Anomaly Detection in ICS Networks with Fuzzy Hashing

William Tatum
Computer Science & Engineering
Texas A&M University
College Station, USA
willtaaa@tamu.edu

Noam Gariani
Computer Science & Engineering
Texas A&M University
College Station, USA
noamgariani@tamu.edu

Keith Alan Crabb
Computer Science & Engineering
Texas A&M University
College Station, USA
lampsdad@tamu.edu

Gabriel De Conto
Computer Science & Engineering
Texas A&M University
College Station, USA
gabriel.deconto@tamu.edu

John A. Hamilton, Jr., Ph.D.
Computer Science & Engineering
Texas A&M University
College Station, USA
hamilton@tamu.edu

***Abstract*—The recent increase in attacks against publicly networked industrial control systems (ICS) has demonstrated a need for network-based anomaly detection systems, offering real-time flagging of potentially malicious activity by internal and external threat actors. Fuzzy hashing, also known as similarity hashing, has gained popularity in malware analysis and digital forensics circles as it provides analysts functionality to determine the similarity of two pieces of data by providing a similarity score. This work proposes a scheme that utilizes the similarity score to find variations from a self-establishing baseline in an ICS network to identify anomalous network traffic sections that could signify malicious activity.**

***Index Terms*—ICS, SCADA, Fuzzy Hashing, Anomaly Detection, Operational Technology**

I. Introduction

Recently, there has been an increase in cyber attacks against networked industrial control systems, negatively impacting security for many critical infrastructure sectors. We propose an anomaly detection scheme that utilizes the TLSH fuzzy hashing algorithm in response to this increased threat.

As the most commonly used networking protocol for operational technology (OT), our system takes advantage of the MODBUS packet structure and the loosely deterministic conversations between programmable logic controllers (PLCs) to establish an observed baseline over time. Despite the advantages, fuzzy hashing with MODBUS proved to be challenging in many ways. The most significant challenge is that there are limited amounts of useful information in a single MODBUS packet, with a total packet size typically being in the range of 43.5 to 255 bytes. MODBUS packets contain only an address, function code, and some data.

Fuzzy hashing algorithms can be wildly inaccurate at this small scale. To overcome this issue of scale, our proposed system uses clusters of packets at a variable size N.

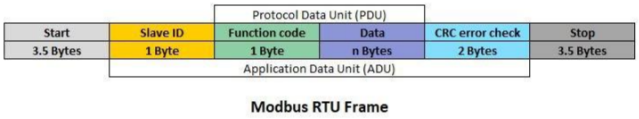

Fig. 1. Figure 1: Modbus RTU Frame Diagram.

A. Datasets Utilized

The datasets used in this paper were the CIC MODBUS dataset 2023 dataset [1] and the ICS_PCAPS dataset on GitHub [2]. This dataset had packet capture files with known attacks and times.

II. Literature Review

The concept of cryptographic hashes is fundamental in ensuring file integrity and uniqueness. Even a minor alteration in the file results in a significantly different hash, effectively preventing overlap between different files. In contrast, similarity hashes, or fuzzy hashes, are designed to produce only slight changes in the hash when a file is slightly altered. This property allows fuzzy hashes to identify similar files. Prominent fuzzy hashing methods today include TLSH and NILSIMSA, which both employ locality-sensitive hashing (LSH) techniques.

The foundational paper on TLSH by Oliver, Cheng, and Chen provides an in-depth analysis of TLSH, comparing it with NILSIMSA, SSDEEP, and SDHASH and discussing the differences between traditional cryptographic hashes like MD5 and SHA-1 versus similarity hashes [3]. Beyond TLSH, other notable fuzzy hashes include SSDEEP, SDHASH, and Nilsimsa. SSDEEP computes context-triggered piecewise hashes (CTPH) by segmenting the file and evaluating a 6-bit hash value for each segment. Early implementations of SSDEEP yielded varying results as it was one of the first fuzzy hashing algorithms. However, it has been refined for specific purposes

to improve its results. SDHASH identifies key features of a file and hashes them using a bloom filter, with similarity scores calculated via a normalized entropy measure. Nilsimsa utilizes bit sampling LSH and computes similarity scores based on the Hamming distance between hashes. TLSH incorporates concepts from Nilsimsa, such as a sliding window of 5 bytes and extracting triplets, but enhances accuracy and speed.

