Application of Approximate Matching on Industrial Control System (ICS) Network Communication Using Ssdeep Algorithm

Student: **Nelson Makau Mutua**

Strathmore University, Nairobi, Kenya, Email: nmutua@strathmore.edu

Supervisor: **Ing. Petr Matousek, PhD., MA**

Brno University of Technology, Brno, Czech Republic, Email: matousp@fit.vutbr.cz

Erasmus Short Term Study Report

ABSTRACT

Network communication is associated with many security challenges. Changes in Internet technologies have allowed for an increase in networked devices, the complexity of cybercrimes and the transfer of huge amounts of data, which can easily be intercepted and manipulated by attackers. The goal of this research is to prove the viability of using approximate pattern matching to profiling Industrial Control System (ICS) communication. The approximate pattern matching has been successfully used on comparing similarity of files in the past. Tshark is a network protocol analyser that will be used to extract interesting fields of an IEC 60870-5 protocol (aka IEC 104) from the ICS communication packet capture files.

IEC 104 is a protocol that provides a communication profile for sending basic telecontrol messages between two systems in electrical engineering and power system automation. This protocol enables communication between control station and a substation via a standard TCP/IP network. The communication is based on the client-server model. An ICS normal profile is computed from the packet capture files to represent a normal ICS traffic. In the anomaly detection phase, unknown ICS network traffic is compared to the normal profile using approximate pattern matching algorithm. In this research, Ssdeep pattern matching algorithm will be used to compute the matching score between profiles to identify anomalies.

Keywords: *IEC 104, Industrial Control System, Ssdeep, Anomaly Detection, Network Traffic*

Table of Contents

ABSTRACT 1
CHAPTER 1: INTRODUCTION 4
1.1 Study Background 4
CHAPTER 2: LITERATURE REVIEW 5
2.1 IEC 60870 Communication 5
2.2 IEC 60870-5 - 104 Protocol 5
2.3 IEC 104 Vulnerabilities 6
2.3.1 Plaintext Mode Message Transmission 6
2.3.2 Lack of Authentication Mechanism 6
2.4 Approximate Pattern Matching Algorithm 6
2.4.1 Block-Based Hashing (BBH) 7
2.4.2 Context-Triggered Piecewise Hashing (CTPH) 7
2.4.3 Statistically Improbable Features (SFI) 7
2.4.4 Block-Based Rebuilding (BBR) 8
2.4. Ssdeep 8
2.4.1 Levenshtein Distance 9
2.5. Ssdeep Test Cases on Files 10
2.6 Conclusion 11
CHAPTER 3: DATA SET 12
CHAPTER 4: SYSTEM DESIGN AND IMPLIMENTATION 13
4.1 System Design 15
4.2 Use Case Diagram 16
4.2.1 Use Case 1: Create-Profile.py Script 17
4.2.2 Use Case 2: Data Extraction 17
4.2.3 Use Case 3 Profile Hashing 18
4.2.4 Use Case 4 Pattern Matching 19
4.3 Sequence Diagram 20
4.3.1 Load and Extract PCAP File 20
4.3.2 Hash Payload File 21
4.3.3 Compare Profiles 21
CHAPTER 5: RESULTS 22
5.1 Overview 22
5.2 Ssdeep Set-up and Installation 22

5.3 TCP Payload Extraction 22
5.4 TCP Payload Hashing 23
5.5 Pattern Matching 24
5.6 Man-In-The-Middle Attack Simulation 26
5.7 System Validation 27
5.8 Conclusion 28
5.9 Future Work 29
REFERENCES 30

CHAPTER 1: INTRODUCTION

1.1 Study Background

Industrial Control Systems (ICS) incorporates numerous kinds of systems, among them, engineering workstations, programmable logic controllers (PLC), and supervisory control and data acquisition (SCADA). Generally, ICS are utilized at the core of national critical infrastructure and usually exposed to public networks that are at a high risk of cyber-attack. The control procedures of critical infrastructures such as power plant, oil and gas facilities, chemical processing plants, traffic control systems, among others are becoming more and more vulnerable to external network threats (Leith & Piper, 2013).

Computer security has moved past ordinary office systems, and attackers are currently likewise focusing on Industrial Control Systems (ICSs) (Bolzoni et al., 2012). The key reason as to why organisation networks are targets for hackers is to obtain confidential information. The contemporary network system is voluminous to such an extent that manual inspection for malware is illogical and a costly task for an organisation to perform. In recent history, the most critical attack against an ICS is the Stuxnet attack (Langner, 2013), which is known as the first digital/cyber fighting weapon. The attack was meant to physically sabotage the Iranian nuclear program. In response to such kind of attacks, the U.S government in 2007 demonstrated how attackers could possibly destroy a power plant with only 21 lines of code (Zetter, 2016). In addition, a similar attack was carried out in late 2015 and early 2016, whereby hackers sabotaged two power distribution companies in Ukraine with an aim of shutting down electricity to more than 80, 000 people in the country. Another case was witnessed in Germany in 2015 where attackers compromised a steel mill and tampered with a blast furnace, leading to massive destruction on the plant (Zetter, 2015).

The motivation/goal of the project is to prove the viability of using approximate pattern matching algorithm to profiling Industrial Control System communication. I aim to study the pattern matching algorithms, read recommended papers and experiment with ICS communication packets. Specific objects are: -

1. To find out how to profile ICS communication using approximate matching algorithm.
2. To find out which approximate matching algorithm can be used to profile ICS network communication.
3. To design and develop an ICS network profiling tool using approximate matching algorithm.
4. To test and validate the effectiveness in terms of accuracy of the solution.

CHAPTER 2: LITERATURE REVIEW

2.1 IEC 60870 Communication

IEC 104 messages are exchanged between the controlled and the controlling station. The controlled station is also called Remote Terminal Unit (RTU) slave. A master station commands the slave station. The controlling station (master station) performs controls of outstation. IEC 104 communication is delivered in the monitoring direction i.e. from the controlled station (slave) to the controlling station (master) or in the control direction i.e. controlling station (master) to the controlled station (slave). Figure below shows a topology of IEC 104 router connected with 104 SCADA monitoring systems using IEC104 protocol over TCP/IP and IEC101 sensors communicating via Modbus RTU with the router (Matoušek, 2017).

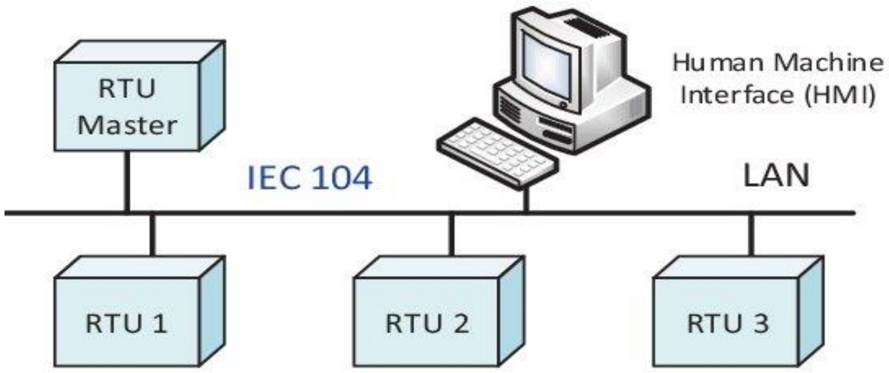


Figure 1: IEC 104 topology

2.2 IEC 60870-5 - 104 Protocol

IEC 60870-5-104 is also commonly referred to as IEC 104. This SCADA protocol is defined within a collection of standards called IEC 60870. This protocol is a TCP/IP adaptation of the long-standing IEC 101 serial protocol, also known as IEC60870-5-101. IEC 101 defines the remote-control functionalities needed in extensive areas. Within the electrical industry, both IEC 104 and IEC 101 are used extensively to help in establishing communications links that connect the electrical control stations to the substations (Haverkort & Remke, 2018).

The application layer of the IEC 104 consists of sub-layers. They are the Application Protocol Control Information (APCI) and Application Service Data Unit (ASDU) sub-layers. Above the TCP layer lies a sub-layer, the APCI, which defines three 104 message types. The formats consist: the (I-format) short for information transfer format, (S-format) known as the numbered supervisory functions and finally the (U-format) unnumbered control functions. U-format type of messages are used for initiating, halting and checking the connection statuses. On the other hand, the I-format type of messages transfer data two or more IEC 104 devices. Finally, S-format messages are crucial in acknowledging I-format messages previously received (Isakov et. al., 2014).

2.3 IEC 104 Vulnerabilities

The existing cyber vulnerabilities and attacks found within the physical and application layer of IEC 104 protocol are as follows;

2.3.1 Plaintext Mode Message Transmission

Due to data transmission in the form of clear text in IEC 104, information transfer between substations and the control centre is conceivably prone to high level cyber-attacks such as information alteration, tapping and espionage. For instance, a cyber-attacker can unveil a Man in-the-Middle (MITM) attack to help in sniffing as well as gathering remote signals, remote measurement values and remote-control commands (Papakonstantinou, 2015). For every case packet can be intercepted, modified and re-injected to the communications channel in order to compromise the security and stability of the SCADA system. These malicious attacks expose the SCADA to future attacks.

2.3.2 Lack of Authentication Mechanism

Due to lack of authentication mechanism in ICS, malicious attackers can gain unauthorized access and compromise information integrity and availability. The attackers can also launch different type of attacks such as; MITM attacks, spoofing attacks or replay attacks. This is a huge system security loophole because lack of authentication provides hackers with a chance to gain easy access and this pauses a serious system vulnerability, which can lead to system compromise thus affecting operations, safety and may result into catastrophic damage (Case, 2016). For example, a false remote-control command such as “open the circuit breaker” could lead the power system to shed load affecting power supply reliability and at the same time exposing the operators and the large population to safety concerns.

