



# **NAVAL POSTGRADUATE SCHOOL**

**MONTEREY, CALIFORNIA**

## **THESIS**

**USING MACHINE LEARNING TO PREDICT THE  
INSIDER THREAT IN A NETWORK ENVIRONMENT**

by

Natasha K. Niemann and Raymond G. Blockmon

March 2021

Thesis Advisor:

Co-Advisor:

Second Reader:

Vinnie Monaco

Brian P. Wood

Charles Timm,

TRAC - NETCOM

**Approved for public release. Distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                 |                                                                |                                                            |  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|----------------------------------------------------------------|------------------------------------------------------------|--|
| <b>REPORT DOCUMENTATION PAGE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                 |                                                                | <i>Form Approved OMB<br/>No. 0704-0188</i>                 |  |
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                 |                                                                |                                                            |  |
| <b>1. AGENCY USE ONLY<br/>(Leave blank)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                 | <b>2. REPORT DATE</b><br>March 2021                            | <b>3. REPORT TYPE AND DATES COVERED</b><br>Master's thesis |  |
| <b>4. TITLE AND SUBTITLE</b><br>USING MACHINE LEARNING TO PREDICT THE INSIDER THREAT IN A NETWORK ENVIRONMENT                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                 |                                                                | <b>5. FUNDING NUMBERS</b>                                  |  |
| <b>6. AUTHOR(S)</b> Natasha K. Niemann and Raymond G. Blockmon                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                 |                                                                |                                                            |  |
| <b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b><br>Naval Postgraduate School<br>Monterey, CA 93943-5000                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                 |                                                                | <b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>            |  |
| <b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b><br>N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                 |                                                                | <b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b>    |  |
| <b>11. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                 |                                                                |                                                            |  |
| <b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b><br>Approved for public release. Distribution is unlimited.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                 |                                                                | <b>12b. DISTRIBUTION CODE</b><br>A                         |  |
| <b>13. ABSTRACT (maximum 200 words)</b><br><br>In the past, cybersecurity professionals relied upon Security Event and Information Management systems to ingest network, server, and host logs to assist in detecting suspicious and malicious activity in the network. Detecting threat activities also included manually inspecting packet captures to glean clues of nefarious activity.<br><br>Our research involves machine learning. We developed a model that observes the packet headers' characteristics when a user accessed a remote file server. Data sets were introduced and host-server configurations were used to determine if our classification model was consistent in identifying file access behavior. We were able to predict and classify file access behavior, such as uploading, downloading, deleting, and moving files on a file server, based upon using headers. The results from deriving the classifications were similar when using different host-server configurations and files.<br><br>Our research demonstrated potential avenues to study file access behavior on an enterprise network. Information repositories like file servers, SharePoint, and online data hosting sites such as Dropbox present a surface threat for information theft. Classifying file access behavior with these online resources presents a valuable goal for cybersecurity. |                                                                 |                                                                |                                                            |  |
| <b>14. SUBJECT TERMS</b><br>machine learning, networks, security, network operation centers, cyber center, cyber, security threat, computer network                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                 |                                                                | <b>15. NUMBER OF PAGES</b><br>107                          |  |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                 |                                                                | <b>16. PRICE CODE</b>                                      |  |
| <b>17. SECURITY CLASSIFICATION OF REPORT</b><br>Unclassified                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <b>18. SECURITY CLASSIFICATION OF THIS PAGE</b><br>Unclassified | <b>19. SECURITY CLASSIFICATION OF ABSTRACT</b><br>Unclassified | <b>20. LIMITATION OF ABSTRACT</b><br>UU                    |  |

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

**USING MACHINE LEARNING TO PREDICT THE INSIDER THREAT IN A  
NETWORK ENVIRONMENT**

Natasha K. Niemann  
Lieutenant, United States Navy  
BSBA, Auburn University, 2014

Raymond G. Blockmon  
Major, United States Army  
BS, Cameron University, 2009  
MA, Brandman University, 2012  
MA, Webster University, 2018

Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN NETWORK OPERATIONS AND TECHNOLOGY**

from the

**NAVAL POSTGRADUATE SCHOOL  
March 2021**

Approved by: Vinnie Monaco  
Advisor

Brian P. Wood  
Co-Advisor

Charles Timm  
Second Reader

Alex Bordetsky  
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

## **ABSTRACT**

In the past, cybersecurity professionals relied upon Security Event and Information Management systems to ingest network, server, and host logs to assist in detecting suspicious and malicious activity in the network. Detecting threat activities also included manually inspecting packet captures to glean clues of nefarious activity.

