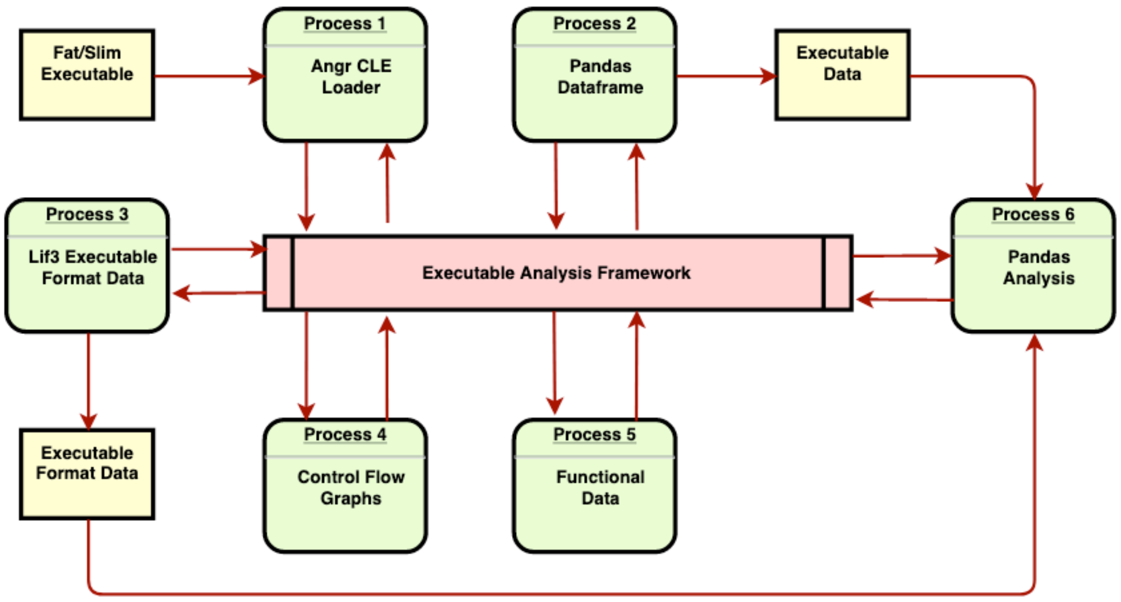
   i. Statistical data

  c. Pattern mappings between dataframes

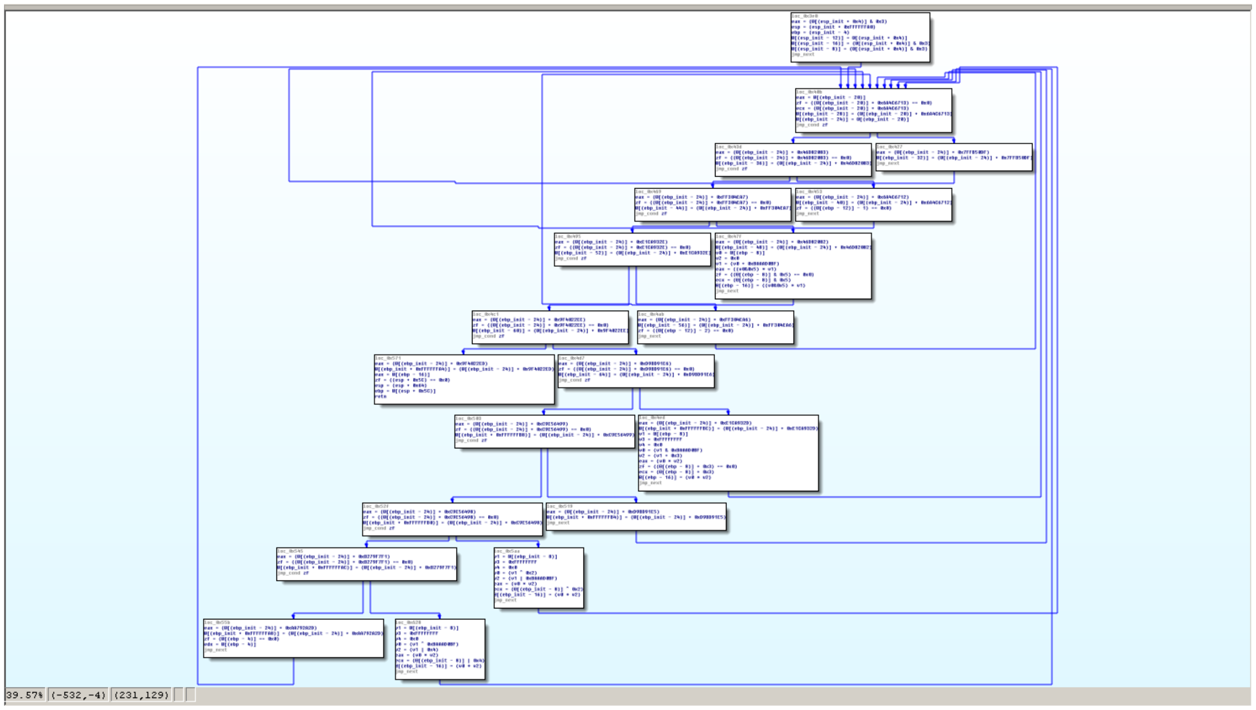
  d. Table visualizations

Once all the data has been collected and processed through the executable analysis framework, the data analysis phase begins, and any patterns or trends held by the dataframes will be identified and then visualized.