2.4 Approximate Pattern Matching Algorithm

In evaluation to cryptographic hashing functions, approximate pattern matching algorithm, otherwise referred to as fuzzy hashing algorithm, attempts to detect the similarity between two or more files by connecting similar inputs to similar outputs, indistinctly known as a fingerprints or profile. Approximate matching algorithms analyse files at byte level making them suitable to compare a large chuck of data and identify similar texts or embedded objects (e.g., an image in a file) or binary fragments (e.g., a virus contained inside a file or a specific data packet in a network communication).

By utilizing a variety of strategies, approximate pattern matching algorithms are able to identify if even a single byte has been changed (NIST, 2014). A confidence score is determined upon successful similarity matching. The confidence score is normally determined by the number of attributes in common between the objects in correlation. The score is anticipated to be high when similarity between the shared content is high and low when less content in the two files is similar. Fuzzy hashing can be categorized into the following four categories: -

2.4.1 Block-Based Hashing (BBH)

Block-based hashing technique produce and store cryptographic hashes for given block of number with fixed byte sizes such as 512 bytes. In more subsequent stages of block-based hashing, it compares two different inputs by evaluating and establishing the number of common blocks on these inputs and providing the measure of the degree of similarity between them. This fuzzy function was designed by Nicholas Harbour under a program that he termed as dcfldd (Harbour, 2002). This fuzzy function splits the input data into several blocks of a fixed length and then computes the corresponding cryptographic hash value for each block. Despite the fact that the hash-based approach is efficient, more reliable and exceptionally proficient. A single byte change at the beginning of a file can change all the other block hashes, making this scheme very vulnerable.

2.4.2 Context-Triggered Piecewise Hashing (CTPH)

This method was developed by Andrew Tridgell, who actualized spamsum, a CTPH application focused on the identification of spam e-mails (Tridgell, 1999). The primary idea behind this concept was to help in locating content markers, known as contexts, within a binary data. This method calculates the hash function of each segment of the file delimited by the contexts and save the sequence of hashes into a file. Therefore, the boundaries of each segment are depended on the content of the object and not determined by a subjectively fixed block size. In 2006, Jesse Kornblum implemented ssdeep based on spamsum. Ssdeep is one of the principal programs for computing CTPH.

2.4.3 Statistically Improbable Features (SFI)

This approach was designed based on the idea of identifying several common attributes for every object under study and then doing a compare whether the features are different. The feature to be identified in this case refers to a sequence of consecutive bits that are identified based on the criteria from the file containing the object. For practical implementation, Vassil Roussev chose to utilize entropy in with a reason to find measurably improbable features. Based on this idea and findings, he developed an algorithm known as sdhash. The objective of this algorithm was to highlight object features that have a rear chance of occurring in other data objects (Oliver, Forman, & Cheng, 2014). The score generated by this algorithm range between zero and one hundred. This range refers to the confidence value to show how certain the algorithm is that the two data objects being compared have non-trivial amount of similarity.

2.4.4 Block-Based Rebuilding (BBR)

This function utilizes external data. The data is made up of arbitrary picked blocks, consistently or in a fixed way, in order to rebuild a file. The process computes the dissimilarity between the bytes of the initial file to the chosen blocks using the Hamming distance or any other measurement (Breitinger & Baier, 2012). Two of the best-perceived usage of this method are SimHash and bbHash.

2.4. Ssdeep

In 2006, Jesse Kornblum released ssdeep. Ssdeep is a fuzzy hashing algorithm that employs similarity digest to be able to determine if the hashes that represent two or more files have similarity. This tool produces fuzzy hashes, which are considered Context Triggered Piecewise Hashes (CTPH). CTPH considers two files similar if they have some identical sub-parts. Fuzzy hashes and other rolling hash methods are useful because they generate a continuous stream of hash values for a rolling window over the binary (Jesse, 2006). This algorithm computes a matching score ranging in between zero to one hundred percent. This score is translated as a weighted level of the similarity degree between the input files, whereby a higher score implies a more similarity between the input files (Martínez, et al., 2015). The signature format produced by ssdeep comprises of a header followed by one hash as indicated below. The following represents the content of the header in the latest versions is:

ssdeep,1.1—blocksize: primary hash: secondary hash, filename,

The word ssdeep in the header format above identifies the algorithm, 1.1 identifies the version of the file format, -- is a separator while the remaining part of the header points out the elements displayed below the header in this order (block size, primary hash, secondary hash and filename).

To compute similarity in file signatures, this tool takes signatures as regular strings and utilize Levenshtein distance measure to determine the degree of correlation. The Levenshtein distance is the number of operations required to change one string into another. There are various ways to characterize the Levenshtein distance, depending on which tasks are permitted. Ssdeep technique of calculating similarity is based on the Levenshtein distance between two strings. This distance differentiates two strings and determines the least number of operations required to transform one string to another. The edit operations allowed in the distance comparison are deletions, insertions, and substitutions of a single character and transpositions of two adjacent characters (Kim, et al., 2017).

In forensics, ssdeep is a widely known and used byte wise approximate matching application and it is considered by a couple of researchers as the de facto standard in a couple of cyber security areas (Breitinger, Stivaktakis, & Baier, 2013). To compute similarity in file using ssdeep, hashes of the input files are computed then used for comparison with another hash or with a data file. The outcome in both scenarios are the same, using one technique or the other is highly depended on the data available to the user. Ssdeep has increased enormous fame in malware detection and analysis in the recent past. At the time of this research, the latest version was 2.14.1 accessible through https://github.com/ssdeep-project/ssdeep/releases/tag/release-2.14.1 (Last accessed on 20th February 2020).

2.4.1 Levenshtein Distance

The Levenshtein distance is a string metric for measuring the number of edits which will transform one string to another. Informally, the Levenshtein distance between two strings is the minimum number of single-character edits (i.e. **insertions**, **deletions**, or **substitutions**) required to change one string into the other. By performing these three operations, the algorithm tries to modify first string to match the second one. In the end we get the Levenshtein distance.

The example below demonstrates how this algorithm can be used to compute distance between two strings "**Saturday**" and "**Sundays**". The character in bold format shows the characters being added or removed at each progression. The Levenshtein distance in this example is 5.

S**a**turday $^{\textbf{\textit{deletion}}}$→S**t**urday $^{\textbf{\textit{deletion}}}$ → Su**r**day $^{\textbf{deletion}}$ → Suday $^{\textbf{insertion}}$ → Su**n**day $^{\textbf{insertion}}$ → Sunday**s**

Levenshtein distance between two strings, *a* and *b* (of length |a| and |b| respectively) is given by equation below: -

Lev *a, b* (|a|, |b|).

Where:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Equation 1: Levenshtein Distance Formula

From this equation, 1(ai≠bi) is the indicator function equal to 0 while ai≠bi and equal to 1 otherwise, and lev a, b (i, j) is the distance between the first *i* characters of (a) and the first *j* characters of (b). The primary component inside minimum corresponds to deletion (from a to b), the second to insertion and the third to match, depending on whether the separate symbols are equivalent.

2.5. Ssdeep Test Cases on Files

Ssdeep can match files with different formats. Like .pdf, .png, .docx, json, csv. One of the most powerful features of ssdeep is the ability to match the hashes of input against a list of known hashes. The following 3 images will be used do some tests of the ssdeep pattern matching algorithm.

Image1

Image2

Image3

Md5 hash for the above 3 images can be computed in ssdeep using this command: -

#Md5sum Image1.png Image2.png Image3.png

*Md5 hash Image 1: (**616892b87f9c4b91a33ab68153ed9fb3**)*

*Md5 hash Image 2: (**5d5a9f4f78abfe3a626e1a7dc5153d92**)*

*Md5 Hash Image 3: (**616892b87f9c4b91a33ab68153ed9fb3**)*

Image 1 and Image 3 are the same hence the hash value is the same.

Fuzzy hashing can be used to identify similarity between files. The fuzzy hash of one file should be computed first and then use the matching mode to match the other file. A matching score percentage is calculated at the end of every match to show how similar the files are. The higher the matching score the more similar the files are.

Taking an example of the above images, **Image1** is first hashed to **image1hash.txt.** The hash value of **Image1** is compared with image3.png. **Image3** is a duplicate of **Image1** hence the match score is 100%.

File signatures for the above images can also be used to compute similarity. After generating fuzzy hashes for the 3 images, it's possible to compare those signatures to each other. The comparison of one or more files of signatures against each other using the –x flag as shown below: -

Command: *#ssdeep –x image1hash.txt imagehash3.png*

Results: *image3hash.png matches image1hash.png (100%)*

Comparison of unknown signatures against a set of known signatures can be done through use of –k flag as shown below: -

Command for hashing all 3 images into one file:

#ssdeep –b image1.png image2.png image3.png >pictures.txt.

#ssdeep –k pictures.txt –k image1.png image2.png image3.png

Results:

image1hash.txt:image1.png matches results.txt:image1.png (100)

image2hash.txt:image2.png matches results.txt:image2.png (100)

image3hash.txt:image3.png matches results.txt:image3.png (100)

image3hash.txt:image3.png matches results.txt:image1.png (100)

Ssdeep can also be used to find truncated files. To prove this functionality, I picked a word document with 582 pages and named this document **book 1**. I then made a copy of book 1 and truncated the copy to 270 pages and named this document **book 2**. To compute the similarity between these 2 documents (original and copy), I hashed the 2 files using this command: -

#ssdeep –b book1.pdf >book1hash.txt

#ssdeep –b book2.pdf >book2hash.txt

The 2 hash files are then saved in different folders. Book1 hash is saved in folder C while book 2 hash is saved in folder D. – p flag is used for pretty matching mode to compute similarity between these 2 files. –l flag is used for relative paths for filenames because the files are saved in different folders. The command below is used to compute similarity between the 2 files.

