Obfuscation-Resilient Static Analysis for Binary Malware Detection

Sean Hodgson, Emily West, Tashi Stirewalt, and Assefaw Gebremedhin

ABSTRACT

As cyber threats become more sophisticated and our dependence on software systems increases, it is crucial to implement robust security measures. One significant aspect of software security is analyzing binary code files, often targeted by cyberattacks. Although various techniques for **static analysis of binary code files** are available, considerable challenges remain. These challenges include but are not limited to, overcoming code obfuscation techniques, handling dynamically generated or self-modifying code, scaling to large codebases, heavy reliance on domain expertise, and high rates of false positives and negatives in practice. In this study, we propose an **tournament selection style dynamic programming LCS algorithm for learning opcode sequence signatures** of similarity between different binary files known to be similar. Using similarity measures of **Cosine, Jaccard, and Damerau-Levenshtein**, we explore the threshold prediction approach. In addition, we explore a **vectorization process using a Bag-of-Words (BoW), TF-IDF, and statistical opcode combined feature embedding to train a Random Forest ML model** to classify binary files into a multiple-class environment. Our experimental results demonstrate that the approach shows great potential and lays a solid foundation for efficient prediction tasks for detecting similarity between binary files resiliently to obfuscation.

RELATED WORK

The field of static analysis of binary code files has undergone significant research. Key papers include Bin Zeng's work on **foundational static analysis techniques** [1], Nikos Karampatziakis's model using structural SVMs [2], and Young-Hyun Choi et al.'s **framework for taint analysis** [3]. Subsequent studies have introduced machine learning to binary code analysis, such as Azadeh Jalilian et al.'s **static signature-based malware detection method** [4] and Rajchada Chanajitt et al.'s **multiclass malware classification using opcode sequences** [5]. However, each approach has its limitations, such as scalability, complexity, and heavy reliance on domain expertise. Despite these challenges, each paper contributes to the development of more secure and reliable software systems.

WASHINGTON STATE UNIVERSITY

OPCODE MNEMONICS

Opcode mnemonic sequences are fundamental components of binary executables, as they **represent the specific operations that a CPU will execute**. These sequences can be obtained using "disassemblers", and they are essential for analyzing binary files. Comparing opcode sequences **can help identify similar operations between binaries**, even if they are obfuscated or self-modifying. Different processors have unique opcode sets, and manufacturers ensure opcode compatibility across generations. A computer instruction begins with the **opcode and specifies the memory address of the operation**. Mnemonics, which represent opcode functions, make working with opcodes more straightforward.

0	1	1	0	0	0	1	0	1	0	1	1
Opcode				Address							

Example bit arrangement of a computer instruction.

Opcode	Operation	Mnemonic
000	Stop the program execution	stop
001	Get a value from a specified memory location and store it in register A	getA
010	Get a value from a specified memory location and store it in register B	getB
011	Copy a value from a register to a specified memory location	put
100	Skip to different instruction line	skip
101	Add the register value of A and B and place the result in memory	addAB
110	Increment the value in register A by 1	incA
111	Complement the value in register A	cmpA

Example opcodes for operations w/ assoc. mnemonics.

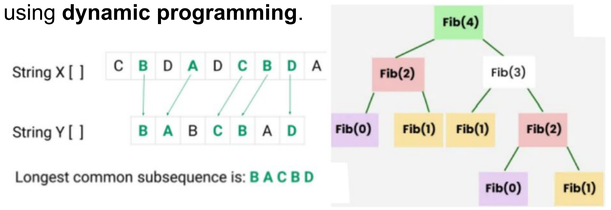
MALWARE CLASSES

The **Dike Malware Dataset**, created by a GitHub user named iosifache, contains **both benign and malicious PE and OLE** files. The dataset was created by downloading PE files, sorting them into malicious or benign categories, and renaming them using their hash. Malicious OLE files were **downloaded from MalwareBazaar**, and benign ones were manually downloaded. A Google Cloud Function was used to **scan the hashes with the VirusTotal API**. The dataset is useful for training AI algorithms, with numeric labels that can be transformed for standard classification. All malware executable binaries in the dataset were **downloaded using a Ubuntu Virtual Machine**, and their **opcode sequences were extracted** for analysis.

type	hash	malice	generic	trojan
0	936c36121f73b1	0.8837209302	0.4285714286	0.428
0	f5678e3e52773a	0.9007633588	0.527027027	0.378
0	ec1f7e652aa8f4	0.9253731343	0.1785714286	
0	a616f9ac37c0a1	0.8636363636	0.453125	

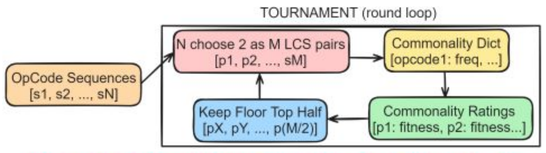
LCS EXTRACTION

The **Longest Common Subsequence (LCS)** algorithm is a computational problem that finds the longest subsequence common to all sequences in a set, with subsequences not required to be consecutive. The LCS problem has an optimal substructure and overlapping subproblems, typically solved using **dynamic programming**.



Dynamic Programming "Fib" function for finding the Fibonacci sequence.

Tournament selection is a technique used in **Genetic Algorithms** to select the fittest individuals from a population for crossover. It involves **conducting tournaments among individuals**, with the **winner** (the one with the best fitness score) **selected for crossover**.