A significant advantage of TLSH is its robustness against adversarial attacks [4]. Through its sliding window and kskip-ngrams, TLSH considers sequences of bytes in adjacent chunks with "skips" between them, offering a defense against algorithmic file alterations intended to preserve high similarity scores. This feature makes TLSH less susceptible to noisy symbols, lowering false positive rates. Kskip-ngrams allow TLSH to ignore specific noisy symbols, such as checksums, reducing their impact on the overall similarity score compared to algorithms using only ngrams.

Third-party validators highlight the benefits of TLSH over other fuzzy hashing algorithms. The paper" Designing the Elements of a Fuzzy Hashing Scheme" explores criteria for fuzzy hashing schemes, focusing primarily on TLSH [4]. Evaluating theoretical configurations based on NIST criteria: accuracy, compression, performance, and security, the study found that skip-n-grams provide greater security than ngrams despite similar accuracy. An optimal configuration was determined to be K=2, N=5 for the best balance of security, accuracy, and performance. However, higher values for K and N can increase accuracy at the cost of performance. TLSH was efficient, fast, and effective with various file types, including packet capture files (Pcap). Comparatively, LSH methods like Nilsimsa, despite their similar methodologies, are more time-consuming.

The paper" Forensic Malware Analysis: The Value of Fuzzy Hashing Algorithms in Identifying Similarities" [5] provides a comprehensive evaluation of multiple hashing methods for malware analysis. It considered various algorithms but found that the SPH algorithm was most applicable to their malware research. They concluded that SPH should not be used alone but as an additional check measure. The paper "Nation-State Threat Actor Attribution Using Fuzzy Hashing" [6] addresses the challenge of attributing cyberattacks to nation-state actors. The proposed framework similarly found that combining fuzzy hashing with another method provided the best results. In this case, it was machine learning to classify malware, using techniques like phonetic alphabets to identify patterns in malicious files. The paper also compares the efficiency of various fuzzy hashing algorithms for binary analysis. This methodology provides insights into using fuzzy hashing for rapid threat attribution and analysis, which could be applied to packet inspection in our project. [6].

YARA rules are a well-known detection method, and in 2019, Nitin Naik published papers assessing YARA rules, Import Hashing, and various fuzzy hashing methods for malware detection. His paper" Triaging Ransomware using Fuzzy Hashing, Import Hashing, and YARA Rules" found that YARA rules outperformed other methods [7]. Fuzzy hashing was criticized for providing overly vague results in malware detection, prompting an independent group to develop a new fuzzy hashing algorithm addressing these shortcomings [8]. Although YARA improved malware analysis over SSDEEP and SDHASH, it gained little popularity, with newer algorithms like NILSIMSA becoming more popular than YARA.

In his subsequent paper," Fuzzy Hashing Aided Enhanced YARA Rules for Malware Triaging," Naik combined various algorithms to see if they collectively outperformed YARA rules alone [9]. The combination yielded superior results, reaffirming the effectiveness of fuzzy hashing when integrated with YARA rules. This promising approach mirrors machine learning techniques and suggests combining methodologies can enhance malware detection.

The paper" A Stealthy False Command Injection Attack on MODBUS-based SCADA Systems" [10] explores vulnerabilities in the Modbus protocol, widely used in ICS SCADA communications for controlling remote devices like PLCs and monitoring physical systems. Given Modbus's lack of modern security features, the paper outlines methods and scenarios for stealthily injecting packets into the system while concealing the attack from the operator. The proposed attack consists of a scouting phase, where the network is scanned, and data is collected, followed by an attack phase involving methods such as ARP poisoning and command injection. Modbus is prone to many attacks, such as spoofing, replay, response injection, command injection, flooding attacks, and more. During the scouting phase, the attacker uses tools like Nmap to scan the network and Wireshark to sniff network traffic. In the attack phase, methods such as ARP poisoning facilitate man-in-the-middle (MITM) attacks, while command injection involves sending forged MODBUS request frames impersonating the HMI. This research is particularly relevant as it highlights potential avenues of attack that could be targeted using fuzzy hashing for detection and mitigation.

In summary, the evolution and application of fuzzy hashing methods like TLSH, SSDEEP, SDHASH, and Nilsimsa demonstrate their crucial role in file similarity detection and malware analysis. The advancements in TLSH, particularly its robustness against adversarial attacks and efficient performance, highlight its superiority in various scenarios. Integrating fuzzy hashing with other detection techniques, such as YARA rules, can further enhance the effectiveness of malware detection and file analysis. Furthermore, exploring the application of fuzzy hashing in different contexts, such as SCADA systems, malware analysis, and firmware security, underscores the versatility and potential of these techniques in improving cybersecurity measures.