Command: *#ssdeep –l –r –p FolderC FolderD*

Results: *FolderC/book1hash.txt matches FolderD/book2hash.txt (41%)*

2.6 Conclusion

From the test result above, it is evident that ssdeep can be used to match different files with different formats as this has been validated in the test cases.

CHAPTER 3: DATA SET

The data used in this research was captured from a SCADA to substation network communication using Wireshark. A total of four packet capture files were generated for IEC 104 analysis in this research. They include:

i. SCADA_to_substation_normal. pcapng – This packet capture file was captured while the network communication between the SCADA and the substation was operating in normal conditions with no human interference. The network communication was between two nodes with IP 172.16.1.1 and 172.16.1.100. This communication took a period of 15 minutes 72 seconds and total of 172 packets were captured.
ii. SCADA_to_substation_attack1.pcapng – This packet capture file was captured while simulating Denial of Service attack (DoS). This attack was simulated by continuously switching on and off one of the communication nodes. This was achieved by using ASDU with C_CD_NA messages (double command). Not all commands were accepted. Out of the 12 commands that were received, only 4 replies were send back. This simulation took 15 minute and 83 seconds. A total of 217 packets were captured in this packet capture file.
iii. SCADA_to_substation_attack2.pcapng – This packet capture file was captured while simulating DoS attack within many substations. This attack involved network nodes ranging from 172.17.1.20 to 172.17.1.119 and was caused by ASDU messages C_IC_NA_1 (interrogation command). During this attack, a total of 688,979 packets were captured and it took a period of 16 minutes and 30 seconds.
iv. SCADA_to_substation_attack3.pcapng – This packet capture file was captured file simulating a scanning attack. This attack was meant to retrieve values from various objects on the target device. This attack used ASDU M_SP_NA messages (single point information). This communication was between two network nodes with IP 172.16.1.1 and 172.16.1.100. This attack took a period of 14 minutes and 17 seconds. A total of 147 packets were captured in this packet capture file.

CHAPTER 4: SYSTEM DESIGN AND IMPLIMENTATION

System architecture is a conceptual model that outlines the structural design of the system (Giraldo et al., 2019). The system architecture addresses how various system component interact to achieve the system functionality. This section provides an understanding of how the system design, structure and user requirements must be supported by the application. The developed ICS network profiling tool is a command line-based application that comprises of three python scripts. They include:

i. Create_profile.py Script – This python script is the first to be executed while running the application. It is integrated with Tshark to extract TCP payload from the packet capture files and save the data into a text file.
ii. Hash_profile.py Script – This is the second python script to be executed. It is used to hash the payload files extracted from the packet capture files. Ssdeep is integrated with this script for to be used for fuzzy hashing.
iii. Compare_profile.py scripts – This is the third python script in the application. This script is used for pattern matching the fuzzy hashes against the comparative files and generated a matching score. Ssdeep is used in this script to compute similarity between the input files.

The steps below discuss the processes and functionality of the scripts in detail;

Step 1: Import the Packet Capture File

This is the first step that takes place when the user runs the create_profile.py script. The main goal of this step is to import packet capture files into the application for further analysis of the IEC 104 protocol. This is achieved by specifying the names of the files to be analysed.

Step 2: TCP Payload Data Extraction

The second step involves the use of Tshark to filter the payload data from the packet capture files imported into the application. Tshark attribute filters used at this stage will determine the data to be extracted from the packet capture files. The Tshark command used in this application to filter IEC 104 payload is: *tshark, -r, pcap_file, -T, fields, -e, tcp. payload, 104apci.*

Step 3: TCP Payload Data Hashing

The third step uses the hash_profile.py script to hash the extracted text file in the stage above. Ssdeep is utilized at this level to fuzzy hash the files. So as to hash a file, ssdeep only utilizes a rolling hash algorithm to generate a pseudo-random value based on the current context of the input. A sliding window is overridden as input and every time the generated rolling hash matches the trigger; the algorithm computes a hash on the current piece. Originating from the spamsum algorithm, ssdeep uses block size for the trigger value.

Step 4: Profile Pattern Matching

The fourth step includes calculating the similarity between files using compare_profiles.py script. During comparison of two or more ssdeep hashes, ssdeep begins by determining the block size. Ssdeep computes the block size for the files to be compared. Following stage is the elimination of recurring sequences. Sequences indicate patterns in the input file which hold little information and therefore have little impact on the content. Finally, ssdeep computes a weighted Levenshtein distance between the two resulting hashes.

The Levenshtein distance represents the minimal number of edits required to change from one input to the other. The standardized match score represents a conservative weighted percentage of how much two inputs are ordered homologous sequences which represents how many of the bits are identical and in the same order. At last, the application generates a matching score in the range 0 –100. The score is a proportion of how similar the input files are, where a higher result implies a greater similarity of the files and lower score is an indication of dissimilarity. Figure below illustrates the processes and interconnections for the ICS network profiling tool.

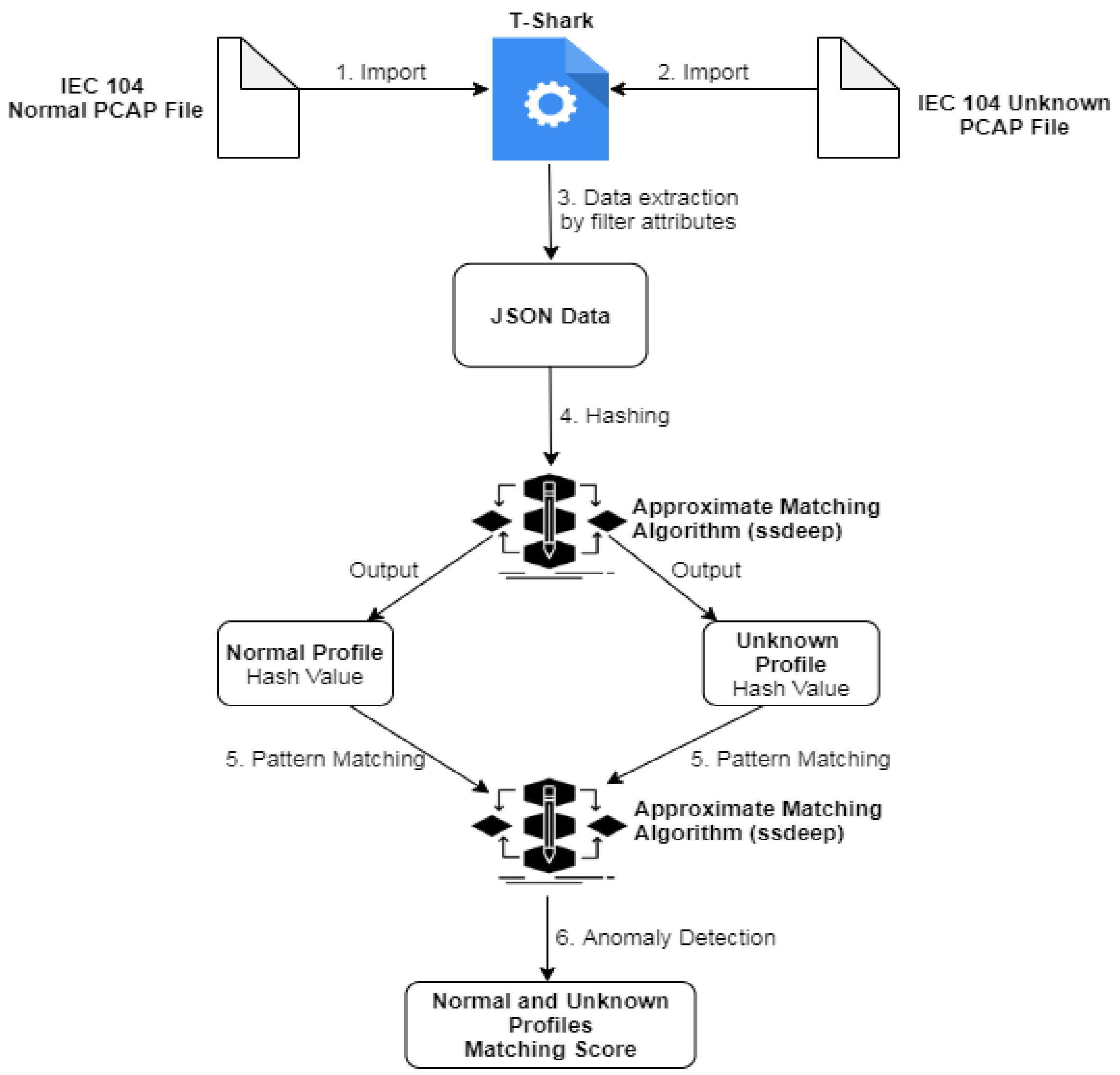


Figure 2: System Architecture

4.1 System Design

Since ICS network communication has proved stability within long term monitoring, approximate pattern matching algorithm was used in this research to detect anomalies in unknown ICS communication. The developed tool was designed to compute ICS network profile that would be used to identify any anomalies in ICS network communication. From this, ICS network administrators are better placed to monitor network communication based on solid facts.

4.2 Use Case Diagram

The application users and interactions of various functionalities of the ICS network profiling tool is illustrated in the use case diagram below. This use case describes the key features offered by the developed prototype. All the external tools and users interacting with the developed solution are also highlighted.

