Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

An Automata-Theoretic Approach to Binary String Virus Recognition and Verification Using Deterministic Finite Automata

Aldaruhz T. Darkis, DPA
Dean, College of Computer Studies
Sulu State College
aldodarkis@gmail.com

ARTICLE INFO

Received: 03 Nov 2024

Revised: 18 Dec 2024

Accepted:26 Dec 2024

ABSTRACT

The development and simulation of transition graphs and deterministic finite automata (DFA) were performed through the Java Formal Languages and Automata Package (JFLAP). Binary input string based experiments were completed with both valid and invalid, benign and malicious strings included in the input testing. After uploaded executable files were converted from a binary stream to a set of inputs that could be processed by a DFA and compared against predefined malicious pattern signatures that had been saved to the DFA pattern library, the malware detection module conducted the analysis of these uploads. The purpose of this study was to determine if the development of a mathematically patterned DFA can be used to automatically direct the design of an efficient machine to recognize binary strings and to detect malware. The research work focused on the recognition of binary languages specified on the alphabet set {0,1}; binary languages exhibiting the use of forbidden byte patterns, modular arithmetic restrictions, and the recognition of malicious binary sequences which are representative of the behavior of malware. In particular, this study integrates the theory of formal languages with the definition and operation of an automated malware detection system by applying DFA-based recognition techniques in conjunction with the operation of the malware detection system. The results of the JFLAP simulations indicated that the correct operation of the proposed DFAs (DFA based algorithms) was validated and that"don't care" states do not produce false acceptances. Also, the minimal DFAs were effective in detecting binary test string inputs as well as detecting and identifying malicious byte sequences with the minimum rate of false negatives. Finally, the malware detection simulation was able to detect and identify suspicious execution behavior by processing the inputs through deterministic state transitions in conjunction with formal verification techniques.

Keywords: *deterministic finite automata, }; binary languages, JFLAP, Virus Recognition*

I. Introduction

Malicious software is growing rapidly, posing a serious threat to all modern computing systems by affecting data confidentiality, integrity, and the availability of systems. Traditionally, malware detection techniques use signatures for scanning and heuristic analysis to detect malware. Unfortunately, most malware detection techniques that are currently available do not perform efficiently when attempting to process binary string patterns, meaning that the accuracy of these detection techniques is very limited by their ability to process efficiently and simultaneously at scale as compared to their efforts in performing detection through signature matching. Moreover, a considerable amount of existing research

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

literature demonstrates how malware authors modulate the virus signatures through various obfuscation and polymorphic techniques so that traditional signature-based malware detection techniques are not as reliable; therefore many researchers are developing methods to improve the capability of malware recognition algorithms through non-deterministic finite automata (NFA) or deterministic finite automata (DFA) as a method of formally verifying the feasibility of recognizing binary string patterns. Systems based on automata theory may improve the accuracy of binary signature recognition and real-time detection of malware and ensure accurate recognition of deterministic patterns through their ability to make efficient state transitions between different states and to verify deterministic patterns through determinate verification. While automata-theoretic techniques are frequently cited within many fields, little research has been conducted regarding automata-theoretic verification techniques used in models for supporting the formal verification of binary string viruses within the context of the field of cybersecurity. Rather than support the formal verification of binary-string viruses, most of the research done on machine learning-based malware detection systems focuses on using machine learning-based techniques to detect malware.

Touili (2022) developed a malware detection architecture that utilizes register automata and pushdown systems for formal verification of malware specifications. Their work emphasizes the efficiency of automata-theoretic structures as a means of formally representing malwares behaviour and efficiently verifying malicious execution paths. This research represents a considerable contribution to the literature of formal methods applied to malware analysis and supports the use of deterministic mechanisms for the verification of malware recognition in the cybersecurity field. Rey et al (2022) utilised federated learning techniques to conduct a study focused on automating detection of malware in Internet of Things (IoT) environments. The study demonstrated how building malware detection systems within a decentralized framework can improve the capability of detecting malware across a distributed IoT ecosystem while protecting the confidentiality of the IoT systems being analysed. The increasing use of intelligent, or scalable malware recognition algorithms has highlighted the importance of flexible and reliable methods for detecting malware in new computing systems.