Our research involves machine learning. We developed a model that observes the packet headers' characteristics when a user accessed a remote file server. Data sets were introduced and host-server configurations were used to determine if our classification model was consistent in identifying file access behavior. We were able to predict and classify file access behavior, such as uploading, downloading, deleting, and moving files on a file server, based upon using headers. The results from deriving the classifications were similar when using different host-server configurations and files.

Our research demonstrated potential avenues to study file access behavior on an enterprise network. Information repositories like file servers, SharePoint, and online data hosting sites such as Dropbox present a surface threat for information theft. Classifying file access behavior with these online resources presents a valuable goal for cybersecurity.

THIS PAGE INTENTIONALLY LEFT BLANK

## TABLE OF CONTENTS

|             |                                                                               |           |
|-------------|-------------------------------------------------------------------------------|-----------|
| <b>I.</b>   | <b>INTRODUCTION.....</b>                                                      | <b>1</b>  |
| <b>A.</b>   | <b>MOTIVATION .....</b>                                                       | <b>2</b>  |
|             | 1. Major Raymond Blockmon, U.S. Army.....                                     | 2         |
|             | 2. Lieutenant Natasha Niemann, U.S. Navy .....                                | 3         |
| <b>B.</b>   | <b>TRADITIONAL VERSUS MACHINE LEARNING.....</b>                               | <b>4</b>  |
| <b>C.</b>   | <b>SCOPE AND OBJECTIVE .....</b>                                              | <b>5</b>  |
| <b>D.</b>   | <b>THESIS STRUCTURE .....</b>                                                 | <b>5</b>  |
| <b>II.</b>  | <b>LITERATURE REVIEW .....</b>                                                | <b>7</b>  |
| <b>A.</b>   | <b>PROACTIVE CYBER DEFENSE WITH MACHINE<br/>LEARNING .....</b>                | <b>7</b>  |
| <b>B.</b>   | <b>PROFILING USER BEHAVIOR BASED ON HUMAN-<br/>MACHINE INTERACTION .....</b>  | <b>9</b>  |
| <b>C.</b>   | <b>PREDICTIVE ANALYSIS WITH PATTERNS OF RAW<br/>INPUT .....</b>               | <b>10</b> |
| <b>D.</b>   | <b>PERIPHERAL ENTRY DATA IN SUPPORT VECTOR<br/>MACHINE.....</b>               | <b>12</b> |
| <b>E.</b>   | <b>CAPTURING SYSTEM PROCESSES WITH<br/>CLASSIFICATION USING K-MEANS.....</b>  | <b>16</b> |
| <b>F.</b>   | <b>ANALYZING MALWARE WITH CUCKOO SANDBOX AND<br/>LEARNING WITH WEKA .....</b> | <b>26</b> |
| <b>G.</b>   | <b>CONCLUSION .....</b>                                                       | <b>29</b> |
| <b>III.</b> | <b>EXPERIMENTAL DESIGN .....</b>                                              | <b>31</b> |
| <b>A.</b>   | <b>PHASE 1: CREATION OF THE TESTING ENVIRONMENT.....</b>                      | <b>32</b> |
|             | 1. Network Configuration .....                                                | 33        |
| <b>B.</b>   | <b>PHASE 2: SMB TRAFFIC GENERATION AND PACKET<br/>CAPTURE .....</b>           | <b>35</b> |
|             | 1. Early Implementation of SMB Traffic Generation with<br>Python .....        | 35        |
|             | 2. Applying the Wireshark Filter .....                                        | 39        |
|             | 3. Matured Implementation of SMB Traffic Generation<br>with Python.....       | 40        |
|             | 4. Wireshark Packet Capture Output.....                                       | 43        |
|             | 5. Quantifying the Wireshark Packet Capture Data .....                        | 45        |
|             | 6. Preparing to Test for SMB Behavior Using Machine<br>Learning.....          | 47        |
| <b>C.</b>   | <b>PHASE 3: MODEL VALIDATION.....</b>                                         | <b>48</b> |