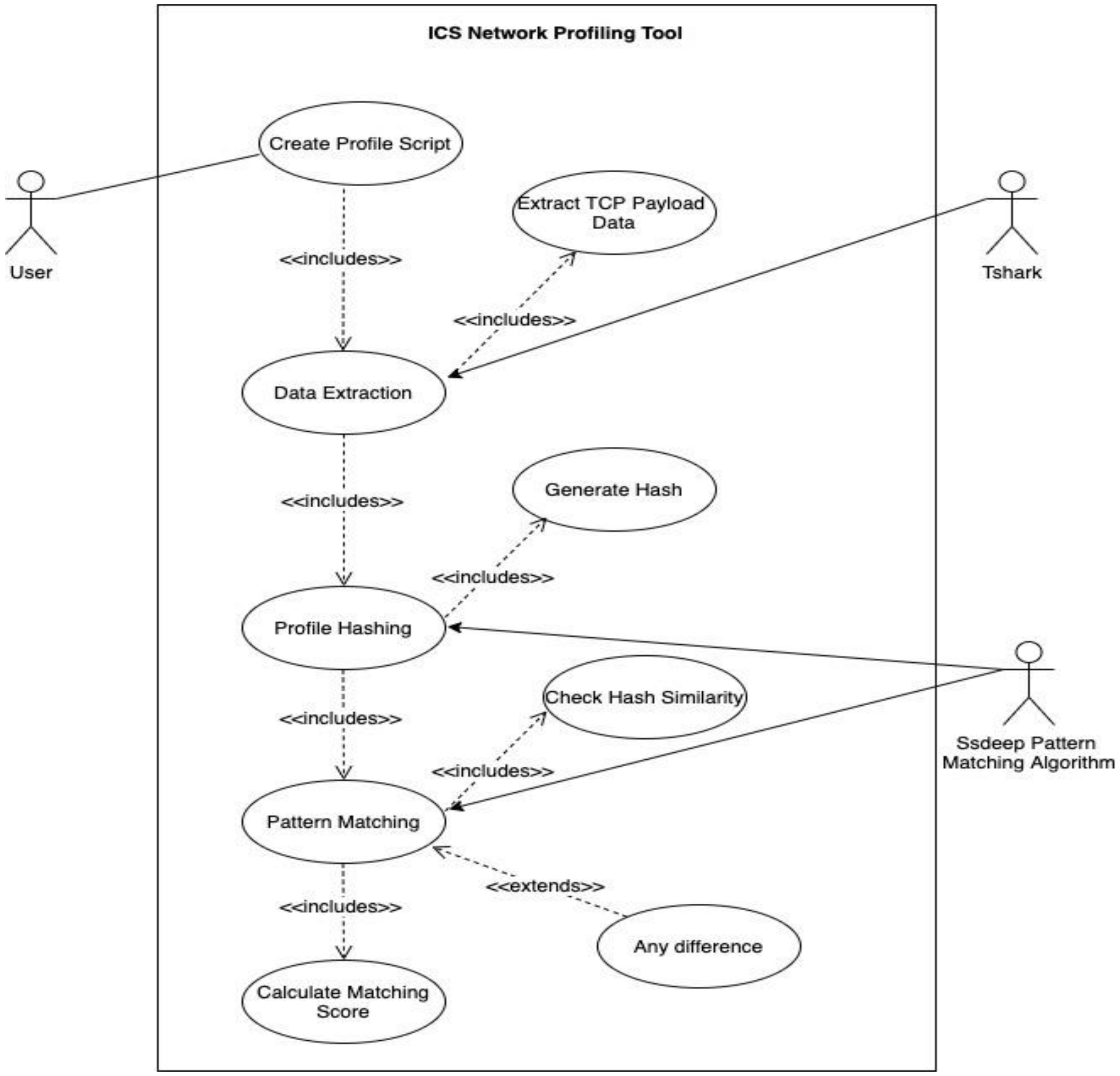


Figure 3: Use Case Diagram

4.2.1 Use Case 1: Create-Profile.py Script

Table 1: Create-profile.py Script Use Case

Use Case Name:	**Create-profile.py script**
Description:	- This use case contains instructions to extract TCP payload data from the PCAP file using Tshark, hashing the network profile and matching the known profile against the unknown profile.
Primary Actor:	- User
Secondary Actor:	- None
Include Use Cases:	- Data Extraction - Profile Hashing - Pattern Matching - Calculate Matching Score
Extend Use Cases:	- None
Precondition:	- ICS communication PCAP file must have been captured.
Post Condition:	- The python script extracts the TCP payload from the packet capture file.
Main Flow:	- The system triggers TCP payload data extraction use case and waits for it to complete.
Alternative Flows:	- Failed data extraction. - Restart Tshark and rerun the script. - The use case ends.

4.2.2 Use Case 2: Data Extraction

Table 2: Data Extraction Use Case

Use Case Name:	**Data Extraction**
Description:	- This use case contains procedures for the application to extract TCP payload data from the ICS captured packet capture file using Tshark as the analysis tool.

Use Case Name:	**Data Extraction**
Primary Actor:	- None
Secondary Actor:	- Tshark
Include Use Cases:	- Extract TCP payload data
Extend Use Cases:	- None
Precondition:	- Tshark must be preinstalled.
Post Condition:	- The application extracts the TCP payload from the ICS packet capture files and saves the results into a text format.
Main Flow:	- The application uses tshark to filter IEC104 payload using the following command: *tshark –r packet file.pcapng –T fields –e iec60870_104.type –e tcp.payload 104apci.* - TCP payload is extracted and saved to a text file for further processing.
Alternative Flows:	- Tshark fails to extract payload. - Restart Tshark.

4.2.3 Use Case 3 Profile Hashing

Table 3: Profile Hashing Use Case

Use Case Name:	**Profile Hashing**
Description:	- This use case contains the procedures for the system to hash the text file saved after extraction of the payload. Ssdeep is used for generating the fuzzy hash.
Primary Actor:	- None
Secondary Actor:	- Ssdeep Pattern Matching Algorithm.
Include Use Cases:	- Generate Hash
Extend Use Cases:	- None

Use Case Name:	Profile Hashing
Precondition:	- Save a text file containing TCP payload.
Post Condition:	- Fuzzy hash for the saved TCP payload.
Main Flow:	- This use case computes the fuzzy hash of the input file and scores the results.
Alternative Flows:	- File hashing fails. - Rerun ssdeep.

4.2.4 Use Case 4 Pattern Matching

Table 4: Pattern Matching Use Case

Use Case Name:	Pattern Matching
Description:	- This use case compares the known and unknown profiles and computes the matching score.
Primary Actor:	- None
Secondary Actor:	- Ssdeep Pattern Matching Algorithm.
Include Use Cases:	- Check hash similarity.
Extend Use Cases:	- Any difference.
Precondition:	- Compute profile hash.
Post Condition:	- Matching score of the profiles.
Main Flow:	- Ssdeep matches the unknown profile against a file of known profile. - The similarity match is calculated between the two input files.
Alternative Flows:	- Pattern matching algorithm fails to run. - Rerun ssdeep.

4.3 Sequence Diagram

This is an interaction diagram that depicts time ordering of messages between different system objects. Creating a normal profile for an ICS network communication and comparing the normal profile against an unknown communication to identify anomalies in the network communication are amongst the main features of the ICS network profiling tool. The figure below indicates the main flow of data and events with regards to the main functionalities of the proposed system.