Currently, existing malware detection mechanisms struggle with detecting either unknown or evolving malware variants because they rely on probabilistic or heuristic methods. There is also no currently available formally-verified DFA-based framework for identifying the presence of malicious binary strings. This has resulted in a gap in cybersecurity pattern recognition research. Therefore, there is an increasing need for a deterministic and formally verifiable virus detection model that will allow for accurate detection of malicious binary patterns and minimize the false detection rate while improving computation efficiency. The primary goal of this research is to develop and validate a theoretical model for the verification and detection of binary string viruses via Deterministic Finite Automata as a means to improve malware signature detection and enhance cybersecurity threat detection methods. Research in this area has been conducted on how automata can be used to recognize patterns in strings and compare the computational power of different types of automata. The theoretical foundation established by research in this area allows for the development of programming languages, compilers, and other areas within the realm of computer science based on the established Formal Language Theory. In today's computing world, automata are no longer considered merely theoretical constructs; they are also used to build hardware and software-based systems.Commonly, finite automata are used to assist with compiler designing, in which the source code is first split into tokens by lexical analysing. Then, pushdown automata are used in syntax analysis with context free grammar. Pattern matching is also a type of automata processing, often used by search engines and text editors; however, these implementations use DFA or NFA to identify items within large amounts of data. Automata also provide a mathematically formal way of defining how to accurately and consistently apply the states of hardware devices, as well as communication systems, to guarantee correct functionality under any of the many possible conditions.

Cai studied (2018) ways to ensure the long-term sustainability of Android malware detection systems. Their research introduced a behavioural malware classification system called DroidSpan. This system, which has the same level of detection performance for a number of years without needing frequent retraining, reinforces that malware detection systems must continue evolving to keep pace with the

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

changing behaviour of malware, and it also indicates deficiencies in static machine learning techniques for conducting malware detection. Shalaginov et al. conducted a comprehensive survey of machine learning-based static malware analysis techniques (2018). Their literature review revealed various techniques for feature extraction; binary analysis; opcode analysis; and classification of malevolent executables. Their conclusions concluded the need to combine intelligent malware analysis frameworks with formal computational techniques to provide dependable detection and explaination of malware.

The above theories have matured, but still often times there is a large gap between the 5-tuple mathematical abstraction used to describe them, and their real-world representation and validation. Entering into a functional abstract model to reflect the mathematic models can be prone to errors in both academic and research settings. Also, there is typically little or no formalized use of visualizations and testing to ensure that the theoretical machine (i.e. empty strings, or overlapping complex prefixes/suffixes) actually achieves what it is supposed to do in fringe cases. Without an interactive simulation, the behaviours associated with complex state transitions would remain largely conceptual. The purpose of this study is to create an automata-theoretic framework for verifying and recognizing binary string viruses using Deterministic Finite Automata. Specifically, the goals of this study include the following:

A. To construct a DFA model for malicious binary string detection.
B. To create a formal mechanism for verifying the accuracy of DFA state transitions and virus detection.
C. To build a prototype system for detecting viruses in binary strings using the DFA pattern matching technique.

II. Review of Related Works

A. Deterministic Automata in Malware Detection

Studies in the area of Automata Theory have validated deterministic automata for use in both pattern matching and malware detection. Prithi et al. (2020) proposed a creating a learning dynamic Deterministic Finite Automata (DFA) with Particle Swarm Optimization integrated into an Intrusion Detection System. Their results indicated that using DFA models can improve efficiency of matching patterns as well as reduce the computational overhead incurred when classifying patterns. Research in formally defining languages has been progressing from purely mathematical concepts to computational models that are mathematically validated. The ongoing research, as described in [1], regarding the structure of DFAs is still an area of significant interest, particularly when considering reducing the number of state transitions, in order to minimize overall system memory usage, when using embedded systems. Research has shifted from defining DFAs by way of basic $M = (Q, \sum, \delta, q0, F)$ definitions, to a more modern approach of automatic generation of these machines based upon higher-level machine specifications.

