Jaydon Christen

CSC 212 Theory of Computing

Professor Hieu Bui

December 10, 2024

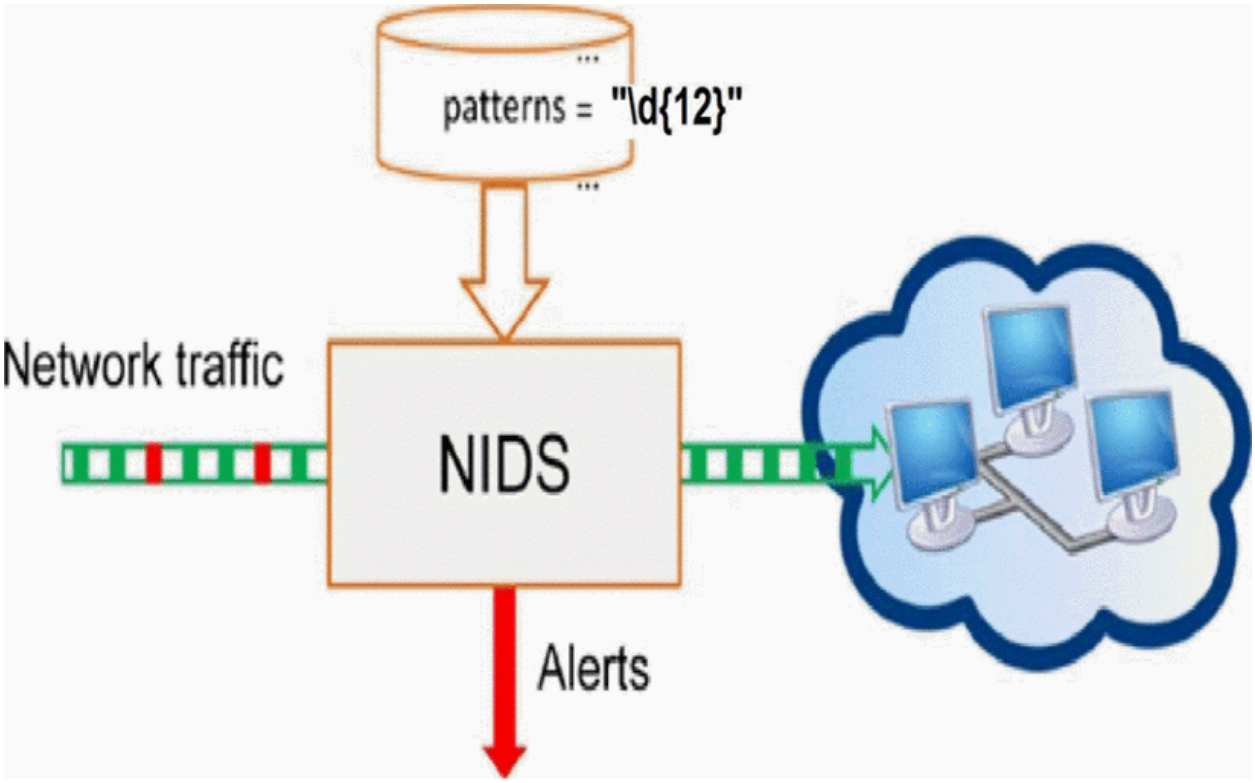
Maximizing IDS Efficiency Using Finite Automata

As technology advances, cybercrime becomes more sophisticated and advanced, making security an increasingly important issue. A key element of securing a system is the Intrusion Detection System (IDS), which detects the presence of malware and triggers a response. These detection systems are critical for system security. As malware becomes more advanced, so must cybersecurity. Finite automata is a mathematical machine which recognize patterns in input. This paper shows how finite automata is used to make IDS more efficient in order to adapt to more advanced and complex threats.

There are two main methods of intrusion detection: anomaly detection and penetration detection. Anomaly detection seeks to single out activity that deviates from normal activity. Penetration detection looks for known attack signatures in network traffic. The main tradeoff between the two methods is that penetration testing can quickly identify and combat known malware but is powerless against new malware that does not have an identified attack signature (Branch, 2003).