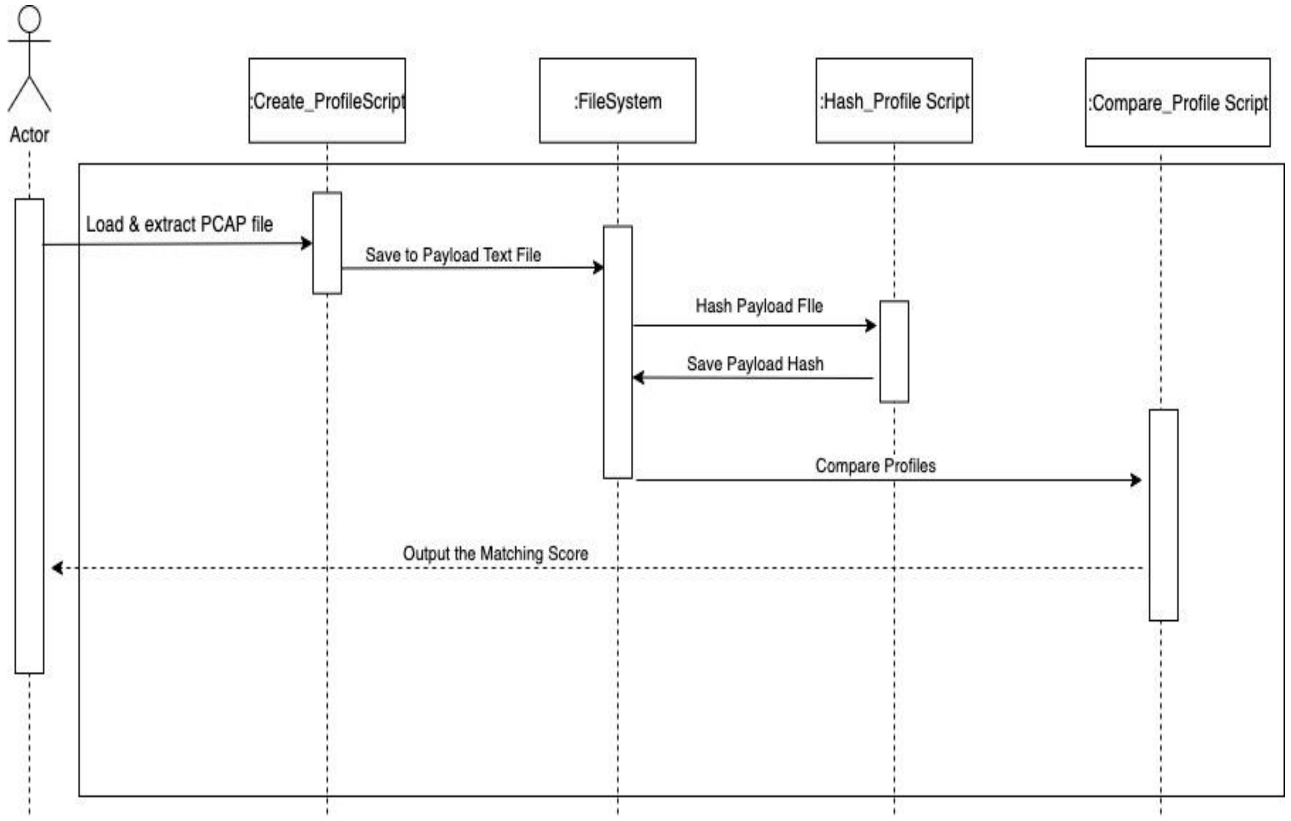


Figure 4: Sequence Diagram

4.3.1 Load and Extract PCAP File

Create profile python script starts by triggering Tshark. Tshark network protocol analyser tool is used to extract network communication TCP payload from the packet capture files. This tool goes through the ICS packet capture files to analyse the IEC 104 payload data according to the specified attribute filters. The extracted data from the TCP payload is then saved into a text file in the file system.

4.3.2 Hash Payload File

The application uses ssdeep algorithm to fuzzy hash the TCP payload data extracted from the ICS packet capture file. The hash file is also saved in a text file in the file system.

4.3.3 Compare Profiles

Compare-profile python script is used to compute similarity between the input files. The comparison can be between the normal profile hashes against the unknown profile hash to compute similarity between the profiles. Ssdeep pattern matching algorithm computes the similarity between the profiles to generate the matching score. The higher the similarity score between the profiles, the fewer anomalies can be detected in the compared profiles.

CHAPTER 5: RESULTS

5.1 Overview

Designing of the proposed application was followed by its development and implementation as per system design. This chapter concentrates on the testing process of the application. On the implementation section, the implementation environments are explored together with the main functionality of the application. The testing part focuses on usability testing and functional testing to validate if the application attains the objectives of the proposed solution. The relevant screenshots of the prototype are provided.

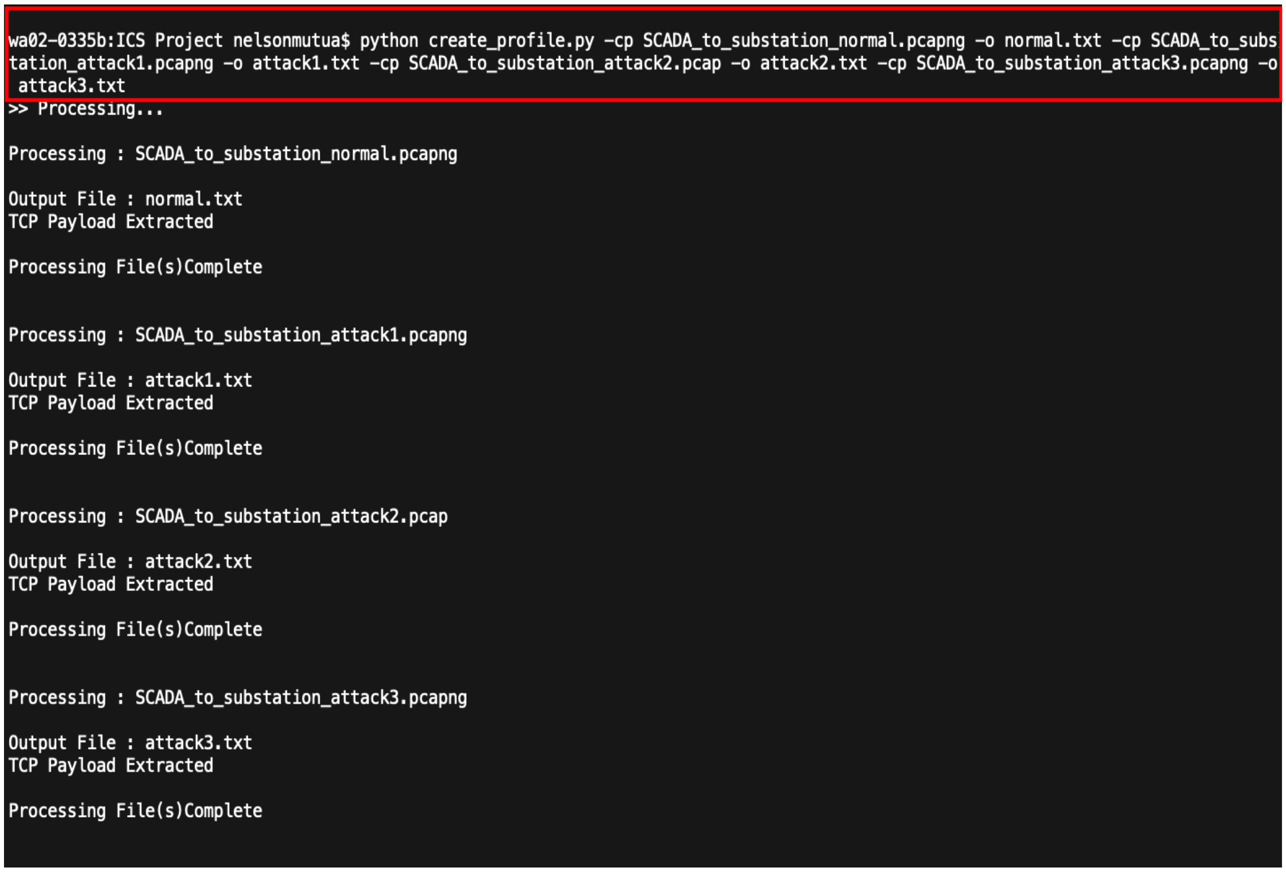
5.2 Ssdeep Set-up and Installation

Ssdeep-2.14.1 is the latest version accessible in GitHub as of the time of documenting this research. The source code can be downloaded through this link: https://github.com/ssdeep-project/ssdeep/releases/tag/release-2.14.1(Last accessed on 13th March 2020). After downloading the algorithm, the user can change into the downloaded directory by using this command: *cd ssdeep-2.14.1*. The next installation step is executing the *./bootstrap* file so as to generate the configure script. Once the script is generated, it is executed by *./configure*. Lastly, the ssdeep program can be complied by running make command, *make* and install it by *make install*. For the installation to work, the user logged into the computer must be a root user on most operating systems to be able to install the program to its default location, /usr/local/bin. This can also be achieved by running *sudo make install command*.

5.3 TCP Payload Extraction

The user is required to run create_profile.py script in order for the TCP payload data to be extracted. Tshark is integrated with the script to extract specific attributes from the ICS packet capture file. In the create_profile.py script, flag -cp is a predefined argument for creating profile while -o is an argument for outputting the results into a text file. Extracting TCP payload from the normal communication packet capture file was achieved through this command:

python create_profile.py -cp SCADA_to_substation_normal.pcapng -o Normal_Payload.txt -cp SCADA_to_substation_attack1.pcapng -o attack1.txt -cp SCADA_to_substation_attack2.pcap -o attack2.txt -cp SCADA_to_substation_attack3.pcapng -o attack3.txt



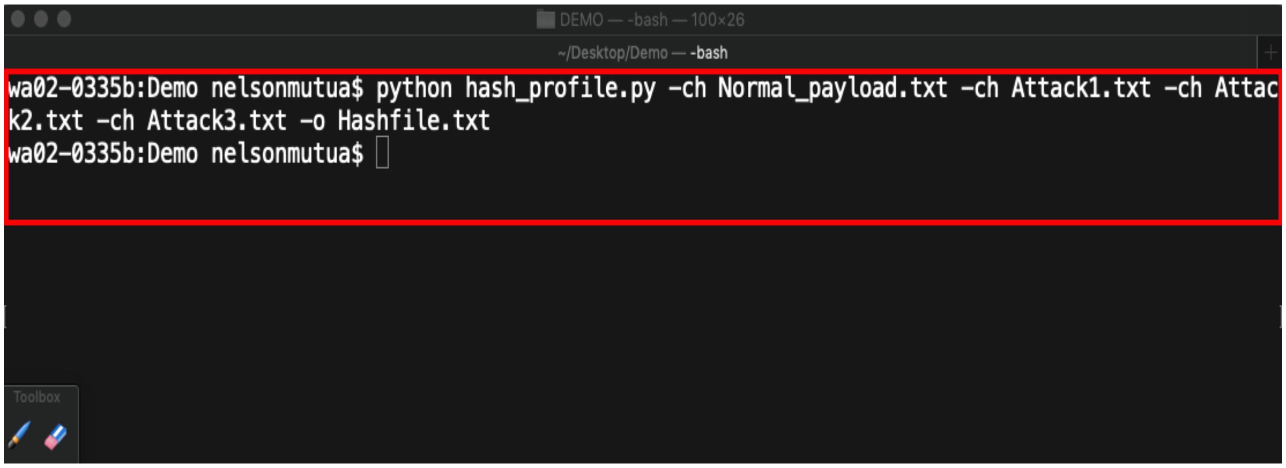
```
wa02-0335b:ICS Project nelsonmutua$ python create_profile.py -cp SCADA_to_substation_normal.pcapng -o normal.txt -cp SCADA_to_subs
tation_attack1.pcapng -o attack1.txt -cp SCADA_to_substation_attack2.pcap -o attack2.txt -cp SCADA_to_substation_attack3.pcapng -o
 attack3.txt
>> Processing...

Processing : SCADA_to_substation_normal.pcapng

Output File : normal.txt
TCP Payload Extracted

Processing File(s)Complete


Processing : SCADA_to_substation_attack1.pcapng

Output File : attack1.txt
TCP Payload Extracted

Processing File(s)Complete


Processing : SCADA_to_substation_attack2.pcap

Output File : attack2.txt
TCP Payload Extracted

Processing File(s)Complete


Processing : SCADA_to_substation_attack3.pcapng

Output File : attack3.txt
TCP Payload Extracted

Processing File(s)Complete
```

Figure 5: TCP Payload Extraction from PCAP

5.4 TCP Payload Hashing

The user is required to run Hash_profile.py script in order to hash the payload text files extracted in the step above. Flag -ch is defined in the script as an argument for creating hash using ssdeep while -o is defined as an argument to output all the file hashes into a text file. The command below was used to hash all the payload files extracted in the stage above in this research.

