A Distributed Platform For Binary Analysis

Jacob Mills Brandon Everhart Taylor Shields
Department of Computer Science, Florida State University
{jjm15e, bte14, ts14g}@my.fsu.edu
https://github.com/h0r53/openbintools

ABSTRACT

A survey of 300+ software development companies [5] suggests over 30% of companies do not use tools for software testing, over 52% of software development is outsourced, and when evaluating software developers, working software is considered more than twice as valuable than code without bugs. Since the majority of code is often developed externally, source code is not always available. This is also the case with proprietary systems and legacy systems. The corollary is, the majority of software systems in use are effectively black boxes, even to their developers. The same challenges also exist within government agencies. Over the past several decades, the United States Department of Defense has attempted to develop tools to reverse engineer systems comprised of billions of lines of code [6]. In this paper, we propose a distributed cross-platform solution for binary analysis by integrating various utilities into a portable software suite.

INTRODUCTION

Software reverse engineering is the practice of deconstructing binary files to reveal their architecture, control flow, and design, with the goal of extracting information that was not previously available. Significant research efforts have taken place to utilize reverse engineering to address the issues with third-party development, proprietary software, and legacy systems. Binary analysis is a useful practice for reverse engineers, software developers, and security professionals. Due to the variety of hardware, programming languages, commercial operating systems, and embedded systems, advanced binary analysis often requires specialized software capable of interpreting hundreds of thousands of assembly instructions and maintaining large databases of cross-references between data and subroutines [7]. This paper presents a solution to combine a variety of freely available binary analysis utilities into a single portable platform. This platform offers web services to distribute the workload of binary processing and therefore minimizes dependencies and resource requirements for performing binary analysis on any given machine or operating system. The topical scope of this project includes binary analysis, malicious software identification, and reverse engineering. Throughout the remainder of this paper, the term 'malware' will be used to describe malicious software.

Objectives

We have found that current tools for centralized binary analysis are often uncommon, expensive, proprietary, non-portable, and difficult to use. To address this issue, we propose OpenBinTools: an open-source distributed system for binary analysis. Our goal is to create an open-source binary analysis platform that is portable, easy to use, modular, and freely available. This software suite should require minimal client resources and dependencies, adhere to development standards, and implement security methodologies to ensure the confidentiality and integrity of users and user data. The OpenBinTools architecture will consist of both client and server applications. Intensive processing and specialized hardware and software configurations will be offloaded to dedicated servers, while client applications are free to perform simplified operations that have negligible resource usage on their respective systems.

As part of our mission, we intend to reuse freely available services for binary analysis and malware identification whenever they are available, so long as additional resource requirements are not incurred within our client software. This work is intended to support the daily activities of reverse engineers, software developers, and security professionals. Research implications of this work include:

- Distributed systems for binary analysis
- Cryptographically secure communications

- Robust communications protocols
- Design standardization
- Software reverse engineering

Related Works

Originally, software engineers developed reverse engineering tools for automating the process of debugging their code and performing binary disassembly. Binary disassembly is a technique in which the bytes of a binary instruction sequence are collected, their operation codes are analyzed, and machine-level instructions are derived [8]. These instructions can be analyzed to determine the control flow sequences within a binary executable, operations performed, I/O behavior, and conditions that are evaluated to determine dynamic behavior during runtime. Today, a variety of software products are available for performing the automated analysis of binary executables, including IDA Pro, OllyDbg, Radare2, Binary Ninja, and countless others. Additionally, Python extensions for exploit development assistance, including PEDA and pwndbg, have been created for even fairly basic hardware debuggers such as the GNU Debugger (gdb).

Reverse engineering and binary analysis have proven to be exceptionally useful for malware analysis and classification. Traditional approaches to automated malware classification involved computing hash signatures, of binary files and comparing them to a virus definition list. Unfortunately, malware authors can simply introduce minor changes to malicious binaries, such as modifying bytes or changing the location or structure of subroutines, and the resulting derivative malware will have a new binary hash, circumventing the virus definition lists containing well-known malware signatures. Newer approaches to malware recognition involve training machine learning models on large sets of malware variants and developing classifiers that can detect malware without needing a pre-computed list of malware signatures, a technique that is becoming even more effective with the advancement of multiprocessing and parallelization techniques [10] [14] [11].

Our work intends to reuse the current state-of-the-art technology by providing dedicated network services for performing remote binary disassembly and malware classification. We seek to improve existing methodologies by creating a cross-platform client to communicate with these network services without the need for advanced binary analysis processing locally. We must also acknowledge various digital forensic utilities that accomplish similar tasks and capabilities to that of the software included within our suite of binary analysis tools, such as readelf, objdump, strings, and file. While these tools perform exceptionally well in their supported environments, we attempted to improve on them where possible by providing alternative versions that are more portable to non-Linux environments.