III. Hashing Algorithms

Fuzzy hashing or locality-sensitive hashing (LSH) is a hashing algorithm where the data is divided into chunks and then combined to form an overall hash where the hashes between two equally sized data segments will have a similar hash if they are similar. We conducted a preliminary study using a variety of fuzzy hashing algorithms, such as TLSH,

Nilsimsa, and others that were not suited for the task, such as SSDEEP. Fuzzy hashing was chosen due to its applicability in malware analysis. Since it can detect structures that are broadly similar but not quite the same, this might work on network traffic.

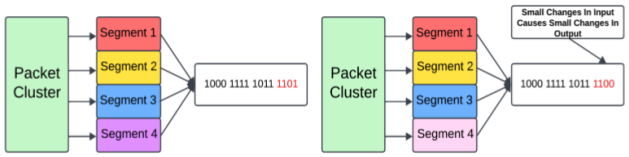

Fig. 2. Figure 2: LSH function diagram.

A. TLSH

TLSH is a fast, relatively accurate, local sensitivity hash. According to "TLSH - A Locality Sensitive Hash" [7], TLSH outperformed Nilsimsa, SDHASH, and SSDEEP in both the false positive and true positive metrics.

This algorithm works on a scale of 0 to infinity. The file has the same if it is a score of 0, and anything above zero indicates the files are somewhat different. Here are the procured similarities based on testing:

- Very similar: 0 to <50
- Similar: 50 to <70
- Somewhat similar: 70 to < 90
- Somewhat not similar: 90 to < 128
- Not similar: 128 to ∞

This algorithm typically has a fast run time to compile packet capture data. However, with massive amounts of data it can take a long time and benefits from multi-threading.

B. NILSIMSA

Nilsimsa is a hashing algorithm designed to identify spam emails. It is a locality-sensitive hashing algorithm (LSH) where changes to the input of the algorithm result in small changes in the output. The goal of the technique was to create hash digests of the messages and then compare the digests together. While effective at identifying similarities in spam emails, Nilsimsa is vulnerable to several different techniques that prevent identification. These techniques include random addition, thesaurus substitution, perceptive substitution, and aimed addition [11]. When processing files that are not email-based, Nilsimsa is excellent at comparing similar files but also results in many false positives [3].

This algorithm works on a scale of -128 to 128. The file is the same if it has a score of 128; and anything below zero means the files are very different. Here are the procured similarities based on testing:

- Very similar: 90 to < 128
- Similar: 80 to < 90
- Somewhat similar: 70 to < 80
- Somewhat not similar: 50 to < 70
- Not similar: 0 to < 50

This algorithm takes a very long time to compile packet capture data. It was found that compiling around 10 MB of data would take 1 minute in time with the program that was not multi-threaded.

C. SSDEEP

This algorithm uses context-piecewise triggered hashing. It was one of the first fuzzy hashing algorithms to emerge and was very effective and fast for the specified purposes. One of the purposes was to analyze emails for similarity purposes.

When applied to packet captures in most of the considered approaches, SSDEEP provided exclusively false positives. It works on a scale of 0 to 100; most the time, it would return 0 or near 0 results. Once it was shown to be ineffective, it was not further pursued. However, it was a good algorithm for consideration as it was the first one that was investigated.

D. Chosen Algorithm

We chose TLSH because it is fast, accurate, and a local sensitivity hash, which worked best for anomaly detection. TLSH also has a much higher robustness to attack, meaning that it is more difficult to detect and evade than other fuzzy hashing algorithms, making it well suited for the task.

IV. Approaches

Throughout the research, four main approaches were attempted, and several data formats were tested, each with a different level of accuracy.

Average percentage similarity between two similar packets' hashes over multiple runs:

- String: 91.33%
- Binary: 74.97%
- Hex: 89.13%
- Lexical: 99.17%

Average percentage similarity between two different packets' hashes over multiple runs:

- String: 4.80%
- Binary: 9.83%
- Hex: 31.03%
- Lexical: 95.43%

The string and hex representations are the most accurate for similar packets, and the string and binary representations are the most accurate for dissimilar packets. Generally, the string representation tends to be the best one. Lexical or just sorting all of the data by ASCII value is utterly useless as it guarantees that it will be similar either way due to how it sorts the data.

A. Signature analysis

The first idea was to use a classic signature analysis with these hashes. This approach was based upon existing techniques like antivirus programs. If there were some known malicious hashes, then the hashes taken against a network capture could be tested against these malicious hashes. This

comparison of hashes would make it relatively simple to prevent known attacks from being recreated in the system.