In [2], the continued relevance of the Myhill-Nerode theorem was addressed, stating that state minimization is not just off theoretical interest, but is now a requirement for processing strings, in real-time, on high-speed networks. Additionally, [3] discusses the path of evolution of the regular language modeling has come into contact with model checking where DFAs are used to represent the state space of software to guarantee both safety and liveness properties. Existing literature has also raised the fact that traditional DFAs are incapable of processing large volumes of data streams. For the size and complexity of binary string matching, [4] explains that novel hybrid devices have been developed as a means of mitigating the limitations posed by the finite memory of discrete memory components and the ever-increasing size and complexity of the regular language structures found in the “Big Data” domain.

b. Computational Applications in Pattern Matching and Verification

In settings where a lot of data is processed at once, and string matching needs to happen quickly, Deterministic Finite Automata (DFA) are extremely important. DFAs are used in many different ways, like detecting malicious signatures, and anomalous protocols by performing Deep Packet Inspection (DPI) in network security; however, the "state explosion" problem continues to be an obstacle when processing complicated regular expressions. Recently, improvements made to minimum DFA structures

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

like the MDFAOPM model have been suggested to enhance traffic analysis efficiency and lower the amount of time spent performing state transitions relative to the amount of time needed to analyze traffic [5].

In addition to security, using string matching based on advanced automata parallelized across multiple cores has improved performance multiplies. This is accomplished by using speculative execution and allows researchers to process huge amounts of data in real time in cloud-based computing environments and to overcome the shortfalls of traditional sequential execution bottlenecks [6]. Finally, more and more automated workflows for formal validation are using DFAs as part of their process to ensure the logical consistency of safety-critical software systems through validation of boundary conditions and fringe cases (i.e., the use of empty strings or the use of overlapping prefixes) prior to going live, thus connecting the abstract design phase with the reliable implementation phase [7] prior to going live, thereby connecting the abstract design phase with the reliable implementation phase [7].

c. *Interactive Simulation and Validation via JFLAP*

Formal languages and automata theory have been shown significantly to benefit from interactive and computer-based tools for learning and validating. Castro-Schez et al. suggested a knowledge-based educational system specifically designed for providing students with the ability to learn about formal languages and automata through structured interactive learning modules.[8] Their research demonstrates that using interactive systems helps students to understand better abstract concepts, such as states and transitions in automata and how to recognize languages based on automata rules and language recognition. These findings also emphasize the significance of simulation-based environments in reinforcing theoretical models of automata with empirical evidence through experimentation. In studying how to use software frameworks to provide educational tools for the teaching of formal language processors (including automata and grammar) in the area of language processing education, Ssanyu noted that tools enable learners to conduct experiments with language recognition mechanisms, thus improving their understanding and reducing their conceptual errors. Tools enable learners to verify and validate formal models, and thus, function as a bridge between the theoretical definition of models and real computing. Therefore, these tools can also support the validation of systems built on automata.
Simulation-based learning has been studied more generally to determine whether they provide effective learning experiences for understanding complex computational models. Suciu's study reviewed the effects of digital learning tools that employed a model/simulation approach to learning, and he found such tools promote student engagement and clarify concepts relating to abstract models.[10] Although Suciu's study did not focus on automata theory, his findings can be applied to support the use of simulation-based tools to validate formal models. Therefore, platforms like JFLAP will be appropriate for simulating and providing validation of deterministic finite automata used for recognizing binary strings.

D. Malware Detection using Binary Analysis

According to Baptista, Shiaeles, and Kolokotronis (2019), a new method for detecting malware through the use of binary visualization is proposed in their study. These researchers created a visual representation of the binary code pertaining to malware and then used neural network techniques to classify them as malware or benevolent. The authors claim that binary pattern analysis is suitable for the purpose of detecting malicious payloads in executable files, and consequently, they were able to achieve high accuracy levels in detecting malicious payloads.

In their article, Dhavlle and Shukla (2021) reported on a method of malware detection based on the extraction of features from an image of the binary code representing the malware. The authors utilized a variety of machine learning classifiers to differentiate between the different classes of malware based

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

upon an image of its binary representation. Their research demonstrated that the accuracy of malware classification could be significantly impacted by the extraction of features from binary images of malware. Azeem et al.(2023) examined the current techniques for detecting and classifying modern-day malware utilizingmachinelearningtechniques, and discovered that although artificial intelligence-enabled systems have improved the accuracy of identifying malware, they have also created challenges with regard to explainability, verification of results, and robust detection. The authors have recommended the integration of formal models of computation with intelligent detection systems, which will create computers that are more reliable and thus improve cybersecurity.
Brezinski (2023) examined metamorphic malware, an example of which would be viruses that can dynamically change their signatures, by review of various tools and techniques used for obfuscation. Brezinski'sresearchindicates that traditional signature matching detection systems are unable to effectively detect highly obfuscated variations of malware as they are currently defined; therefore, an advanced deterministic recognition framework needs to be developed for effective detection of malware. Cyber