Data Analysis



Analysis of the retrieved fat and slim binaries will be performed using artifacts generated by Angr and LIEF, as well as statistical data provided by Pandas using the individual dataframes for each binary; one for LIEF and another for Angr data. Using LIEF we can compare executable format information, such as the header, resources, imports, or exports. Angr will be used for the analysis and comparison of rebased function entrypoints and endpoints – it can also generate control flow graphs (CFGs) usable during the analysis and comparison phase.



First, the LIEF dataframe will provide access to various information about the headers, sections, segments, and symbols. Second, using Angr we will populate our dataframe using the vex intermediary representation (IR) for functional block data [16]. Vex IR is an in-between for representing different abstract mechanisms or instructions within code, and it works across widely diverse architectures. Meaning, the IR data is resistant to changes introduced by cross-compilation.

For instance, the IR abstracts can be used to represent register names as symbols for symbolic execution, to represent memory access, memory segmentation, or instruction side-effects. The representation has five main classes of objects:

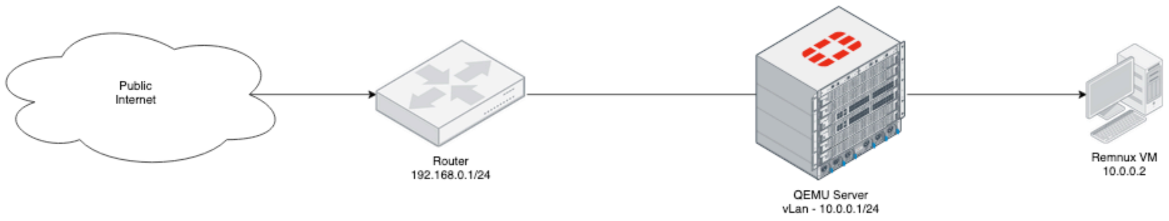
1. Expressions: a calculated or constant value
2. Operations: a modification of IR expressions, e.g., arithmetic

3. Temporary variables: temporary variables as internal registers
4. Statements: changes in state of target machine, e.g., memory read/writes
5. Blocks: a collection of IR statements

With this data from LIEF and Angr, we can create categorical groupings using the two dataframes and Pandas built-in methods. By comparing the IR values for each group of IR blocks or a function, we should be able to infer the similarity of two functions from separate binaries. Furthermore, this method of analysis should allow us to infer which types of functions or regions within a binary are most useful for attribution methods.

The objective of this analysis then, is to identify trends or patterns beyond simple signatures targeting shared databufs or hashes like an analyst might use with Yara [17]. But to instead provide functional data identifying other areas useable by analysts for robust malware attribution resistant to fat binaries. The next section describes the process for replicating this analysis.

Procedures Followed



Before the research could even be conducted, an environment to practice the data collection and analysis was needed. The environment used in this study is not a concrete requirement, however, it does ease the setup process and save time. By using the QEMU [18] virtualization environment, any analysis of the malware is contained and damage is localized to the virtual machine (VM) and virtual network.

QEMU is an awesome virtualization platform that allows you to run multiple operating systems on one computer. It's open-source, which means it's free to use and anyone can contribute to its development. You can emulate different types of computer hardware, like x86 or ARM, and run different operating systems, such as Windows or Linux. It's widely used for software development and testing, as well as for virtualization purposes. However, the majority of the malware used in this study is x86 and the analysis does not require emulation capabilities. QEMU is used here solely for the containerization of the analysis environment.

Using QEMU, a virtual machine is then created using the Remnux [19] distro based on Ubuntu 20.04 (Focal). Remnux is a Linux toolkit built for reverse-engineering and analyzing malicious software. It provides a suite of free tools created by the community. This enables analysts to investigate malware without needing to spend time finding, installing, and configuring the tools. As for the virtual hardware provisioned for the VM, it was given four cores, eight gigabytes (GB) of memory, and 50GB of storage space. A clone of the VM disk will be provided for validity and repeatability reasons.

With the Remnux environment configured and running, the researcher then needed to install the three python 3.9 libraries using pip: LIEF, angr, and pandas. In some cases, it is easier to create a virtual env for the project which prevents python import contamination. Typically, the process would appear as:

```
$> cd ~/ && mkdir venvs
$> python3 -m venv ~/venvs/binary_analysis
$> source ~/venvs/binary_analysis/bin/activate

$ binary_analysis > python -m pip install ipython pandas lief angr

…

$ binary_analysis>
```

The researcher can now use the environment to create scripts using the imported libraries. The additional ipython package is used for interactive importing and executing of scripts created. Before the researcher can use any of the scripts, they need to collect the executable samples from their respective families.