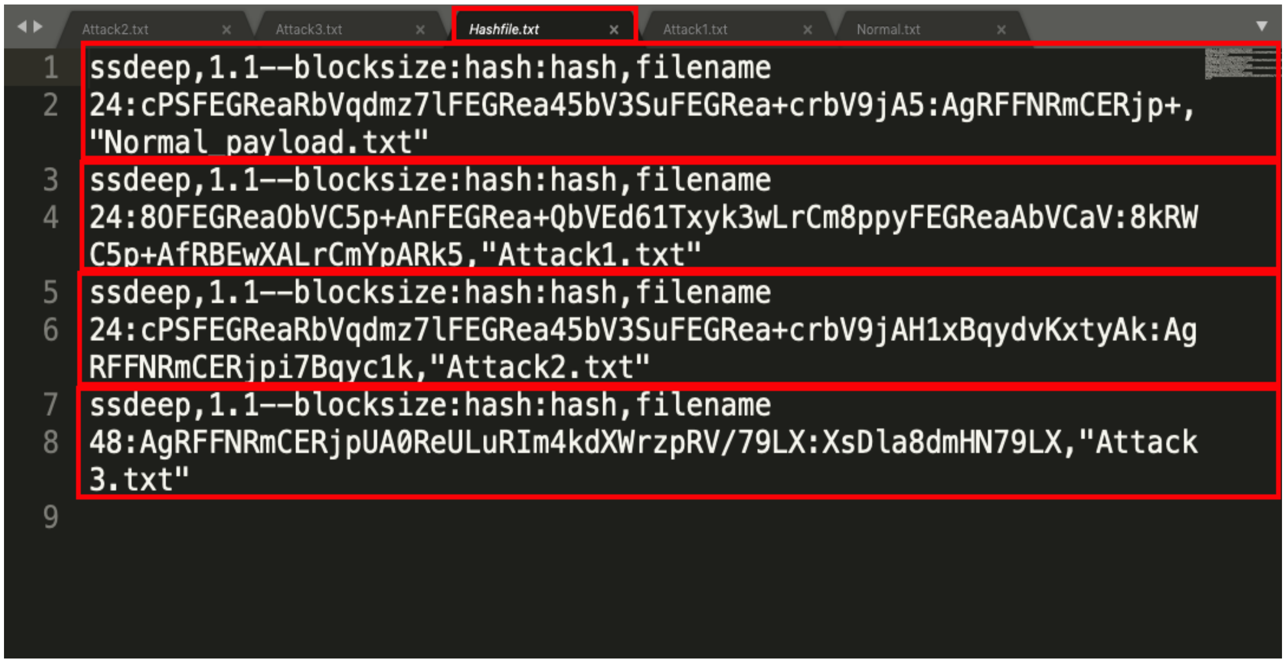
Python hash_profile.py -ch Normal_Payload.txt -ch attack1.txt -ch attack2.txt -ch attack3.txt -o HashFile.txt



```
wa02-0335b:Demo nelsonmutua$ python hash_profile.py -ch Normal_payload.txt -ch Attac
k2.txt -ch Attack3.txt -o Hashfile.txt
wa02-0335b:Demo nelsonmutua$ 
```

Figure 6: Payload hashing command

The figure below shows the hash values generated from the ICS payload files and saved into a text file.



```
ssdeep,1.1--blocksize:hash:hash,filename
24:cPSFEGReaRbVqdmz7lFEGRea45bV3SuFEGRea+crbV9jA5:AgRFFNRmCERjp+,
"Normal_payload.txt"
ssdeep,1.1--blocksize:hash:hash,filename
24:80FEGRea0bVC5p+AnFEGRea+QbVEd61Txyk3wLrCm8ppyFEGReaAbVCaV:8kRW
C5p+AfRBEwXALrCmYpARk5,"Attack1.txt"
ssdeep,1.1--blocksize:hash:hash,filename
24:cPSFEGReaRbVqdmz7lFEGRea45bV3SuFEGRea+crbV9jAH1xBqydvKxtyAk:Ag
RFFNRmCERjpi7Bqyc1k,"Attack2.txt"
ssdeep,1.1--blocksize:hash:hash,filename
48:AgRFFNRmCERjpUA0ReULuRIm4kdXWrzpRV/79LX:XsDla8dmHN79LX,"Attack
3.txt"
```

Figure 7: Normal and Attack hash values

5.5 Pattern Matching

Compare_profile.py script is used for pattern matching purposes. The user can compare one or more payload files against the hash values using –c flag. The file containing the hash values to be used for comparison against the input files can flagged using -hf. The output result is the similarity percentage between the hash values and the input files.

The results obtained show that the normal profile hash value matches Normal_Payload.txt file 100%. Attack 1 hash value matches Normal_Payload.txt file 50%. Since SCADA_to_substation_attack1 contained packets of DoS attack simulation, the attack 1 communication profile could not match the normal communication 100%. Attack 1 packet capture file contained an ASDU messages with double commands. The attack simulation caused a malfunction between the communication nodes. This resulted to instability in the IEC 104 network communication. The application was able to detect the threshold of the malicious packets contained in attack 1 file.

The network profiling tool can also compare one hash file against many files. Compare_profiles.py script was used to compute similarity of the normal ICS hash value against attack 1, 2 and 3 payloads extracted from the packet capture files. The similarity matching score between the normal profile and attack 1, attack 2 and attack 3 is 50%, 86% and 50 respectively. The application was able to detect malicious communication in all the packet files. The command below was used to pattern match the normal hash against attack 1, attack 2 and attack 3 payloads.

python compare_profile.py -hf Hashfile.txt -c Normal_payload.txt -c Attack1.txt -c Attack2.txt -c Attack3.txt.