"rootkit": ["add", "add", "add", "mov", "push", "mov", "mov", "xor", "and", "xor", ..., "jl"],
"downloader": ["push", "mov", "sub", "mov", "cmp", "je", "cmp", "jb", "jmp", "push", "push", ..., "fidiv"],
"worm": ["push", "push", "call", "ret", "push", "push", "call", "ret", "jmp", "jmp", "jmp", ...,],
"encrypter": ["push", "mov", "sub", "mov", "mov", "mov", "mov", "mov", "mov", "cmp", ..., "mov"],
"generic": ["mov", "jmp", "int3", "int3", "int3", "int3", "int3", "push", "mov", "mov", ..., "pop"],
"backdoor": ["push", "mov", "sub", "push", "lea", "push", "add", "push", "sub", "call", ..., "add"],
"spyware": ["push", "mov", "mov", "mov", "pop", "ret", "int3", "int3", "int3", "int3", ...,]}

Example abbreviated learned class signatures.

"File Name": "00255bab3a31b108a9de13b0944a50b5fa866dc53c139c2da04007b952b719ad",
"Actual class": "backdoor",
"Detect class": "rootkit",
"Cosine Similarity": 0.6271614292508758,
"Jaccard Similarity": 0.13559322033898305,
"Damerau-Levenshtein Distance": 0.0013061650992685476

Example similarity measure result of a file.

VECTORIZATION & ML

The opcode sequences were transformed into **feature embeddings** for machine learning using **Bag of Words (BoW)**, **Term-Frequency Inverse-Document-Frequency (TF-IDF)**, and **basic statistical measures**.

The **Random Forest Classifier** is a machine-learning algorithm used for both **classification and regression tasks**. It is an **ensemble method** that combines **multiple decision trees** to make more accurate predictions.

RESULTS

Cosine

	precision	recall	f1-score	support
0	0.92	0.98	0.95	7890
backdoor	1.00	0.10	0.19	97
downloader	1.00	0.15	0.27	97
encrypter	0.87	0.26	0.40	100
generic	1.00	0.64	0.78	100
ransomware	0.24	0.44	0.31	91
rootkit	1.00	0.07	0.13	100
spyware	1.00	0.21	0.35	100
trojan	0.36	0.05	0.09	100
worm	1.00	0.03	0.06	100
accuracy			0.91	8775
macro avg	0.84	0.29	0.35	8775
weighted avg	0.91	0.91	0.88	8775

Jaccard

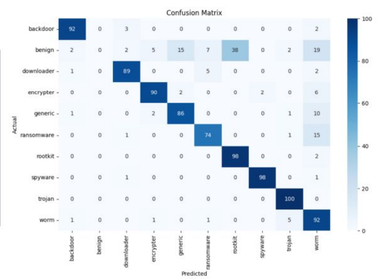
	precision	recall	f1-score	support
0	0.92	0.97	0.95	7890
backdoor	0.42	0.10	0.17	97
downloader	0.75	0.15	0.26	97
encrypter	0.59	0.36	0.45	100
generic	0.98	0.65	0.78	100
ransomware	0.23	0.44	0.30	91
rootkit	1.00	0.17	0.29	100
spyware	0.81	0.21	0.33	100
trojan	0.36	0.09	0.14	100
worm	0.33	0.04	0.07	100
accuracy			0.90	8775
macro avg	0.64	0.32	0.37	8775
weighted avg	0.89	0.90	0.88	8775

Damerau-Levenshtein

	precision	recall	f1-score	support
0	0.92	0.98	0.95	7890
backdoor	1.00	0.10	0.19	97
downloader	1.00	0.15	0.27	97
encrypter	0.87	0.26	0.40	100
generic	1.00	0.64	0.78	100
ransomware	0.24	0.44	0.31	91
rootkit	1.00	0.07	0.13	100
spyware	1.00	0.21	0.35	100
trojan	0.36	0.05	0.09	100
worm	1.00	0.03	0.06	100
accuracy			0.91	8775
macro avg	0.84	0.29	0.35	8775
weighted avg	0.91	0.91	0.88	8775

Random Forest Classifier

	precision	recall	f1-score	support
backdoor	1.00	0.88	0.94	17
downloader	0.86	1.00	0.92	12
encrypter	0.92	0.75	0.83	16
generic	1.00	0.72	0.84	18
ransomware	1.00	0.69	0.82	13
rootkit	1.00	0.95	0.98	21
spyware	0.95	0.95	0.95	19
trojan	0.82	1.00	0.90	14
worm	0.50	0.92	0.65	12
accuracy			0.87	142
macro avg	0.89	0.87	0.87	142
weighted avg	0.91	0.87	0.88	142



When there are numerous well-understood and labeled examples of malware available, the OpCode Sequence approach can be used to extract a signature for each class of malware. This provides a **strong basis for efficient prediction tasks** that can **effectively detect similarities between binary files**, even in the presence of **intentional obfuscation**.

REFERENCES

[1] Zeng, Bin. "Static Analysis on Binary Code." (2012).

[2] Karampatziakis, Nikos. "Static analysis of binary executables using structural SVMs." (2010).

[3] Choi, Young-Hyun . "A framework of static analyzer for taint analysis of binary executable file." (2013).

[4] Jalilian, Azadeh. "Static Signature-Based Malware Detection Using Opcode and Binary Information." (2020).

[5] Chanajitt, Rajchada. "Multiclass Malware Classification Using Either Static Opcodes or Dynamic API Calls." (2022)

ACKNOWLEDGEMENTS

This work is supported by funding for the VICEROY Northwest Institute for Cybersecurity Education and Research (CySER) provided by The Office of the Undersecretary of Defense for Research and Engineering, in collaboration with the Air Force Research Laboratory and Griffiss Institute.