Intrusion detection systems work in two ways. The first way is network-based, analyzing packet data and identifying malware, such as Distributed Denial-of-Service (DDoS) attacks. Network-based intrusion detection systems (NIDS) are typically penetration identification systems (Branch, 2003). This is because most attacks that are delivered through a network, such as DDoS attacks have commonly known attack signatures, and the extra speed in detecting as

opposed to anomaly-based detection greatly increases efficiency (Scarfone, 2007). The below image is a simple model of a NIDS:



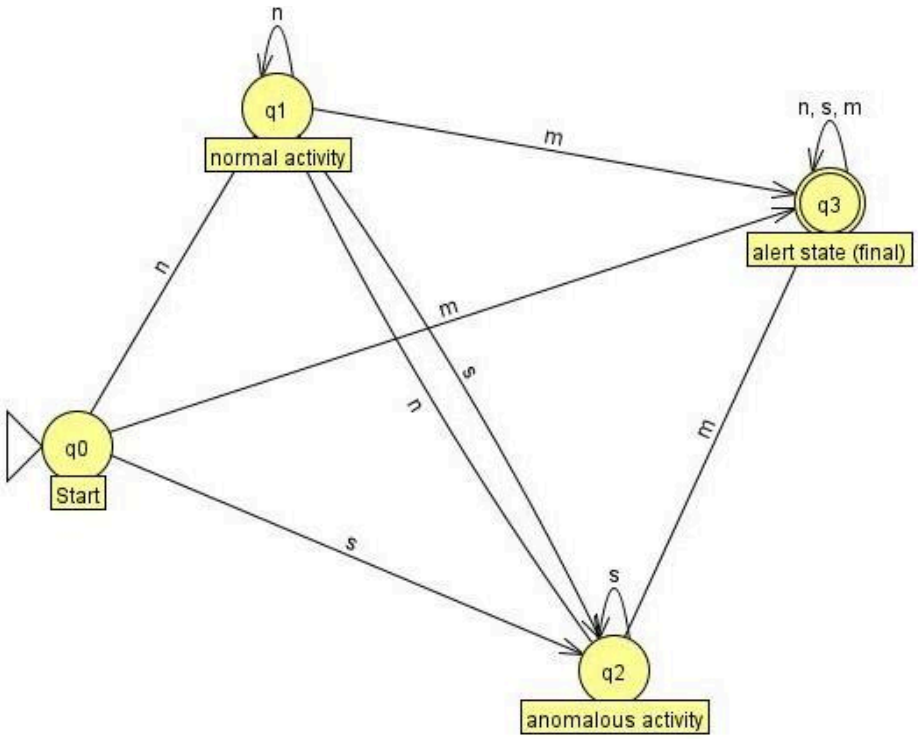
In this image, the NIDS monitors network traffic, searching for imputed search patterns of attack signatures (Marathe, 2014). When such patterns are identified, it sends an alert to a network admin, who investigates the threat and takes appropriate action to mitigate it. As network complexity and network-based attacks become more advanced, ways to keep up with these advances must be found. The second way IDS works is through host-based systems (HIDS), in which the activity of the host system is monitored. The data is analyzed and compared to a dataset of attack signatures, similar to how it is in NIDS, however, it has the advantage of being able to see attacks that do not move across the network (Branch).

Modern Intrusion Detection Systems are more and more adopting finite automata theory as a way to maximize their speed and efficiency. A finite automaton is a simple and powerful machine for computation (Havlena, 2021). They are very useful for pattern recognition and problem-solving. These computation machines contain a finite number of states that process input strings, which are in the form of regular expressions. A finite automaton is made up of a finite set of states, a finite set of input symbols, a transition function, a start state, and a set of final states. There are two main types of finite automata: deterministic finite automata (DFA) and nondeterministic finite automata (NFA). DFAs can only be in one state at any given time, and for every state, they must have exactly one transition for every input symbol. In contrast, NFAs can be in multiple states at once and can have any number of state transitions, including none. NFAs have the advantage of being easier to design as they have fewer states, but they have the disadvantage of requiring more transitions per state, resulting in higher time complexity. Though DFAs have more states, they are more efficient for pattern matching due to their deterministic nature. (Linz, 2011) We will go further into detail about this later in this paper.

Finite automata theory is an important and applicable concept to IDS as attack sequences are written in the form of regular expressions. Finite automata can provide solutions to past inefficiency and unoptimized processing speed in IDS. Finite automata have the advantages of increased efficiency in pattern matching and higher flexibility in complex pattern matching, as opposed to the more basic string matching algorithms used in the past such as the Knuth-Morris-Pratt Algorithm or Boyer-Moore Algorithm (Dharmapurikar, 2007). Modeling the system in the form of finite automata can increase the speed of pattern matching, however using automata on its own has its drawbacks.