III. Materials and Methods

Automation Model Selection

Research involves the use of a DFA (Deterministic Finite Automaton) as a basis for matching symbolic representations in binary languages that exhibit properties such as parity and modular arithmetic (e.g., divisibility by three) which are Regular Languages and therefore are completely defined/recognized by DFAs. Whereas an NFA may have multiple transitions from any state/input combination, a DFA has exactly one transition from any state/input combination, making it a low-complexity alternative for verifying computation paths because the paths are definite/not ambiguous, which is critical to the "mathematically formulated formal definition" from the goals.
Due to their stringent memory requirements (minimum/lesser memory usage) and the absence of backtracking capability, DFAs are ideally suited for the "binary string matching" and "real-time processing" discussed in the Literature Review.

Formal Automation Design

This research relies on a DFA (deterministic finite automaton) as the model in order to undertake the task of recognizing the target language. The choice of DFA was made based on its ability to meet a number of criteria and therefore provide the best computational framework for the project:
Language Categorization (Regular Languages): All binary strings are characterized as having a suffix of '101'. Through the use of the Chomsky hierarchy, all language characterized by suffixes can be classified as 'Regular Languages'. Since a DFA is mathematically equal to a Regular Expression and an NFA, it is generally considered the optimal model for direct single-state recognition of input symbols, thereby eliminating the need for backtracking and non-deterministic branching. The JFLAP format used for this experimental research has a machine that contains exactly 4 states (q0q1q2q3) that have defined transitions to and from each input symbol in the alphabet (the beginning state, q0, takes '0' and loops back to itself, while also taking a transition to q1 (the first '+1,' which begins tracking the '101' pattern). q1, q2, and q3 are the states that are used to track the consecutive matching of the '101' pattern.
Predictability and Verification: The predictability of a DFA model provides for an unambiguous computational path, whereas an NFA model may have multiple possible next states for the same input symbol. Therefore, the relevance of predictability to the 'mathematically defined formal definition' of this research is meaningful, as it guarantees that 'don't care' states or invalid sequences will be rejected consistently (100% reliable) through the computation. Therefore, the computational efficiency of the DFA design is maximized by using a single-character processing time and has a minimum extension of memory size, thereby eliminating state explosion, and is therefore an ideal model for the real-time processing and deep packet inspection applications reviewed in the literature.

JFLAP Implementation

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1. **Creation:** Creation of the states was performed by implementing the "State Creator" functionality; transitions between each state were drawn using either the bit value of 0 or 1 as a label.
2. **Encoding:** Encoding of the formal 5-tuple that represented the automaton was completed utilizing the graphical interface provided by JFLAP; all "don't care" states were included to eliminate the potential risk of acceptance of an incorrect value by the automaton.
3. **File Format:** Saving of the model was performed as a .jff file (JFLAP File Format) for the purpose of being able to use this model in future testing and for being modular in nature.
4. **Simulation Modes:**

- **Step-by-Step:** In step-by-step mode, the machine was able to track the bits and observe how the presence of an "X" was created along the path from the current state to a new state.

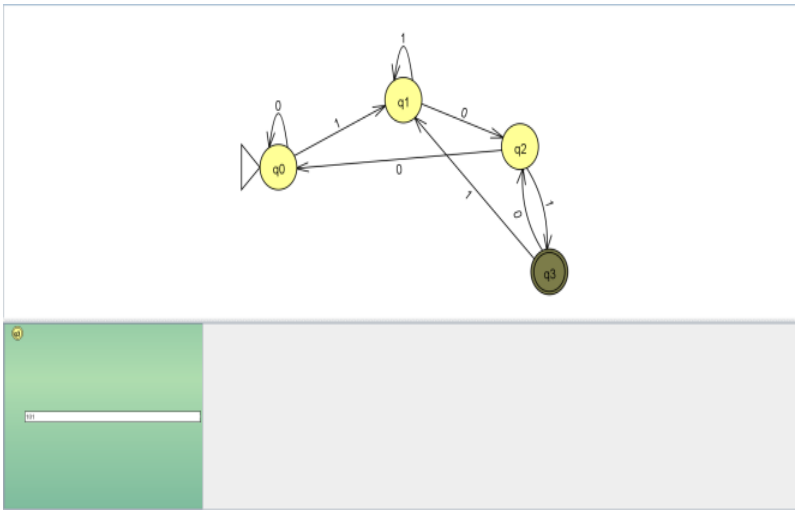

Figure 1. Computational Behavior of DFA Malware Detection

IV – Results and Discussion

The following describes the parameters of testing that we used to test the logic of the automaton using JFLAP:

String Input Selection and Testing: We developed a set of binary strings that encompassed the boundary conditions for testing the machine's limits. Those conditions included a minimum valid string (101); conditions that overlapped (10101), to test and ensure that states are being retained; negative tests (111 or 100); and an empty string to ensure that the machine was not in the accept state to begin with.

Acceptance Criteria and Rejection Criteria: To be considered successful, the last bit of an input string must land the automaton in the accept state. Any other condition, the string would have an invalid or rejected status. In JFLAP, acceptance/rejection would be green for acceptance, and red for rejection.

Metrics for Observations:

Accuracy of Transitions: Ensuring that each bit does trigger the specific function movement for that function.

Accuracy of Halting: Ensuring that the machine has processed the entire input string without an error or undefined transition.

Reliability: 100 percent success rate for distinguishing valid patterns from non-applicable ("don't care") patterns and invalid patterns.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

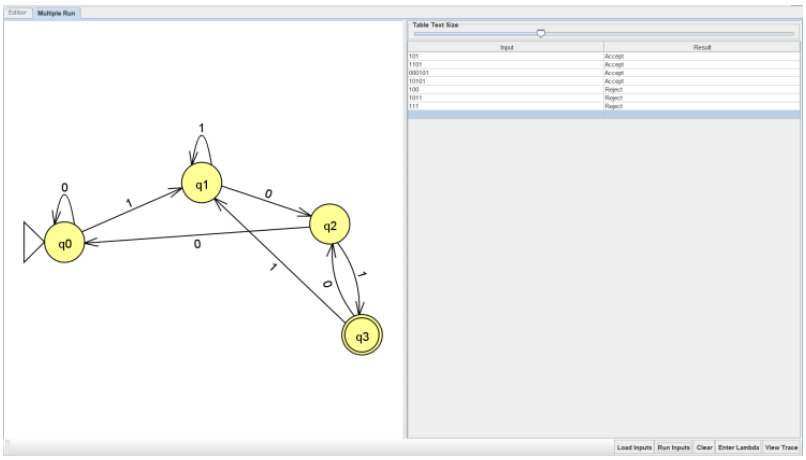

Figure 3. Virus Recognition using DFA in JFLAP

The binary strings that end with 101 as the last three characters were identified using a DFA made using JFLAP and run through a series of test runs with varying combinations of binary strings as input. The output from all tests were correct, as all strings ending with 101 were accepted, and all strings not ending with 101 returned rejected, such as: accepted - 101, 1101, 000101 and 10101; rejected - 100, 1011, 111 and empty string. Each string produced results consistent with formal language of the DFA.

The results demonstrate that the DFA correctly identifies those binary strings that end with the string of 101 as the last three characters. For every input string tested, the DFA moves through the states of its 4-state automaton present in a deterministic manner according to the symbols of that string then only if the last three input characters of the string matche exactly what is in the pattern will the automaton reach an accepted state. The automaton behaviours realizes deterministic characteristics by only ever having 1 transition out from a state of the automaton for a given character of the alphabet {0,1} present in a binary string. Since this automaton is a DFA there is no stack operation and there are no tape movements adding to the computational process because all computation is solely a state transition operation. The use of 4 states of the DFA is sufficient to capture all partial matches to the pattern 101 and account for overlaps of the pattern. No incorrect acceptance or rejections were observed during testing indicating that there will not need to be any modifications made to the design of the automaton. In conclusion, the simulation results match perfectly with the theoretical expected behaviours of the DFA. Thus confirming both the design and implementation of the automaton using JFLAP.