SOFTWARE SUITE

The OpenBinTools software suite is comprised of a variety of utilities written in the Python and C++ programming languages. We chose Python due to the flexibility acquired by executing in a portable virtual environment, which allows the majority of our software to run on any operating system with Python support. Specifically, we selected the Python3 variant of Python to be forward compatible and to harness modern innovations of the Python language and interpreter. OpenBinTools includes a single C++ utility for monitoring file accesses attempted by all processes on a system. Prior experience with reverse engineering has convinced us that such a tool is novel and helpful for dynamic binary analysis. The utility is written in C++ due to its low-level nature of operation, which requires hooking Linux shared library function calls for file access notification. Therefore, this tool is only supported in the Linux environment. We did not create a Windows version of this utility because similar freely available software exists in the Windows SysInternals suite. The following subsections describe the utilities included in the initial release of the OpenBinTools platform.

Client

The OpenBinTool client (OBTClient) is a platform-independent utility with minimal dependencies. Users simply need Python3 installed on their systems to utilize the functionality of the client. Two modes of operation are provided, a command line interface (CLI) and a read-evaluate-print-loop (REPL) interface. A variety of options can be performed on the OBTClient without requiring interaction with the server-side application, including file identification, string identification, and file header parsing. When the server is accessible through a network connection, the client has the ability to upload files to the server for analysis. Then, the client can issue commands for the server to perform,

such as assembly or disassembly routines, virus analysis, and interaction with Radare2 and PwnTools python libraries. Note, these libraries do not need to be installed on the client machine, providing for greater portability.

The CLI allows for the OBTClient to be used as part of scripting tasks. It initiates the client for a single pass over the binary performing actions signified as command line arguments. The REPL allows for continuous interaction with the OBTClient and a persistent socket connection with the OBTServer. Commands can be issued individually, multiple binaries can be analyzed during a session, and users will continue to receive an interactive interface until the command to quit is issued.

Server

The OpenBinTools server offloads dependencies and intensive processing from the client. It is capable of processing connections and interactions with multiple clients in parallel. The core functionalities of the server include:

- Performing assembly and disassembly of uploaded files
- Identifying functions within a binary executable
- Determining binary security mechanisms
- Analyzing the addresses and ranges of binary sections
- Displaying Linked-Library information
- Identifying the virtual and physical addresses of the main routine
- Piping commands to Radare2
- Interfacing with the VirusTotal API to scan binaries for viruses

The OBTServer is designed to execute within a Linux environment. An installation script is provided and will install all of the required dependencies. The fully qualified domain name (FQDN) or IP address of the server should be logged so that the client can be configured accordingly. By default, the service will listen on port 11337; however, this can be adjusted to fit the needs of the hosting environment. Note: firewall management and port forwarding may be required depending on the hosting environment. Once listening, the server will spawn individual threads to service each client connection. The server implements a custom application protocol that is preconfigured in the client.

AccessNotify

AccessNotify is a standalone utility included in the OpenBinTools suite for binary analysis. The goal of AccessNotify is to hook Linux shared library function calls and system calls for file access operations and to determine the I/O behavior of processes during dynamic analysis. This works by using the Linux fanotify API for intercepting notifications and file system events, including requests to open, read, and write to files. Some of the notable information extracted from events include the file being accessed, the process ID of the accessor, and the file operation being performed. This information is aggregated into data structures designed to store all file access operations performed by each process. Upon receiving the appropriate signal from a user, the AccessNotify process stops and the monitored file access operations are organized by process and displayed.

StringTool

StringTool identifies byte sequences within a binary that are within the range of printable characters, similar to the strings utility available on Linux systems. Users have the option to provide a tolerance specifying the number of consecutive characters required to constitute a string. Additionally, this utility provides the number of occurrences that each string was observed and avoids printing duplicates of the same string.

FileTool

FileTool contains the magic numbers, offsets, file extensions, and descriptions for 140 different file types. This tool analyzes the byte-stream of a binary and identifies any matching file signatures. It is useful for determining file information in environments where the Linux file utility is not available. If a binary file does not match any signatures in the FileTool database, it is likely that the input is either raw data or text.

LoaderTool

The OpenBinTools LoaderTool is capability of parsing Executable and Linkable Format (ELF), MS-DOS, and Portable Executable (PE) file headers. This information includes but is not limited to:

- Magic Numbers
- Architecture (32 - 64 bit)
- Byte Ordering
- File Type

- Entry Point
- Program Headers
- Section Headers
- Compilation Timestamps
- Relocation Positions

VirusTotal API

The OpenBinTools platform integrates a VirusTotal API, enabling users to submit files to VirusTotal for analysis using a centralized API key managed by the server. Clients do not need their own API keys. Binaries are uploaded to the OpenBinTools server, analyzed by a large set of modern anti-virus utilities, and results are returned to the client. This enables the client to quickly evaluate files for potentially malicious recognized content, without executing the binaries locally.