To solve the problems of efficiency and flexibility in traditional algorithms, the advantages and disadvantages of both DFAs and NFAs must be examined to determine which to use. Efficiency aims for smaller linear time complexity, and DFA only has the time complexity of $O(n)$, while NFA has a higher complexity of $O(n \cdot 2^Q)$. This difference in efficiency occurs because DFA can only have one path, while NFA can traverse multiple paths at the same time. A DFA also has the advantage of being inherently unambiguous by its deterministic nature. Since NFA can have multiple transitions of the same symbol, this leads to ambiguity, in addition to the previously mentioned drawback of slower processing. In terms of flexibility, NFA is better in its ability to have multiple transitions at each state for the same input symbol, while DFA is confined by requiring exactly one transition per input symbol. NFAs are also much easier to construct for more complex patterns. Another aspect is memory requirement: DFA requires much more memory as they have many more states, especially when complex expressions lead to state explosion (Xu, 2014). Taking these advantages and drawbacks into consideration, DFA is a better choice when the goal is efficiency and memory is not a constraint. NFA has the upside of being more flexible, but that does not make up for the slower speed.

To construct an automata for an IDS, we need to make the different states, the alphabet, and a transition function. We want to be able to detect several different attack types, but we will start by creating a rudimentary example DFA. We will have a start state, a state for normal activity, a state for suspicious activity, and an alert state for confirmed malware. For our alphabet, we will have a symbol for when a normal packet is input (n) and a symbol for when a suspicious packet is detected (s). This could either be malware or an unexpected sequence. We will also have a symbol for confirmed malware (m). The following image shows this simplified DFA of an IDS:

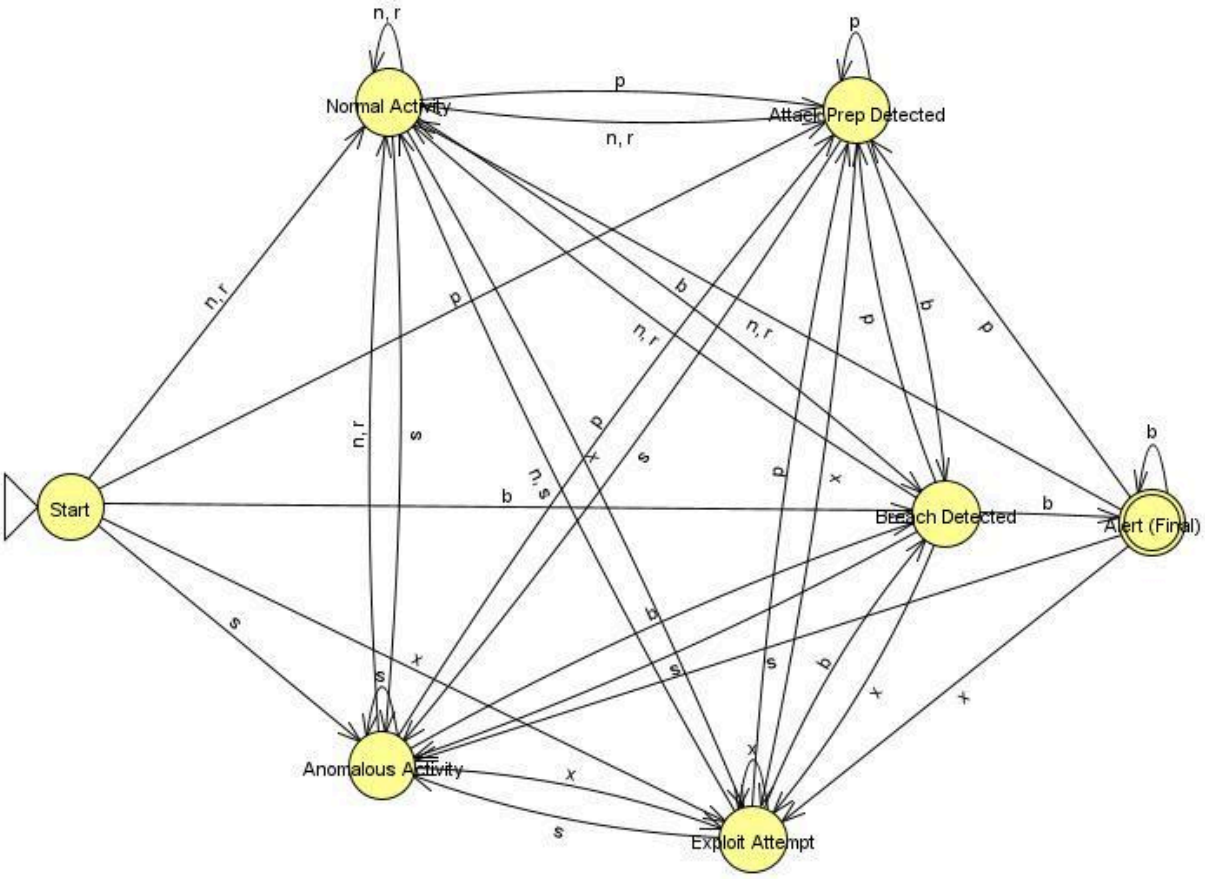


We can see in this simplified image how the IDS can use a DFA. As the IDS looks through network packets, the DFA continues to loop through normal activity and anomalous activity until malware is confirmed, triggering the final state.

We will now make a more complicated DFA for different types of attacks. We will build off of our previous DFA, but add states for a few different things that could be detected, and add more inputs as well. The first state we will add is an attack preparation state. This is for activity commonly seen before an attack is carried out, such as network scanning for exploits. We will add another state for if an exploit attempt is being attempted, such as a SQL injection. We will add another state for a data breach detected. For inputs, we will remove the malware symbol, and add corresponding symbols for the added states. This list of inputs includes an input for packets that resemble preparations for an attack (p), an input for a possible exploitation detected (x), and an input for a possible data breach (b). We will also add another state for attack mitigation (r).

This if for if a threat is detected, but is mitigated by an admin or Intrusion Prevention System (IPS), and the DFA returns to normal packet inspection.

The below immage shows this more advanced DFA (See Appendix I for more information).



This is an implementation of DFA to model an IDS. All possible inputs are accounted for, and the model will loop forever. Once it reaches the final state, it can be brought back to normal activity once the attack is mitigated (‘r’ input).

Though there are many upsides to replacing traditional algorithms with DFA, there are also downsides, making fully DFA-based pattern-matching inefficient. As the input sequence becomes more complex, the memory requirement to store all of the states becomes very large. This limits the ability of solely DFA-based pattern matching from being used on a large scale. The solution for this is use of hybrid algorithms of the two approaches (string matching and DFA). This hybrid approach is used on a large scale by the Snort IDS. Snort uses the Aho-Corasick algorithm, which generates DFAs for pattern matching. This algorithm is much more efficient because instead of processing each pattern individually, it puts all patterns together in preprocessing, greatly decreasing the number of operations that will be needed during pattern-matching (Roesch, 1999).

To conclude, finite autonama, particularly DFA, can be used to optimize the pattern-matching process in Intrusion Detection Systems. This paper showed how a DFA can be created to perform pattern matching, which works much more efficiently than other pattern matching approaches. Though DFA on its own has drawbacks in its memory overhead requirement, these limitations can be mostly overcome by using a hybrid algorithms solution, such as the Aho-Corasick algorithm. In the future, we can continue to work on making more efficient DFAs, and further adapt algorithms to use them in even more effective ways. I am very interested in this topic, as I plan on having my career in cybersecurity, and I will continue to research how this can be done. This project taught me a lot more about how useful finite autonama is in today's world, and all the problems it can help solve.

Appendix I: DFA design

States = {Start, Normal Activity , Anomalous Activity, Attack Preperation Detected, Exploit Attempt, Data Breach Detected, Alert State (Final)}

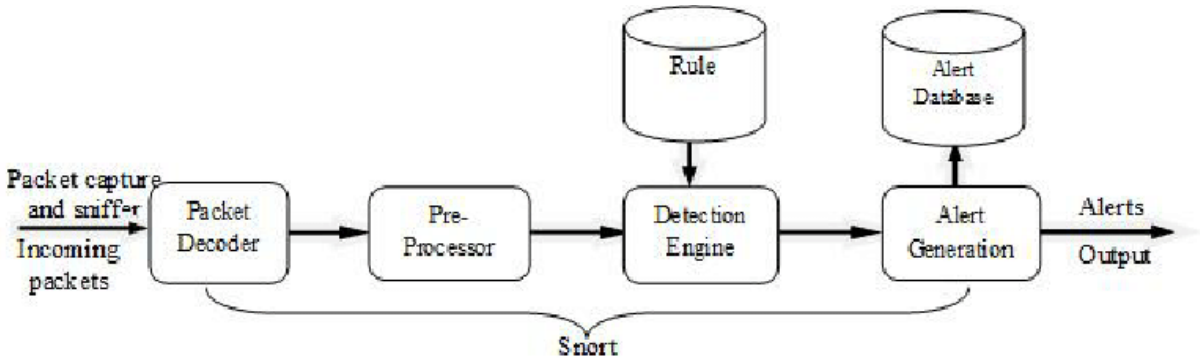
Alphabet = {n, s, p, x, b, r)