Results: Some issues found with this one were the small size of the packets and how similar each one was due to that. Since these are very small packets containing often very similar information over long periods of time, it is difficult to analyze packets or groups of packets against known malicious packets or groupings of packets.

B. Single Packet

Single packet analysis ended up being the worst approach since MODBUS packets contain very little useful information to work with. This approach was inspired by deep packet inspection and determining what useful information could be gathered from a MODBUS packet. One of the big problems was that MODBUS packets are small so the data could match with far more than just a malicious packet and would throw many false positives. This one aimed to see if traditional deep packet inspection techniques would work on a machine blocking access to a secure network.

Results: Unfortunately, this requires a minimum of 50 bytes of data. This is a big challenge with MODBUS packets since they tend to have very little data, so this was not a good approach.

C. Whole file

By hashing the entire file with all the data, a representation of the data in a hash format can be retrieved. This hash represents the packet capture and, when compared, was hoped to identify the attack type based on other attack packet captures. This approach was based on malware analysis techniques with fuzzy hashing. Instead of relying on a static cryptography hash that would completely change with the slightest change, fuzzy hashing would stay pretty similar regardless of the change. The thought was to apply this to network traffic analysis to find a specific attack or anomaly in the network based on other files from a dataset.

Results: This method was very good at detecting if the packet capture was taken on the same network but could not find the intricate details of a network attack, which caused a lot of false positives initially. After trimming the packet capture data, like time stamps, there were fewer false positives, but it still was not able to reliably detect specific attacks.

D. Clustering

The cluster method worked the best out of all of them. This approach was based on the sliding window technique. The cluster method would be useful if there was some time period containing an attack similar to a known attack period. This result was also similar to the signature analysis, but instead of a single packet inspection, it involved clusters of packets. The cluster method also fixed the issue with the size of the packets as now the goal was to look at whole clusters of them. This method could find attacks that cause the network traffic to deviate from the baseline. Depending on the size of the cluster, it can also be a way to compress network traffic into hashes and reduce the storage impact.

Results: This gave the best results, as a baseline will form after a while. This baseline allows for analyzing the data within the window against the known normal baseline. The cluster method allowed for detecting changes against that normal.

E. Chosen Method

We chose clustering because it could also provide a rolling baseline. Clustering would resist minor changes in the network traffic, such as time and sequence numbers while being susceptible to large changes to the traffic in a short time period. Additionally, the rolling baseline could define a network in terms of this system, making it easy to determine normal from abnormal traffic since these systems are largely deterministic. Determinism was a big part of this choice, especially since it allowed for easier analysis since these systems would likely be repeating similar commands over and over again, and any interruptions to that would provide clues.

V. Detecting Anomalous Traffic

Our proposed system for detecting anomalous traffic relies on creating and updating a list of fuzzy hashes. By taking packet clusters and splitting the different conversations seen between PLCs, we were able to establish a rolling baseline of hashes that show what network traffic should look like. This system would require a network capturing device or software to allow us to create the packet clusters. By funneling these hashes into the TLSH algorithm, we receive an output of hashes broken into different files for resulting conversations. One of the challenges with anomaly detection in ICS systems is the MODBUS packets (fig. 1) which are very sparse. With very little data and information based on just register changes it could be difficult to perform behavior analysis and see such a difference. To address this problem, we clustered packets together to gain a meaningful block of data of sufficient size.

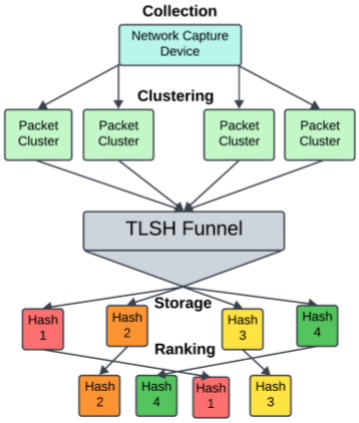

Fig. 3. Process of detecting anomalous traffic.

Using the native functionality in TLSH to create similarity scores, we can then create a sliding window where similarity scores for these hashes can be compared against each other. By comparing similarity scores within the given window, we are given an output showing the packet clusters that are least similar to the rest of the traffic in the given sliding window.