|            |                                                                    |           |
|------------|--------------------------------------------------------------------|-----------|
| <b>IV.</b> | <b>DATA ANALYSIS AND RESULTS .....</b>                             | <b>49</b> |
| <b>A.</b>  | <b>CLASSIFICATION METRICS .....</b>                                | <b>49</b> |
| <b>B.</b>  | <b>SMB CLASSIFICATIONS AND RESULTS .....</b>                       | <b>53</b> |
| 1.         | Upload SMB Operation.....                                          | 54        |
| 2.         | Download SMB Classification .....                                  | 55        |
| 3.         | Lateral Copy SMB Classification .....                              | 55        |
| 4.         | Delete SMB Classification .....                                    | 56        |
| 5.         | Overall SMB Feature Model Performance.....                         | 56        |
| 6.         | Classification Figure Results.....                                 | 58        |
| <b>V.</b>  | <b>CONCLUSION .....</b>                                            | <b>67</b> |
| <b>A.</b>  | <b>SIMULATING SMB NETWORK TRAFFIC .....</b>                        | <b>67</b> |
| 1.         | Limitations with the Classification Model with<br>Encryption ..... | 68        |
| 2.         | Classification usage in an Enterprise Environment .....            | 69        |
| <b>B.</b>  | <b>CONCLUSION .....</b>                                            | <b>70</b> |
|            | <b>INITIAL DISTRIBUTION LIST .....</b>                             | <b>87</b> |

## LIST OF FIGURES

|            |                                                                   |    |
|------------|-------------------------------------------------------------------|----|
| Figure 1.  | Snippet of a User's Command-Line History .....                    | 12 |
| Figure 2.  | One-Class SVM of User D .....                                     | 14 |
| Figure 3.  | Two-Class SVM of User D.....                                      | 15 |
| Figure 4.  | ROC Scores for Users A, B, C, & D Detected as Masqueraders.....   | 15 |
| Figure 5.  | Discrete KDE (left) versus Continuous KDE (right).....            | 19 |
| Figure 6.  | Thread Count (x-axis) versus Process ID (y-axis).....             | 21 |
| Figure 7.  | Centroid Graph.....                                               | 22 |
| Figure 8.  | Network Behavioral Profile Normal versus Abnormal Activity .....  | 23 |
| Figure 9.  | Port Behavioral Profile Normal versus Abnormal Activity.....      | 24 |
| Figure 10. | CPU Behavioral Profile Normal versus Abnormal Activity.....       | 24 |
| Figure 11. | Hard Drive Behavior Profile Normal versus Abnormal Activity ..... | 25 |
| Figure 12. | Overall Matured Experimental Design Flowchart.....                | 31 |
| Figure 13. | Logical Network Diagram of Experimental Environment.....          | 32 |
| Figure 14. | Actual Experimental Equipment Used.....                           | 33 |
| Figure 15. | Host and IP Diagram.....                                          | 33 |
| Figure 16. | SMB Drive Attached on the NUC .....                               | 35 |
| Figure 17. | Random File Opener <i>Fileopenerrandom.py</i> Flowchart .....     | 38 |
| Figure 18. | Wireshark Filter Settings .....                                   | 39 |
| Figure 19. | Output from <i>fileopenerrandom.py</i> .....                      | 40 |
| Figure 20. | <i>SMBGenerator.py</i> High-Level Flowchart.....                  | 41 |
| Figure 21. | Single SMB Operation Flowchart.....                               | 41 |
| Figure 22. | <i>SMBGenerator.py</i> Sample Output .....                        | 42 |
| Figure 23. | Wireshark Packet Capture Sample in CSV Format .....               | 43 |

|            |                                                            |    |
|------------|------------------------------------------------------------|----|
| Figure 24. | Last Close SMB Command and Keep-Alive Packets.....         | 45 |
| Figure 25. | feExtractionSMBCount.py High-Level Process.....            | 47 |
| Figure 26. | feExtractionSMBCount.py Sample Output .....                | 47 |
| Figure 27. | Confusion Matrix .....                                     | 50 |
| Figure 28. | Recall Formula.....                                        | 51 |
| Figure 29. | Precision formula .....                                    | 51 |
| Figure 30. | f1-score .....                                             | 51 |
| Figure 31. | Windows 10 Massive Upload Chart and Histogram.....         | 58 |
| Figure 32. | Server 2019 Massive Upload Chart and Histogram .....       | 59 |
| Figure 33. | Windows 10 Massive Download Chart and Histogram.....       | 60 |
| Figure 34. | Server 2019 Massive Download Chart and Histogram.....      | 61 |
| Figure 35. | Windows 10 Massive Lateral Copy Chart and Histogram .....  | 62 |
| Figure 36. | Server 2019 Massive Lateral Copy Chart and Histogram ..... | 63 |
| Figure 37. | Windows 10 Massive Delete Chart and Histogram.....         | 64 |
| Figure 38. | Server 2019 Massive Delete Chart and Histogram.....        | 65 |
| Figure 39. | Windows 10 Overall SMB Chart.....                          | 66 |
| Figure 40. | Server 2019 Overall SMB Chart.....                         | 66 |