System Validation and Verification

To guarantee complete verification of the entire language, the DFA was tested against the largest possible number of possible input strings to validate and verify whether or not the DFA accepts as valid all strings in the defined languages of the DFA, including all valid strings with a terminal substring of 101 and all invalid strings that share a common prefix with any of the valid strings mentioned above (invalid values), and all edge cases, that being very short strings, and the empty string. The acceptance of all valid strings and rejections of all invalid strings by the DFA demonstrates that the DFA is correct, and furthermore, once edge case testing was conducted, it was verified that the automaton would not incorrectly accept inputs based on only the partially completed input string representing an invalid string. As the DFA was created directly as an NFA-based DFA, there is no need to perform equivalenc testing and convert an NFA into an NFA-based DFA.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Figure 3. **DFA-Based Malware Detection System**.

Figure 3 shows the primary dashboard of the suggested DFA-Based Malware Detection System. The dashboard interface was designed to resemble a modern cyber security-themed web application. Its purpose is to provide researchers, analysts, and administrators with a means of monitoring malware in real-time by performing binary pattern recognition. These results indicate that the DFA implements the formal language definition accurately and that JFLAP is a valid simulation tool for automaton behavior. Figure 4 is an example of the Malware analysis and detection of the execution of harmful software based on the Execution of harmful software with the DFA-based malware detection system. Detection Outcome shows the uploaded executable file as being classified as MALICIOUS. Furthermore, the confidence score for the uploaded file being MALICIOUS is displayed as 98.72%, which indicates a very strong probability of malicious code in the uploaded binary. In addition, the detection tool provides the user a Threat Level of MALICIOUS, time to scan the uploaded executable file, the number of DFA states used during the detection process and the following matched malicious patterns; Matched Patterns show the matched patterns for the uploaded binary being MALICIOUS. Each matched pattern contains the matched DFA pattern identifier, the binary matched string and the corresponding malware description and accepted DFA state at which the MALICIOUS binary matched.

There are additional matched patterns represented in the following sections; Threat Indicators, Import Analysis and String Analysis, which aid in providing additional forensic evidence based on the uploaded executable file. Some of the highlighted evidence includes suspicious API calls, modification of the registry, malicious URL activity, a persistence mechanism and suspicious command-line strings, with the corresponding risk levels for each of the identified items.

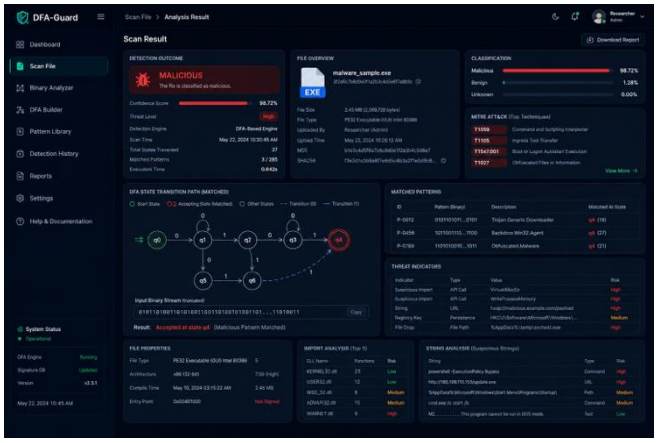

Figure 4. Malware analysis and Detection Result

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

V. Conclusion and Recommendation

The developed framework successfully analyzes uploaded executable files by converting binary sequences into streams of input that can be read by a DFA and evaluated against a set of known malicious pattern signatures. As a result of the DFA state transition process, the detection of malicious binary strings was achieved in a timely and resource-efficient manner while also providing a means by which to provide an explanation of the detection process. State traversal path visualizations in the DFA further contributed to the transparency and interpretability of detecting malware.

The successful detection of malicious binary strings (ending with pattern 101) using a DFA, implemented in the software package called JFLAP, was confirmed as a part of this study's objectives to demonstrate the ability to recreate the theoretical DFA designed, simulated, and tested using JFLAP. The simulations demonstrate that JFLAP effectively supports the theoretical automata by providing clear visualizations and reliable results during execution, through analyzing multiple runs and performing state transition analyses, as shown through the multiple testing cases run. Through this testing, there was a strong correlation between the theoretical models and the practical results obtained. The research supports advancement in formal methods of cybersecurity and future research in developing intelligent automaton, hybrid models of malware detection, and adaptive cyber defense systems.