VI. Future Work

With our proposed system, there is future work required to ensure it's dynamically responsive to a wide variety of systems. With the wide number of communication protocols designed for ICS Networks, our system should be tested to ensure compatibility. Future analysis methods should be explored to match what is commonly seen in production ICS networks, such as including system set points to verify the integrity of network data. Including lexical analysis procedures through text replacement, similar to methods seen in intrusion detection systems like snort before the packet clusters are hashed may also help decrease the number of false positives within the system.

This system would also likely perform most optimally as a primary indicator with additional validation built on top. Several other mechanisms, such as malware analysis [5] have shown more effective results utilizing fuzzy hashing alongside other validation techniques. It has also been noted that fuzzy hashing can work well with machine learning to attribute attacks between nation-state threat actors [6] and given the similarity score-focused design, it is highly likely that machine learning would work well for automatic analysis in this environment.

Acknowledgment

The authors would like to gratefully acknowledge all support provided by the Texas A&M Cybersecurity Center. Specifically, the Texas SCADA Testbed, which lent the equipment and research space necessary to complete this paper.

References

[1] K. Boakye-Boateng, A. Ghorbani, A. Lashkari, "Securing Substations with Trust, Risk Posture, and Multi-Agent Systems: A Comprehensive Approach," 2023 20th Annual International Conference on Privacy, Security and Trust (PST), Copenhagen, Denmark, 2023, pp. 1-12, doi: 10.1109/PST58708.2023.10320154.

[2] I. Fraza˜o, P. Abreu, T. Cruz, H. Arau´jo, P. Simo˜es, "Denial of Service Attacks: Detecting the Frailties of Machine Learning Algorithms in the Classification Process," 13th International Conference on Critical Information Infrastructures Security (CRITIS 2018), ed. Springer, Kaunas, Lithuania, September 24-26, 2018, Springer series on Security and Cryptology, 2018. DOI: 10.1007/978-3-030-05849-4 19.

[3] J. Oliver, C. Cheng and Y. Chen, "TLSH – A Locality Sensitive Hash," 2013 Fourth Cybercrime and Trustworthy Computing Workshop, Sydney, NSW, Australia, 2013, pp. 7-13, doi: 10.1109/CTC.2013.9.

[4] J. Oliver and J. Hagen, "Designing the Elements of a Fuzzy Hashing Scheme," 2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC), Shenyang, China, 2021, pp. 1-6, doi: 10.1109/EUC53437.2021.00028.

[5] N. Sarantinos, C. Benzaïd, O. Arabiat and A. Al-Nemrat. (2016). "Forensic Malware Analysis: The Value of Fuzzy Hashing Algorithms in Identifying Similarities." IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 2016, pp. 1782-1787, doi: 10.1109/TrustCom.2016.0274.

[6] M. Kida and O. Olukoya, "Nation-State Threat Actor Attribution Using Fuzzy Hashing," IEEE Access, vol. 11, pp. 1148–1165, 2023, doi: https://doi.org/10.1109/access.2022.3233403.

[7] Naik, P. Jenkins and N. Savage, "A Ransomware Detection Method Using Fuzzy Hashing for Mitigating the Risk of Occlusion of Information Systems," 2019 International Symposium on Systems Engineering (ISSE), Edinburgh, UK, 2019, pp. 1-6, doi: 10.1109/ISSE46696.2019.8984540, accessed 14 August 2024.

[8] Y. Li, S. Sundaramurthy, A. Bardas, X. Ou, D. Caragea, X. Hu, & J. Jang, "Experimental Study of Fuzzy Hashing in Malware Clustering Analysis" CSET 15, 8th Workshop on Cyber Security Experimentation and Test, USENIX, Washington, DC, 10 August 2015. https://www.usenix.org/system/files/conference/cset15/cset15-li.pdf, accessed 8 August 2024.

[9] N. Naik, P. Jenkins, N. Savage, L. Young, K. Naik, J. Song, T. Boongoen, & N. Iam-On, N. "Fuzzy Hashing Aided Enhanced YARA Rules for Malware Triaging," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 2020, pp. 1138-1145, doi: 10.1109/SSCI47803.2020.9308189.

[10] W. Alsabbagh, S. Amogbonjaye, D. Urrego and P. Langendörfer, "A Stealthy False Command Injection Attack on Modbus based SCADA Systems," 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-9, doi: 10.1109/CCNC51644.2023.10059804.

[11] E. Damiani, S. Vimercati, S. Paraboschi, & S. Pierangela, 2004. "An Open Digest-based Technique for Spam Detection." The Security, Privacy, and Data Protection Laboratory (SPDP). http://spdp.di.unimi.it/papers/pdcs04.pdf , (Accessed 14 August 2024).