## LIST OF TABLES

|          |                                                      |    |
|----------|------------------------------------------------------|----|
| Table 1. | Classification Results before Feature Selection..... | 27 |
| Table 2. | Classification Results after Feature Selection.....  | 28 |
| Table 3. | Windows 10 and Rasp1 File Types .....                | 36 |
| Table 4. | Server 2019 and Rasp2 File Types .....               | 37 |
| Table 5. | Regression Analysis Fields .....                     | 44 |

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF ACRONYMS AND ABBREVIATIONS

|         |                                                                  |
|---------|------------------------------------------------------------------|
| .acsm   | Adobe Content Server Message file extension                      |
| .csv    | Comma Separated Value file extension                             |
| .doc    | Document file extension                                          |
| .docx   | Document Extensible Markup Language file extension               |
| .gif    | Graphics Interchange Format file extension                       |
| .html   | HyperText Markup Language file extension                         |
| .jpg    | Joint Photographic Group file extension                          |
| .mod    | Module file format extension                                     |
| .mp4    | Moving Pictures Expert Group-14 file extension                   |
| .mw     | Text document file extension                                     |
| .odt    | OpenDocument Format file extension                               |
| .one    | OneNote file extension                                           |
| .pcap   | Wireshark Packet Capture file extension                          |
| .pdf    | Portable Document Format file extension                          |
| .png    | Portable Networks Graphic file extension                         |
| .ppt    | PowerPoint file extension                                        |
| .pptx   | PowerPoint Open Extensible Markup Language file extension        |
| .py     | Python file extension                                            |
| .txt    | Text document file extension                                     |
| .xls    | Excel file extension                                             |
| .xlsx   | Excel Open Extensible Markup Language Spreadsheet file extension |
| .xml    | Extensible Markup Language file extension                        |
| ACK/Ack | Acknowledgment flag in TCP                                       |
| AIML    | Artificial Intelligence Machine Language                         |
| ANSI    | American National Standard Institute                             |
| API     | Application Processing Interface                                 |
| ARFF    | Attributed Relation File Format                                  |
| Avg     | Average                                                          |
| BIOS    | Basic Input/Output System                                        |
| BS      | Bachelor of Science                                              |
| BSBA    | Bachelor of Arts & Bachelor of Science                           |
| CIFS    | Common Internet File System                                      |
| CLI     | Command Line Interface                                           |
| CMD     | Command                                                          |
| CPT     | Cyber Protection Team                                            |

|          |                                                |
|----------|------------------------------------------------|
| CPU      | Central Processing Unit                        |
| CSIS     | Center for Strategic and International Studies |
| CSV      | Comma Separated Value                          |
| DCO      | Defensive Cyber Operations                     |
| DNA      | Deoxyribonucleic Acid                          |
| DOD      | Department of Defense                          |
| DoDIN    | Department of Defense Information Network      |
| DoS      | Denial-of-service                              |
| Dst      | Destination                                    |
| FIN      | Finished flag in TCP                           |
| FN       | False Negative                                 |
| FP       | False Positive                                 |
| FPR      | False Positive Rate                            |
| Ft       | Fort                                           |
| FTP      | File Transfer Protocol                         |
| GB       | Gigabytes                                      |
| GetInfo  | Get Information                                |
| G.O.A.T. | Greatest of All Time                           |
| GUI      | Graphical User Interface                       |
| HTML     | HyperText Markup Language                      |
| HTTPS    | HyperText Transfer Protocol Secure             |
| ID       | Identification                                 |
| IDS      | Intrusion Detection Systems                    |
| Info     | Information                                    |
| IOTCL    | Input/Output Control                           |
| IP       | Internet Protocol                              |
| IPS      | Intrusion Prevention System                    |
| JSON     | JavaScript Object Notation                     |
| KB       | Kilobyte                                       |
| KDE      | Kernel Density Estimation                      |
| KNN      | K-nearest neighbors                            |
| LAN      | Local Area Network                             |
| Len      | Length                                         |
| LMT      | Logistic Model Trees                           |