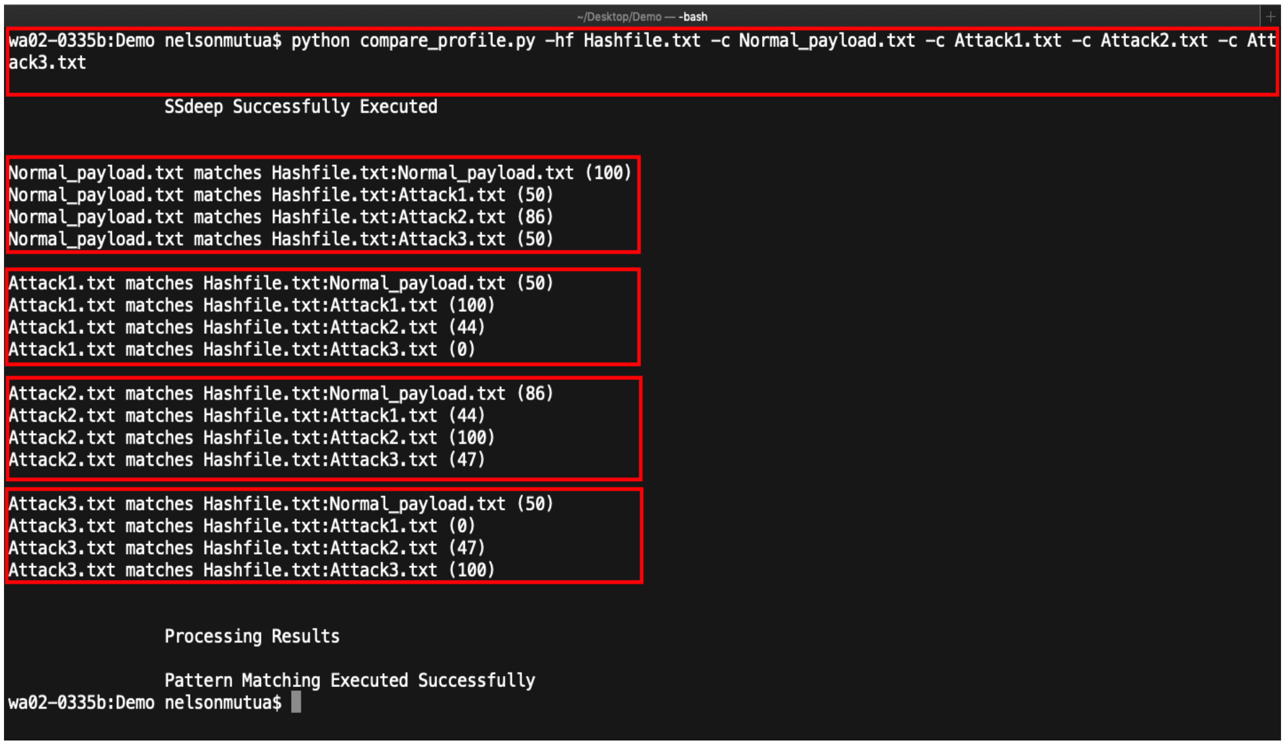
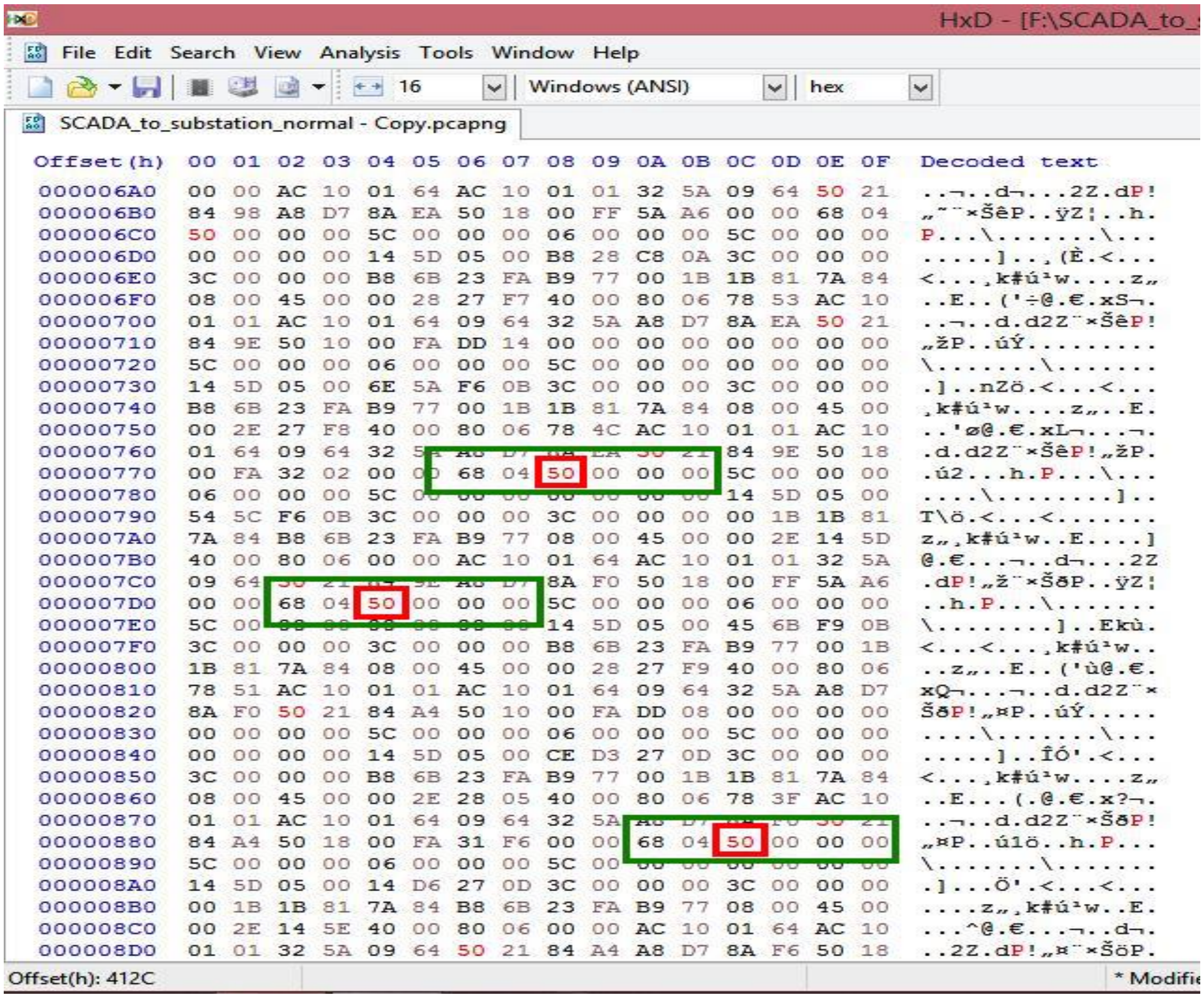

```
~/Desktop/Demo — -bash
wa02-0335b:Demo nelsonmutua$ python compare_profile.py -hf Hashfile.txt -c Normal_payload.txt -c Attack1.txt -c Attack2.txt -c Att
ack3.txt

                SSdeep Successfully Executed


Normal_payload.txt matches Hashfile.txt:Normal_payload.txt (100)
Normal_payload.txt matches Hashfile.txt:Attack1.txt (50)
Normal_payload.txt matches Hashfile.txt:Attack2.txt (86)
Normal_payload.txt matches Hashfile.txt:Attack3.txt (50)

Attack1.txt matches Hashfile.txt:Normal_payload.txt (50)
Attack1.txt matches Hashfile.txt:Attack1.txt (100)
Attack1.txt matches Hashfile.txt:Attack2.txt (44)
Attack1.txt matches Hashfile.txt:Attack3.txt (0)

Attack2.txt matches Hashfile.txt:Normal_payload.txt (86)
Attack2.txt matches Hashfile.txt:Attack1.txt (44)
Attack2.txt matches Hashfile.txt:Attack2.txt (100)
Attack2.txt matches Hashfile.txt:Attack3.txt (47)

Attack3.txt matches Hashfile.txt:Normal_payload.txt (50)
Attack3.txt matches Hashfile.txt:Attack1.txt (0)
Attack3.txt matches Hashfile.txt:Attack2.txt (47)
Attack3.txt matches Hashfile.txt:Attack3.txt (100)


                Processing Results

                Pattern Matching Executed Successfully
wa02-0335b:Demo nelsonmutua$ 
```

Figure 8: Pattern Matching Different Attack Files

5.6 Man-In-The-Middle Attack Simulation

Through the MITM attack, an attacker can inject or alter readings and commands in the communication stream in real time. While intercepting packets, some packet can be dropped, altered or a new packet can be injected with arbitrary outcome. Attackers can inject malicious data into the ICS communication to cause harm. In this research, MITM attack was simulated by artificially modifying the first APCI control field byte for the first 20 IEC 104 packets in the normal packet capture file. This binary modification was performed by using a binary tool editor known as Hex Editor. For instance, the control field in a packet binary data 6804**43**000000 was modified to 6804**50**000000. The bolded characters represent the changes.


HxD - [F:\SCADA_to_
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
SCADA_to_substation_normal - Copy.pcapng

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

000006A0  00 00 AC 10 01 64 AC 10 01 01 32 5A 09 64 50 21  ..¬..d¬...2Z.dP!
000006B0  84 98 A8 D7 8A EA 50 18 00 FF 5A A6 00 00 68 04  „˜¨×ŠêP..ÿZ¦..h.
000006C0  50 00 00 00 5C 00 00 00 06 00 00 00 5C 00 00 00  P...\.......\...
000006D0  00 00 00 00 14 5D 05 00 B8 28 C8 0A 3C 00 00 00  .....]..¸(È.<...
000006E0  3C 00 00 00 B8 6B 23 FA B9 77 00 1B 1B 81 7A 84  <...¸k#ú¹w....z„
000006F0  08 00 45 00 00 28 27 F7 40 00 80 06 78 53 AC 10  ..E..('÷@.€.xS¬.
00000700  01 01 AC 10 01 64 09 64 32 5A A8 D7 8A EA 50 21  ..¬..d.d2Z¨×ŠêP!
00000710  84 9E 50 10 00 FA DD 14 00 00 00 00 00 00 00 00  „žP..úÝ.........
00000720  5C 00 00 00 06 00 00 00 5C 00 00 00 00 00 00 00  \.......\.......
00000730  14 5D 05 00 6E 5A F6 0B 3C 00 00 00 3C 00 00 00  .]..nZö.<...<...
00000740  B8 6B 23 FA B9 77 00 1B 1B 81 7A 84 08 00 45 00  ¸k#ú¹w....z„..E.
00000750  00 2E 27 F8 40 00 80 06 78 4C AC 10 01 01 AC 10  ..'ø@.€.xL¬...¬.
00000760  01 64 09 64 32 5A A8 D7 8A EA 50 21 84 9E 50 18  .d.d2Z¨×ŠèP!„žP.
00000770  00 FA 32 02 00 00 68 04 50 00 00 00 5C 00 00 00  .ú2...h.P...\...
00000780  06 00 00 00 5C 00 00 00 00 00 00 00 14 5D 05 00  ....\........]..
00000790  54 5C F6 0B 3C 00 00 00 3C 00 00 00 00 1B 1B 81  T\ö.<...<.......
000007A0  7A 84 B8 6B 23 FA B9 77 08 00 45 00 00 2E 14 5D  z„¸k#ú¹w..E....]
000007B0  40 00 80 06 00 00 AC 10 01 64 AC 10 01 01 32 5A  @.€...¬..d¬...2Z
000007C0  09 64 50 21 84 9E A8 D7 8A F0 50 18 00 FF 5A A6  .dP!„ž¨×ŠðP..ÿZ¦
000007D0  00 00 68 04 50 00 00 00 5C 00 00 00 06 00 00 00  ..h.P...\.......
000007E0  5C 00 00 00 00 00 00 00 14 5D 05 00 45 6B F9 0B  \........]..Ekù.
000007F0  3C 00 00 00 3C 00 00 00 B8 6B 23 FA B9 77 00 1B  <...<...¸k#ú¹w..
00000800  1B 81 7A 84 08 00 45 00 00 28 27 F9 40 00 80 06  ..z„..E..('ù@.€.
00000810  78 51 AC 10 01 01 AC 10 01 64 09 64 32 5A A8 D7  xQ¬...¬..d.d2Z¨×
00000820  8A F0 50 21 84 A4 50 10 00 FA DD 08 00 00 00 00  ŠðP!„¤P..úÝ.....
00000830  00 00 00 00 5C 00 00 00 06 00 00 00 5C 00 00 00  ....\.......\...
00000840  00 00 00 00 14 5D 05 00 CE D3 27 0D 3C 00 00 00  .....]..ÎÓ'.<...
00000850  3C 00 00 00 B8 6B 23 FA B9 77 00 1B 1B 81 7A 84  <...¸k#ú¹w....z„
00000860  08 00 45 00 00 2E 28 05 40 00 80 06 78 3F AC 10  ..E...(.@.€.x?¬.
00000870  01 01 AC 10 01 64 09 64 32 5A A8 D7 8A F0 50 21  ..¬..d.d2Z¨×ŠðP!
00000880  84 A4 50 18 00 FA 31 F6 00 00 68 04 50 00 00 00  „¤P..ú1ö..h.P...
00000890  5C 00 00 00 06 00 00 00 5C 00 00 00 00 00 00 00  \.......\.......
000008A0  14 5D 05 00 14 D6 27 0D 3C 00 00 00 3C 00 00 00  .]...Ö'.<...<...
000008B0  00 1B 1B 81 7A 84 B8 6B 23 FA B9 77 08 00 45 00  ....z„¸k#ú¹w..E.
000008C0  00 2E 14 5E 40 00 80 06 00 00 AC 10 01 64 AC 10  ...^@.€...¬..d¬.
000008D0  01 01 32 5A 09 64 50 21 84 A4 A8 D7 8A F6 50 18  ..2Z.dP!„¤¨×ŠöP.