SYSTEMS ARCHITECTURE

Best practices for concurrent, parallel, and distributed computing were carefully considered during the development of the OpenBinTools software suite. To minimize resource loads, the client application operates within one process using a single CPU thread. The server processes multiple clients in parallel on a multicore system designed for high performance. The workload of processing is distributed between the client and server applications based on resource requirements and system dependencies. All applications implement signal handling for interrupt requests by safely closing any relevant files or sockets before deallocating dynamically acquired resources and terminating respective processes. Exception handling is implemented to handle any runtime errors that may occur, preventing instability or system crashes. If a network error occurs during client processing, the network connection is closed and the client reverts to an offline mode. Furthermore, unstable connections with one client will not affect the other sessions of the server.

Security Considerations

During the implementation of the OpenBinTools platform, we faced the dilemma of creating a secure system or keeping client side requirements as minimal as possible. We identified man-in-the-middle (MitM) attacks as a potentially catastrophic vulnerability for end users of our platform. The National Institute of Standards and Technology's (NIST) Computer Security Resource Center (CSRC) describes MitM attacks as, "An attack where the adversary positions himself in between the user and the system so that he can intercept and alter data traveling between them."[1] The OpenBinTools original implementation did not provide secure communication between the client and server and was therefore susceptible to MitM attacks. A adversary with malicious intent could manipulate data in transit; say to change the result of a virus check from positive to negative, leading the end user of the platform to unknowingly execute malicious software. Due to the MitM attack threat, we implemented cryptographically secure communication between the client and server using Diffie-Hellman key exchange, symmetric keys, ephemeral keys, and the Blowfish encryption and decryption algorithm using cipher-block-chaining mode. To avoid added infrastructure complexity and client side requirements, we selected symmetric keys over asymmetric keys for our cryptography suite. The Diffie-Hellman key exchange algorithm is used to facilitate the creation and agreement of symmetric keys when a client connects to the server. We followed the design principle of ephemeral keys, or short lived keys, when implementing the secure communication. Ephemeral keys are only used for a short period, then new symmetric keys must be generated. This helps to provide perfect forward secrecy, as obtaining the symmetric key will only allow access to the data transmitted during the short lifetime of the key. The short lived nature of the keys allows for the key size to be much smaller, leading to faster key creation. Currently, OpenBinTools performs a key exchange and generates a unique symmetric key for every new connection to the server. Future work could include also using a set time-to-live for the keys in case a connection is held open. Using ephemeral keys per connection, we decide to use the Blowfish encryption and decryption algorithm due to its ease of use and excellent speed [13]. As the focus of this project was not the securing of communication, any potential users of the platform should thoroughly review and weigh all mentioned security concerns before proceeding in platform utilization.

The platform also uses temporary files on the server to handle data loaded by clients. To avoid race condition vulnerabilities due to temporary file creation and the permissions in the /tmp directory, we create cryptographically secure unique random file names for every new temporary file. Upon completion and exiting the

platform, temporary file handling and deletion is thoroughly handled.

Application Protocol

Due to varying endianness amongst systems, network streams require special consideration when parsing. Additionally, arbitrary sized packet streams, along with sequential requests, can cause parsing issues. In order to establish reliable communications between the client and server, a custom application protocol was required. The first 4 bytes of a datastream are an unsigned integer value stored in little endian. This value defines the number of bytes to read in the remaining message. The server always responds with a status code indicating the success or specific error encountered during processing.

STANDARDIZATION

OpenBinTools incorporates the standardization practices of Pylint, PyCharm, and PEP8. Pylint [4] is a code analyzer that scores Python code on multiple categories. PyCharm [3] is an IDE for writing quality Python code and PEP8 [2] stands for Python Enhancement Proposal 8, which is a coding convention for Python. The use of these standards contributes to the following characteristics of OpenBinTools:

- Code maintainability
- Code portability
- Readability
- Ensuring best practices
- Security

Code maintainability ensures that our code contains less bugs while code portability strengthens our goal of making OpenBinTools portable and provides higher compatibility between various environments. Through ease of understanding our code via readability, vulnerabilities or bugs in the code will be decreased. By ensuring best practices and using standardization for the purposes mentioned above, the overall security of OpenBinTools is enhanced.