|         |                                                          |
|---------|----------------------------------------------------------|
| LT      | Lieutenant (USN)                                         |
| MA      | Master of Arts                                           |
| MAJ     | Major (USA)                                              |
| MB      | Megabyte                                                 |
| MD5     | Message-Digest 5                                         |
| ML      | Machine Learning                                         |
| N6      | Communications and Information Systems Directorate (USN) |
| NBSS    | NetBIOS Session Service                                  |
| NETCOM  | Network Technology and Enterprise Command (USA)          |
| NIC     | Network Interface Card                                   |
| No.     | Number                                                   |
| No_Op   | No Operation                                             |
| NOC     | Network Operations Center                                |
| NSN     | National Stock Number                                    |
| NUC     | Next Unit of Computing                                   |
| OS      | Operating System                                         |
| PCAP    | Packet Capture                                           |
| PDF     | Portable Document Format                                 |
| PySMB   | Python Server Message Block                              |
| RASP    | Raspberry Pi                                             |
| Rev     | Revision                                                 |
| ROC     | Receiver Operating Characteristic                        |
| SD      | Secure Digital                                           |
| Seq     | Sequence                                                 |
| SHA1    | Secure Hash Algorithm 1                                  |
| SIEM    | Security Information and Event Management                |
| SLE     | Single Left Edge                                         |
| SMB     | Server Message Block                                     |
| SMB2    | Server Message Block Version 2                           |
| SMB3    | Server Message Block Version 3                           |
| SMB CMD | Server Message Block Command                             |
| Src     | Source                                                   |
| Std     | Standard                                                 |
| SVM     | Support Vector Machine                                   |
| SYN/Syn | Synchronize flag in TCP                                  |
| TCP     | Transmission Control Protocol                            |

|      |                                            |
|------|--------------------------------------------|
| TLS  | Transport Layer Security                   |
| TN   | True Negative                              |
| TP   | True Positive                              |
| TPR  | True Positive Rate                         |
| TRAC | The Research and Analysis Center           |
|      |                                            |
| UDP  | User Datagram Protocol                     |
| U.S. | United States                              |
| USA  | United States Army                         |
| USB  | Universal Serial Bus                       |
| USN  | United States Navy                         |
| UTC  | Coordinated Universal Time                 |
| UU   | Unclassified                               |
| UW   | University of Wisconsin                    |
|      |                                            |
| WEKA | Waikato Environment for Knowledge Analysis |
| Win  | Window                                     |

## ACKNOWLEDGMENTS

To my mother and father, Olga and Paul, thank you for the love and unwavering support through my years while serving in the military. I also want to thank my children, Raeleah, Ray Jr., Savion, and Samarea for being the most fantastic kids; you guys keep me on my toes.

I want to thank Dr. Vincent Monaco, Mr. Brian Wood, and MAJ Charles Timm for their dedicated support and mentorship. At times I felt like I took on a project beyond my technical abilities; however, I never felt alone during this strenuous time.

Last but not least, I would like to give my sincerest gratitude to LT Natasha Niemann. I was under immense pressure, and it was a godsend that you jumped in both feet first to take on the lion's share of the work. We made a great team. I hope one day, we get to do it again!

—MAJ Raymond Blockmon, United States Army

Firstly, I want to thank my family for always pushing me to become the best version of myself and to achieve my goals no matter what the degree of difficulty. Even through the rollercoaster of my career, I can always count on my parents and sister to be my biggest cheerleaders.

I express my gratitude to Dr. Vincent Monaco, Mr. Brian Wood, and MAJ Charles Timm for believing in Ray and me and our abilities to complete this thesis. You have been great mentors throughout this process and pushed both Ray and me out of our comfort zones.

To my fiancé Bill, we could not have done this without you! You provided us data for the project! You also gave me constant encouragement through the highs and lows. You held me accountable to get things done, and I am so thankful for the most supportive partner.

Finally, to my thesis partner MAJ Raymond Blockmon, you are the G.O.A.T. Thank you from the bottom of my heart for letting me join you in this endeavor. I have learned A LOT by writing this thesis with you. I hope to work with you again; the Army is lucky to have you!

—LT Natasha Niemann, United States Navy

## I. INTRODUCTION

Many large-scale organizations enjoy an enterprise network's benefits that allow for data collaboration, storage, and processing. These networks are often geographically separated where a user in Miami, Florida, will need to access a file server based in Los Angeles, California. People in these organizations may not be familiar with others who work remotely, fostering insider threat activity on the network. These organizations look towards cybersecurity professionals to prevent, detect, and react to such threats. Cybersecurity professionals are heavily relied upon to set up network sensors, enable auditing logs, and monitor intrusion detection systems to detect hackers from the outside the network and those on the inside who pose a significant threat to the organization.

In 2019, the Center for Strategic and International Studies (CSIS) published a paper that found 82% of their employers in eight countries did not have sufficient cybersecurity personnel to defend their organization's network (Crumpler & Lewis, 2019). CSIS also determined that 71% of the employers said that cybersecurity professionals' shortage directly and negatively impacted the organization (Crumpler & Lewis, 2019).