Offset(h): 412C                                             * Modifie
```

Figure 9: Binary Data Manipulation

After manipulating the first APCI control fields in the first 20 packets, the payload is reextracted from the edited SCADA_to_substation_normal.pcapng using create_profile.py script. The terminal command used to extract the payload from the manipulated pcap file is: *python create_profile.py -cp SCADA_to_substation _edited.pcapng -o Edited_Payload.txt*

The normal and edited payloads were then hashed using hash_profile.py script. The python command used to hash the normal and edited payloads is: *Python hash_profile.py -ch Normal_payload.txt -ch Edited_payload.txt -o Editedhash.txt*

Similarity was then computed to compare the original normal payload file with the manipulated payload. The matching score between these packet capture files is 94%. This is a proof that this tool can be used to detect for any manipulation of binary data in packet capture files. The python command below was used to pattern match the normal and manipulated packet capture file. *Python compare_profile.py Editedhash.txt -c Normal_payload.txt -c Edited_payload.txt.*

5.7 System Validation

System validation is defined as the set of activities and processes aimed at verifying that the application is performing as expected in line with the set objectives. The main objective of this research was to develop an application that profiles ICS communication using approximate matching algorithm. The implemented application was able to achieve the above in that it has been able to extract IEC 104 payload from ICS packet capture files. The application can then hash the payload using ssdeep and compare different communication profiles to compute the matching score.

Testing was done to ensure the developed application did not have bugs. A bug is defined as an error or flaw in a computer application that causes the software to give an incorrect or unexpected result (SteelKiwi, 2015). Testing for bugs was done through conducting several tests repeatedly to determine the result given was consistent. NIST's guidelines state that the results of an application should be repeatable and reproducible (Brunty, 2017).

i. **Repeatability** refers to obtaining the same results when using the same method on identical test items in the same environment by the same operator using the same equipment within short intervals of time. The tests were performed five times using the same packet capture files and produced the same results.

ii. **Reproducibility** refers to obtaining the same results being obtained when using the same method on identical test items in different laboratories with different operators utilizing different equipment. The application was installed in Mac OS, Linux and Windows operating systems and peer reviewed using the same packet capture files produced the same results for all instances.

The performance of the developed application was determined by using three metrics namely accuracy, sensitivity and specifying. Accuracy is the overall success rate of the application. It gives the proportion of the input test that are correctly identified. Sensitivity also known as true positive is the rate at which attack profiles were detected by the normal profile to ascertain attacks packets. Specifying also known as true negative is the rate at which normal communication was correctly detected as normal and attack free. Therefore, sensitivity quantifies the avoidance of false negative (attack profiles incorrectly detected as normal profile) while specifying quantifies the avoidance of false positive (normal communication incorrectly detected to contain an attack).

From the results obtained, the normal profile hash value matched the normal payload 100%. Out of the five tests done on every operating system, all the results turned out to be true negative (normal communication was correctly detected). Attack 1, 2 and 3 profiles were positively identified to contain attack packets in comparison with normal profile with matching scores of 50%, 86% and 50% respectively.

5.8 Conclusion

The objectives of the study sought to understand the ICS network communication and IEC 104 protocol. Consequently, the study sought to analyse different pattern matching algorithms and find out how similarity is computed in files. Further, how to profile ICS network communication using pattern matching algorithm. Extensive research in this study, along with testing different algorithms in files, helped to inform the need of the developed ICS network profiling tool. This tool was built to inform the ICS network administrators and researchers on how unknown ICS network communication can be analysed and determine whether network packet files contain attacks. Unknown network communication profile highly differing with the known profile can be an indication of malicious packets or attacks contained in a network communication. Encouragingly, the application was received with positive feedback from the need for its functionality and usability.

5.9 Future Work

ICS is known to have relatively fixed operational objects and business processes which result in relatively stable traffic pattern under normal conditions. Fluctuations of network traffic generally indicate the status change in ICS communication. The scope of this study was to detect anomalies in ICS network packet capture files using approximate pattern matching algorithm with focus on IEC 104 protocol. Future work will concern studying how to analyse and detect anomalies in ICS network communication in a real time manner. This can also be enhanced by incorporating other network attributes such as the source and destination IP, port number and average time interval between adjust packets. Pattern matching algorithm can then be used to match to match different attack patterns. Finally, SMS and email alert capability feature and identification of the attack packets can also be developed for the tool to offer real time monitoring and more comprehensive security service.

REFERENCES

Breitinger, F., & Baier, H. (2012). A fuzzy hashing approach based on random sequences and hamming distance.

Breitinger, F., Stivaktakis, G., & Baier, H. (2013). FRASH: A framework to test algorithms of similarity hashing. Digital Investigation, 10, S50-S58.

Brunty. (2017). Validation of Forensic Tools and Software: A Quick Guide for the Digital Forensic Examiner.

Case, D. U. (2016). Analysis of the cyber-attack on the Ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC), 388.

D. Hadžiosmanović, D. Bolzoni, S. Etalle, and P. Hartel, "Challenges and opportunities in securing industrial control systems," in 2012 Complexity in Engineering (COMPENG). Proceedings, June 2012, pp. 1–6.

Giraldo, J., Urbina, D., Cardenas, A. A., & Tippenhauer, N. O. (2019, June). Hide and seek: An architecture for improving attack-visibility in industrial control systems. In International Conference on Applied Cryptography and Network Security (pp. 175-195). Springer, Cham.

Goel, S., Hong, Y., Papakonstantinou, V., & Kloza, D. (2015). Smart grid security. London: Springer.

Harbour, N. (2002). Dcfldd. defense computer forensics lab. In Net5. 5.2 (Vol. 1).

Ho, T., Oh, S. R., & Kim, H. (2017). A parallel approximate string matching under Levenshtein distance on graphics processing units using warp-shuffle operations. PloS one, 12(10).

Kerkers, M., Chromik, J. J., Remke, A., & Haverkort, B. R. (2018, February). A Tool for Generating Automata of IEC60870-5-104 Implementations. In International Conference on Measurement, Modelling and Evaluation of Computing Systems.

Langer, R. (2013, November). Resources. Retrieved from Langner: Retrieved 13th March 2020 from (langer.com): https://www.langner.com/wp-content/uploads/2017/03/to-kill-a-centrifuge.pdf

Leith, H. M., & Piper, J. W. (2013). Identification and application of security measures for petrochemical industrial control systems. Journal of Loss Prevention in the Process Industries, 26(6), 982-993.

Martínez, V. G., Álvarez, F. H., Encinas, L. H., & Ávila, C. S. (2015). A new edit distance for fuzzy hashing applications. In Proceedings of the international conference on security and management (sam) (p. 326). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Matoušek, P. (2017). Description and analysis of IEC 104 Protocol. Faculty of Information Technology, Brno University of Technology, Tech. Rep.

NIST, S. (2014). 800-168,“ Approximate Matching: Definition and Terminology,” Jul.

Oliver, J., Forman, S., & Cheng, C. (2014, November). Using randomization to attack similarity digests. In International Conference on Applications and Techniques in Information Security (pp. 199-210). Springer, Berlin, Heidelberg.

Remke, A., & Haverkort, B. R. (2018, February). A Tool for Generating Automata of IEC60870-5-104 Implementations. In Measurement, Modelling and Evaluation of Computing Systems: 19th International GI/ITG Conference, MMB 2018, Erlangen, Germany, February 26-28, 2018, Proceedings.

Skoko, V., Atlagic, B., & Isakov, N. (2014, November). Comparative realization of IEC 60870-5 industrial protocol standards. In 2014 22nd Telecommunications Forum Telfor (TELFOR) (pp. 987-990). IEEE.

Tridgell, A. (1999). Efficient algorithms for sorting and synchronization.

Zetter, Kim. (2016). Inside the cunning, unprecedented hack of Ukraine’s power grid. Wired Magazine. Retrieved 12th February 2020 from (wired.com) https://www.wired.com/2016/03/inside-cunningUnprecedented-hack-ukraines-power-grid/