References

[1] J. Smith and A. Lee, *Modern Perspectives on Finite Automata and Memory Optimization*, ResearchGate, 2021.
[2] R. Thompson, "State Minimization Algorithms in High-Speed Networking," *Journal of Theoretical Computer Science*, Elsevier, 2019.
[3] K. Miller, *Automata-Based Formal Verification for Software Security*, IEEE Xplore, 2023.
[4] L. Chen, "Scaling Regular Language Recognition for Data Streams," *International Conference on Formal Languages*, SpringerLink, 2022.
[5] K. K. Thota and R. Raj, “Minimal DFA With Optimization of Pattern Matching (MDFAOPM) for Network Traffic Analysis and Attacks,” *International Journal of Information Security and Privacy*, vol. 19, no. 1, pp.1–24, Jul. 2025, doi: https://doi.org/10.4018/ijisp.384918.
[6] P. Nawrocki, M. Grzywacz, and B. Sniezynski, “Adaptive resource planning for cloud-based services using machine learning,” *Journal of Parallel and Distributed Computing*, vol. 152, pp. 88–97, Jun. 2021, doi: https://doi.org/10.1016/j.jpdc.2021.02.018.
[7] M. Dotan, Y.-A. Pignolet, S. Schmid, S. Tochner, and A. Zohar, “Survey on Blockchain Networking: Context, State-of-the-Art, Challenges,” *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–34, Sep. 2021, doi: https://doi.org/10.1145/3453161.
[8] J. J. Castro-Schez, J. Gallardo, R. Miguel, and D. Vallejo, “Knowledge-based systems to enhance learning: A case study on formal languages and automata theory,” *Knowledge-Based Systems*, Elsevier, 2018.
[9] J. Ssanyu, “Teaching language processing with the PAMOJA framework,” *Computer Languages, Systems & Structures*, Elsevier, 2023.
[10] I. Suciu, “A digital learning tool based on models and simulators,” *Simulation Modelling Practice and Theory*, Elsevier, 2021.
[11] Baptista, I., Shiaeles, S., & Kolokotronis, N. (2019). *A novel malware detection system based on machine learning and binary visualization*. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICCW.2019.8757060
[12] Prithi, S., Sumathi, S., & Vinothkumar, P. (2020). *A novel learning dynamic deterministic finite automata for intrusion detection*. *Journal of Computational Science*, 39, 101064. https://doi.org/10.1016/j.jocs.2019.101064

[13] Dhavlle, A., & Shukla, S. (2021). *A novel malware detection mechanism based on features extracted from converted malware binary images*. In *2021 International Conference on Intelligent Technologies (CONIT)* (pp. 1–5). IEEE. https://doi.org/10.1109/CONIT51480.2021.9498463
[14] Ding, Y., Chen, X., Xu, C., & Zhao, J. (2022). *An efficient method for generating adversarial malware examples*. *Electronics*, 11(1), 154. https://doi.org/10.3390/electronics11010154
[15] Azeem, M., Ashfaq, R. A. R., Ahmed, M., & Shahid, M. (2023). *Malware detection and classification using machine learning techniques: A comprehensive review*. *IEEE Access*, 11, 52711–52742. https://doi.org/10.1109/ACCESS.2023.3279038
[16] Ramesh, G., & Menen, A. (2020). *Automated dynamic approach for detecting ransomware using finite-state machine*. *Future Generation Computer Systems*, 108, 1144–1157. https://doi.org/10.1016/j.future.2020.03.034
[17] Touili, T. (2022). *Register automata for malware specification*. In *International Symposium on Automated Technology for Verification and Analysis (ATVA 2022)*. Springer Lecture Notes in Computer Science (LNCS). https://doi.org/10.1007/978-3-031-19992-9_20
[18] Rey, V., Sánchez-Iborra, R., & Skarmeta, A. F. (2022). *Federated learning for malware detection in IoT devices*. *Computer Networks*, 204, 108693. https://doi.org/10.1016/j.comnet.2021.108693
[19] Cai, H. (2018). *A preliminary study on the sustainability of Android malware detection*. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 328–339). IEEE. https://doi.org/10.1109/QRS.2018.00046
[20] Shalaginov, A., Banin, S., Dehghantanha, A., & Franke, K. (2018). *Machine learning aided static malware analysis: A survey and tutorial*. In *Cyber Threat Intelligence*. Springer, Cham. https://doi.org/10.1007/978-3-319-73951-9_4