Cybersecurity professionals are in high demand and low supply. These network defenders have the heavy task of managing an enterprise. Security information and event management (SIEM) applications such as Splunk only present these professionals with known data trends. The data trends only show what has happened previously. This leaves the defenders to react more to cybersecurity events than on the proactive approach to preventing cyber-related events from occurring. These professionals rely on many different data types and can be overwhelming when finding clues to who and what is occurring.

The enterprise network can send and receive many different data formats such as full-motion video, email, chat, word documents, and voice of internet protocol. Cybersecurity professionals must understand everything that is happening on the network to determine if it is experiencing malicious activity or not. Therefore, these cybersecurity professionals spend most of their time digging and sifting through network traffic to come up with answers to what is happening.

Network traffic is a compilation of encapsulated data that contains valuable information such as the payload and window size, the transport control protocol length, and network time, which is essential for encryption and authentication methods. Additionally, network traffic carries data from host to server and vice versa. The traffic can contain data about files, usernames, passwords, and commands given for File Transport Protocol and Telnet applications.

Network traffic can also provide clues to what a malicious user may do based on network patterns and highlighting statistical anomalies. The problem with identifying malicious user behavior is that most organizations use reactive methods. The reactive nature gives the malicious user enough time to further expand in the network, establish backdoors, install malware, and exfiltrate sensitive data and files before being caught or raising awareness to the network defenders. This is important because it leaves private industries, government agencies, and military organizations constant threat of their network and information systems being vulnerable—our drive to complete this research successfully by enhancing cybersecurity professionals using machine learning

## **A. MOTIVATION**

### **1. Major Raymond Blockmon, U.S. Army**

While assigned to the U.S. Army Cyber Protection Brigade, we understood the need for statistical anomaly detection to assist the Cyber Protection Teams (CPT) to hunt and disrupt adversary activity on neutral and friendly networks. This may not sound labor-intensive or complicated, but it is very much so.

A few years ago, during a training exercise event, I led a small team of cyber operators defending a network during an exercise event. Our training environment consisted of several file and domain controller servers and fictitious user accounts. The red team, which signifies as the adversary, already had backdoors and other avenues of approach to re-enter the network established if we used a method that prevented them from doing so. The actions conducted by the adversary were overwhelming when considering that I had tasked the team members with individual roles and responsibilities like monitoring a network, server, and host logs, managing group policy, and implementing

firewall rules. Of course, no good plan survives the first contact with the enemy. We experienced numerous denial-of-service events and tampering of user accounts.

What became glaringly obvious was that our method to understand and predict the adversary would be futile because we attempted to do this manually. This consequently caused workforce to be pulled away from other critical tasks. We were relying on trying to catch the adversary. The adversary was continuously exploiting the capability gap I faced. The adversary kept my team on the ropes by distracting us and moving our attention away from their activity, while the adversary's real objective was to move sensitive files off to a rogue file server quietly.

I realized that there must be a better and more efficient way to defend the network, using the system and network data. Locard's principle<sup>1</sup> is most prevalent in cyber: every action committed by an individual, whether a legitimate or a malicious user, will have their presence recorded through the system and other appliances that record traffic. It is nearly impossible to infiltrate a network without leaving a clean trace behind.

After my time at the Cyber Protection Brigade ended, I wanted to give back to the Army something that can be used in the future and increase its capability to defend networks more vigorously and efficiently. This has led me to exploit the incredible capabilities that machine learning has to offer.

## **2. Lieutenant Natasha Niemann, U.S. Navy**

I have always been interested in Defensive Cyber Operations (DCO) since I was in college; this led me to pursue my degree in Information Systems Management. During my time with Commander Sixth Fleet's Staff in Italy, I was assigned as a DCO planner for the N6 directorate. I felt that many of the network defending strategies that were available for scenario planning to use in exercises were not efficient and that we had to rely on our CPTs. They do their job well but being overseas made things a bit complicated. As MAJ Blockmon mentioned, "that our method to understand and predict what the adversary was

---

<sup>1</sup> Dr. Edmond Locard (1877-1966), a pioneer in forensic science, said that every contact left by an individual leaves a trace of their presence behind. For example, holding a glass cup will leave traces of fingerprints and DNA from sweat on the glass.

going to do was futile.” I have also thought the same thing during my time in the military, with different network threats happening around my operations. We must keep our networks safe, and our project allows us to explore a capability gap within the Department of Defense (DOD).