EVALUATION

Our original goal was to deploy a client with zero dependencies beyond Python 3. During our initial deployment, we found that the pycrypto Python package was required to implement the cryptographic features desired to achieve authentication and integrity. In an effort to promote security, we compromised and forced the client to include this dependency. On the Windows system, we found that in order to do so we needed to first install the Microsoft Visual C++ Compiler for Python 2.7. Then, steps needed to be taken to ensure pycrypto was installed successfully, including executing:
set CL=-FI"%VCToolsInstallDir%\include\stdint.h"
to set a command line environment variable required for a successful installation. Finally, pycrypto could successfully be installed on the Windows environment.

Additionally, the VirusTotal API would sometimes queue analysis for binaries it had never seen before. This resulted in the necessity to call the utility a second time to acquire virus recognition results, since the results were not available during the first query to the API.

We also found that disassembly routines for Microsoft Portable Executable files is not currently supported. Libraries we used to perform disassembly only provide support for Linux Executable and Linkable Format at this time. We found that PE disassembly can be achieved by using the Linux objdump utility, which may be included in future versions of our software.

Otherwise, the OpenBinTools platform met our goals for implementing a distributed system for binary analysis. The server supported multiple concurrent connections from clients executing on a variety of operating systems. File, string, header, and virus recognition performed well, assembly and disassembly routines were successful, and we successfully piped commands from clients to a Radare2 instance running on the server. Our implementation established the groundwork for an open-source, portable, and extensible binary analysis platform.

Future Work

In future iterations of the OpenBinTools platform we hope to make strides towards supporting, researching, and implementing the following features:

- Symbolic execution [12] [9]
- Integration with Windows SysInternals Suite
- Utilization of Radare2's webserver to provide visual binary analysis accessible through the server over HTTP
- Improve REPL with *nix commands like: ls, pwd, cd

- Improve CLI with flag for formatting output as a JSON object
- Provide support for Mac OS X
- Rigorous review of security

CONCLUSION

OpenBinTools is an open-source distributed platform for binary analysis. By including a client-server architecture, the client services are easily distributable to multiple machines running minimally Python 3. A single server may service multiple clients concurrently. The tools provide easy to use interfaces with help menus and various modes of operation. To support future growth and platform independence, the software in this suite adheres to the latest development standards. This tool is intended to be useful for reverse engineers, software developers, and cybersecurity professionals.

REFERENCES

[1] Man-in-the-middle attack. National Institute of Standards and Technology Computer Security Resource Center, *https://csrc.nist.gov/glossary/term/man_in_the_middle_attack*.

[2] Pep8. *https://www.python.org/dev/peps/pep-0008/*.

[3] Pycharm. *https://www.jetbrains.com/pycharm/*.

[4] Pylint. *https://www.pylint.org/*.

[5] State of Software Development. Coding Sans, 2018. [Online Document] [Cited 2018 Nov 25] Available HTTP https://codingsans.com/uploads/landing/State-of-Software-Development-2018_final.pdf.

[6] Aiken, Peter et al. DOD legacy systems; reverse engineering data requirements. *Communications of the ACM*, May 1994, p. 26+ Academic OneFile. https://link.galegroup.com/apps/doc/A15353016/AONE?u=googlescholar&sid=AONE&xid=d046a7ee. Accessed 25 Nov. 2018.

[7] Brumley D., Jager I., Avgerinos T., Schwartz E.J. BAP: A Binary Analysis Platform. In: Gopalakrishnan G., Qadeer S. Computer Aided Verification. CAV 2011. Lecture Notes in Computer Science, vol 6806. Springer, Berlin, Heidelberg. 2011.

[8] Coakley C., Freeman J., Dick R. Next-Generation Protection Against Reverse Engineering . Anacapa Sciences, Inc., 2005.

[9] J. D. Miller. Binary disassembly block coverage by symbolic execution vs. recursive descent. Master's thesis, Air Force Inst of Tech Wright-Patterson AFB OH School of Engineering and Management, 2012.

[10] R. Searles et al. Parallelization of Machine Learning Applied to Call Graphs of Binaries for Malware Detection. 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), St. Petersburg, 2017, pp. 69-77.

[11] S. Petit R. Ubal, J. Sahuquillo and P. Lopez. Multi2sim: A simulation framework to evaluate multicore-multithreaded processors. *19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*, Rio Grande do Sul, 2007, pp. 62-68.

[12] Springer J. M., Feng W. Teaching with angr: A Symbolic Execution Curriculum and CTF. Portland State University Department of Computer Science, 2018.

[13] Swati Singh and Sudhir Kumar. Analysis of various cryptographic algorithms. *International Journal of Advanced Research in Science, Engineering and Technology*, Vol. 5, Issue3, March 2018, pp. 5341-5349.

[14] Yan G., Brown N., Kong D. (2013) Exploring Discriminatory Features for Automated Malware Classification. In: Rieck K., Stewin P., Seifert JP. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2013. Lecture Notes in Computer Science, vol 7967. Springer, Berlin, Heidelberg.