For purposes of repeatability, the chosen malware families are:

1. Pre-compiled.
   a. SilverSparrow
2. Compiled from source using Cosmopolitan libc compiler (version 2.2) [9].
   a. Athena [20]
   b. APT34 [21]

Using the malware samples or source as identified by Malpedia, we download them from their repository and prepare them to be loaded through the executable analysis framework. If the sources chosen are not compiled, then that is completed using the Cosmopolitan libc compiler. Data is then further gathered from the samples using the angr and LIEF scripts.

With the samples ingested into the analysis framework and data gathered, the researcher should be ready to begin the analysis phase and prepare visualizations from the executable features encapsulated within the pandas dataframes. All data from this point should be the same, unless the researcher performs independent changes or additions to the retrieved feature set; e.g., rebasing offsets of function entrypoints for comparisons of multiple binaries. For repeatability of the described processes, any researcher should refer to the public repo for access to binaries, scripts, and an environment image usable with QEMU.

Validity and Reliability

For purposes of repeatability and validity of the test data and results, all executables are labeled using the previously defined method. Each of the binaries hashes is also verified using md5sum (version 8.32) during the labeling process. The specific versions of tools, libraries, and databases are also documented. Any artifacts generated during the process are also archived as well. All data will be publicly accessible from the shared GitHub repository.

The Researcher

The researcher has worked in the cybersecurity field for the past six years and holds a Bachelor of Science in Cyber Operations and a Master of Science in Computer Science. The researcher currently works for Idaho National Laboratory as a cybersecurity researcher. In the past, they worked for Northrop Grumman, First Financial Bank USA, and as a student researcher for Dakota State University. The researcher does not have a direct relationship or familiarity with the owners of the malware repositories, nor with any of the tools used that could have an impact on the data represented or impart bias on the research study.

Summary

In conclusion, this chapter of the research study demonstrates that comparison case study methodology is an effective approach to analyzing and comparing fat binaries. By examining the case studies in depth, we identified key similarities and differences in binary structure, limitations of attribution, and possible static features and trends shared by fat binaries and their slim counterparts. The use of this methodology allows us to gain a comprehensive understanding of the complexities involved in fat binary analysis and provide valuable insights into its potential applications and limitations. Our findings hope to contribute to the field of fat binary analysis and provide a useful reference for future research in this area. Chapter four will discuss our results and analysis using the processes defined here.

REFERENCES

[1] R. K. Yin, *Case Study Research and Applications: Design and Methods*. SAGE Publications, 2017.

[2] V. R. Basili, “Quantitative evaluation of software methodology,” 1985.

[3] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. Manjunath, “MalGrid: Visualization Of Binary Features In Large Malware Corpora,” *ArXiv Prepr. ArXiv221102696*, 2022.

[4] J. Pattanaik, “Case study on IoT Malware”.

[5] R. Raphael, “Various Data Mining Techniques to Detect the Android Malware Applications: A Case Study,” *Int. J. New Technol. Res.*, vol. 5, Jul. 2019, doi: 10.31871/IJNTR.5.6.33.

[6] “angr homepage.” https://angr.io/ (accessed May 05, 2023).

[7] “pandas - Python Data Analysis Library.” https://pandas.pydata.org/ (accessed May 05, 2023).

[8] “LIEF Homepage,” *LIEF*, Jul. 18, 2021. https://lief-project.github.io/ (accessed May 05, 2023).

[9] “Cosmopolitan Libc: build-once run-anywhere C library.” https://justine.lol/cosmopolitan/ (accessed May 05, 2023).

[10] “VirusTotal - Home.” https://www.virustotal.com/gui/home/upload (accessed May 05, 2023).

[11]“HybridAnalysis.” https://hybrid-analysis.com/ (accessed May 05, 2023).

[12] “Interactive Online Malware Analysis Sandbox - ANY.RUN.” https://app.any.run/ (accessed May 05, 2023).

[13] “GitHub: Let’s build from here,” *GitHub*. https://github.com/ (accessed May 05, 2023).

[14] “Malpedia Library.” https://malpedia.caad.fkie.fraunhofer.de/ (accessed May 05, 2023).

[15] B. Donohue, “Silver Sparrow macOS malware with M1 compatibility,” *Red Canary*. https://redcanary.com/blog/clipping-silver-sparrows-wings/ (accessed May 05, 2023).

[16] “PyVEX - pyvex documentation.” https://api.angr.io/projects/pyvex/en/latest/quickstart.html (accessed May 05, 2023).

[17] M. Brengel and C. Rossow, “YARIX: Scalable YARA-based Malware Intelligence.,” presented at the USENIX Security Symposium, 2021, pp. 3541–3558.

[18] “QEMU.” https://wiki.qemu.org/Main_Page (accessed May 05, 2023).

[19] “REMnux: A Linux Toolkit for Malware Analysis.” https://docs.remnux.org/ (accessed May 05, 2023).

[20] “theZoo/malware/Source/Original/Athena at master · ytisf/theZoo,” *GitHub*. https://github.com/ytisf/theZoo (accessed May 05, 2023).

[21] “theZoo/malware/Source/Original/APT34 at master · ytisf/theZoo,” *GitHub*. https://github.com/ytisf/theZoo (accessed May 05, 2023).