## **B. TRADITIONAL VERSUS MACHINE LEARNING**

One of the many tools that cybersecurity professionals use is the security information and event management (SIEM). The SIEM ingests logs to develop trends from the network. These professionals also use native tools such as Microsoft PowerShell, Security Onion, and other tools to assist with threat hunting. Consequently enough, most of these tools are more reactive than proactive. Cybersecurity professionals must monitor and analyze network and system activity to identify potentially malicious activity and then further expand into an area that appears to be suspicious.

Using static analysis (where a cybersecurity analysis is sifting through network and system logs using SIEM or other means after the malicious or suspicious event occurred), the network traffic can help determine the file access behavior by analyzing the headers in the network packet. We can automate this process using Python to speed the analysis and inspect the network packet headers quickly.

This research focuses on finding ways to improve network security by identifying activities and behaviors of insider threats in the DOD through machine learning (ML). We established a test network to provide a baseline of regular user traffic. We then introduced simulated malicious traffic of user activity on the network. Subsequently, we compare the efficacy and suitability of various ML algorithms to see which yields the best results at identifying the malicious behavior from regular baseline traffic. Using ML and operational data, this research will draw upon a new technology that can address a capability gap, achieve proactive defense, and use an ML model to predict insider threat activities.

## **C. SCOPE AND OBJECTIVE**

Our research's scope includes developing an environment using Python to generate and analyze network traffic without external factors influencing the behavior. Doing so allowed us to classify and predict file access behavior by using the network traffic. This environment will enable us to use deductive reasoning without additional, unwanted influences that may change the network behavior.

The data will assist in generating the network traffic consisting of various files, types, and sizes. This provides us with a variety of work patterns to analyze the network traffic. Extracting the data from the network traffic allowed us to introduce it to our model, which allowed us to classify and predict the type of file access behavior that occurred.

This research's primary objective is to identify endangered machines by predicting user behavior through SMB operations analyzing network traffic with machine-learning algorithms. Our demonstrated approach may provide a low-cost solution to discover insider threats within a network.

Additionally, our research aims to assist cybersecurity professionals at an enterprise level with predicting malicious activity is more strenuous and difficult to achieve. This capability is a low-cost, high pay-off solution that can improve as more data becomes available to a machine-learning model. To classify file access behavior on a file server and identify if an insider threat who may want to steal, hide, or delete critical files, we must answer two questions:

Research Question 1: Can we determine unique classifications for each file access behavior using packet header data with machine learning?

Research Question 2: If we use the same machine learning model in two different client-server configurations using two different data sets, will the regression model determine similar classification characteristics?

## **D. THESIS STRUCTURE**

In Chapter I, we provided a general overview of how cybersecurity is currently is practiced in the operational environment through static analysis and SIEMs. These types

of methods are painstakingly slow and intense on resource consumption. The capability gap needed to expedite rapid and effective cybersecurity response is with machine learning.

In Chapter II, we researched different machine learning methods with different data sets on how to classify malicious and benign activity. We studied the literature review by observing several research teams on their approach and using machine learning to increase cybersecurity effectiveness. For example, how to classify user and system behavior based on keystroke and mouse patterns, application use and preference, and system processes behavior such as hard drive speed and memory utilization.

In Chapter III, we set up our lab using hardware and Microsoft products to establish our research environment. We then developed in-house software solutions by using Python to program and automate network traffic. We established the ground truth on accessing a file server by creating logs.

In Chapter IV, we then subjected our results from the data by extracting the network packets captured in Chapter III. We classified the results with our machine learning model. Our team identified the independent variables such as Packet Length and Transmission Control Protocol (TCP)Length. We derived our target features (dependent variables) such as Read, Write, Close, and Create. The results provided us with the classifications based on user file access behavior.

Finally, in Chapter V, we review our research findings, and we offered recommendations to continue research at an enterprise level and identified limitations with our research in Chapter V.

## **II. LITERATURE REVIEW**

A critical task in cyber defense is to identify and prevent insider threats from occurring. Depending on the network's size, the system logs must be sifted through by Security Information and Event Management (SIEM) systems. A SIEM is an enterprise application that ingests system logs from servers and network appliances, which then generate statistical trends and provide visualization to the data collected. A user can query the SIEM in the hopes of identifying anomalous activity that may or may not be apparent to the users, such as data exfiltration that leaks slowly to an unauthorized remote server.