Transition Table:

Current State	Input Symbol	Next State
Start	n	Normal Activity
Start	s	Anomalous Activity
Start	p	Attack Prep Detected
Start	x	Exploit Attempt
Start	b	Data Breach Detected
Start	r	Normal Activity
Normal Activity	n	Normal Activity
Normal Activity	s	Anomalous Activity
Normal Activity	p	Attack Prep Detected
Normal Activity	x	Exploit Attempt
Normal Activity	b	Data Breach Detected
Normal Activity	r	Normal Activity
Anomalous Activity	n	Normal Activity
Anomalous Activity	s	Anomalous Activity
Anomalous Activity	p	Attack Prep Detected
Anomalous Activity	x	Exploit Attempt
Anomalous Activity	b	Data Breach Detected
Anomalous Activity	r	Normal Activity
Attack Prep Detected	n	Normal Activity
Attack Prep Detected	s	Anomalous Activity
Attack Prep Detected	p	Attack Prep Detected
Attack Prep Detected	x	Exploit Attempt
Attack Prep Detected	b	Data Breach Detected
Attack Prep Detected	r	Normal Activity

Current State	Input Symbol	Next State
Exploit Attempt	n	Normal Activity
Exploit Attempt	s	Anomalous Activity
Exploit Attempt	p	Attack Prep Detected
Exploit Attempt	x	Exploit Attempt
Exploit Attempt	b	Data Breach Detected
Exploit Attempt	r	Normal Activity
Data Breach Detected	n	Normal Activity
Data Breach Detected	s	Anomalous Activity
Data Breach Detected	p	Attack Prep Detected
Data Breach Detected	x	Exploit Attempt
Data Breach Detected	b	Alert
Data Breach Detected	r	Normal Activity
Alert State (Final)	n	monitoring
Alert State (Final)	s	Anomalous Activity
Alert State (Final)	p	Attack Prep Detected
Alert State (Final)	x	Exploit Attempt
Alert State (Final)	b	Alert State (Final)
Alert State (Final)	r	Normal Activity

Appendix II: Snort IDS model



References

1. Boumkheld, N., & El Koutbi, M. (2019). Intrusion detection system based on a deterministic finite automaton for smart grid systems. In K. Arai, S. Kapoor, & R. Bhatia (Eds.), *Advances in Information and Communication Networks* (pp. 203–211). Springer. https://doi.org/10.1007/978-3-030-03405-4_13
2. Branch, J. W. (2003). *Extended automata-based approaches to intrusion detection*. Rensselaer Polytechnic Institute. http://www.cs.rpi.edu/~szymansk/theses/branch.ms.03.pdf
3. Dharmapurikar, S., & Lockwood, J. (2007). Fast and Scalable Pattern Matching for Network Intrusion Detection Systems. *IEEE Journal on Selected Areas in Communications*, 24(10), 1781–1792.
4. Havlena, V. (2021). *Efficient automata techniques and their applications*. https://theses.cz/id/ma6r00/1058.pdf
5. Linz, P. (2011). *An Introduction to Formal Languages and Automata* (5th ed.). Jones & Bartlett Learning.
6. Marathe, N., & Rathod, P. M. (2014). A survey on finite automata-based pattern matching techniques for network intrusion detection systems (NIDS). *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. https://ieeexplore.ieee.org/abstract/document/7002456/
7. Roesch, M. (1999). *Snort: Lightweight Intrusion Detection for Networks*. Proceedings of the 13th USENIX Conference on System Administration. https://www.usenix.org/legacy/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf
8. Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)* (NIST Special Publication 800-94). National Institute of Standards and Technology. Retrieved from https://www.nist.gov/publications/guide-intrusion-detection-and-prevention-systems-idps
9. Xu, Y., Jiang, J., Wei, R., Song, Y., & Chao, H. J. (2014). TFA: A Tunable Finite Automaton for Pattern Matching in Network Intrusion Detection Systems. *IEEE Journal on Selected Areas in Communications*, 32(10), 1910–1923. https://doi.org/10.1109/JSAC.2014.141010