Some organizations use proprietary software that supports their cybersecurity program. For example, using McAfee for an anti-virus solution and Tanium for a host end-point solution to inspect for anomalous or malicious activities on the system (Colbaugh & Glass, 2011). Regardless of the software available, these systems and networks present a significant problem to cyber defenders due to their increased size and complexity within organizations. Reacting to cyber threats and attacks inside a network is a daunting task; however, the post threat and attack reaction method are outdated as cyber defenders are now looking towards Machine Learning (ML) to proactively prevent these threats and attacks before they occur (Colbaugh & Glass, 2011).

### **A. PROACTIVE CYBER DEFENSE WITH MACHINE LEARNING**

Most network operations centers (NOCs) have introduced a SIEM solution to help cyber defenders monitor through logs. These logs are snapshots that record either completed or attempted user and system events on the network, such as: logging in or failing to log in to the network, using system tools such as Microsoft PowerShell, or creating, deleting, and modifying files and folders on systems. For the most part, every action is recorded by the network servers. If configured correctly, these event logs are stored to be retrieved later for review by administrators and cyber defenders. To date, most NOCs use some method to identify and prevent threats, such as implementing an Intrusion Prevention/Intrusion Detection System (IPS/IDS) solution (Masduki, Ramli, Saputra & Sugiarto, 2015). The IPS/IDS can inspect network traffic in real-time by matching patterns

of malicious IP (internet protocol) addresses attempting to access the local network. It can match hexadecimal values of files against its library of known malware (i.e., signatures) ruleset. If the IPS/IDS does receive an IP address that is known to be malicious based on its ruleset, the sensor will alert the administrators through the SIEM. The IPS/IDS may prevent the IP address from entering the local area network by dropping the IP packet if the sensor can do so (Fraley & Cannady, 2017).

ML offers a unique capability for cyber defenders because of its ability to process large quantities of data and then categorize and label the data based on classification (Fraley & Cannady, 2017). ML can detect future insider threats based on behavior patterns before the threat is realized (Fraley & Cannady, 2017). Some essential features that ML offers in comparison to SIEMs are conducting statistical analysis such as regression modeling that displays the relationship between variables, e.g., a graph that plots unknown IP addresses, how many times the unknown IP addressed contacted the network. It can also determine what intervals a rouge process is associated with an unknown IP address trying to leave and go outside of the local area network (LAN). Using the regression model, the unknown IP address's repeated connection attempts may indicate that it belongs to a command-and-control server, for instance. As this pattern develops, other workstations that experience the same network activity behavior even with different unknown IP addresses should warrant further investigation.

A SIEM displays simple statistical data. For instance, a SIEM will display how many times an external IP address is connected to the company's web server or how many times a user has failed to log into their account by entering the wrong password (Mayhew, Atighetchi, Adler, & Greenstadt, 2015). SIEMs such as ArcSight, MacAfee, and Splunk are limited in predictive analysis. By the time a cyber defender searches the SIEM for information, the insider threat may have executed their mission, leaving the organization in a panic. Some SIEMs are limited in capabilities, and some are not employed to their full potential. In either case, this leaves cybersecurity professionals less capable of defending their networks (Mayhew et al., 2015).

## **B. PROFILING USER BEHAVIOR BASED ON HUMAN-MACHINE INTERACTION**

Researchers at the University of Wisconsin (UW) were able to use ML to determine which users were indicative of system misuse or posed an insider threat (Madison, Shavlik, Shavlik, and Fahland, 2001). Constructing a testing environment and using their skills on a Windows 2000 server, the team was able to engineer a prototype anomaly-detection system that generated statistical results based on user actions such as keystrokes and network bandwidth usage (Shavlik et al., 2001). Furthermore, the team could determine a pattern of behavior mapped to each user on the network. For example, one user may use the mouse to navigate through the application window, while another user may use the keyboard to switch through different applications (Shavlik et al., 2001). The Shavlik team also conducted event log monitoring by capturing key registry locations such as crucial system files, login abnormalities, and changes to accounts that were deemed suspicious. The research team also incorporated security policies to alert when unauthorized users try to access restricted files.

Shavlik et al. (2001) then developed an algorithm that measured the input from the user and baselined activity on mouse movement and keystrokes mapped to their profile. When a malicious user accessed a victim host, the team hypothesized that the algorithm would detect a change in keyboard and mouse input activity that does not fit within the target victim's behavior profile. This would signal an alert that the 'owner' is not operating their host (Shavlik et al., 2001). The UW team trained the algorithm to monitor each user and capture 50,000 keystrokes equated to two weeks' worth of activity. The team also captured another 50,000 keystrokes from each user to train their algorithm, followed by another 30,000 keystrokes to serve as test data. This test data is independent of the training data and allows for an unbiased evaluation. The training data validated the algorithm that should produce the expected results. The model will then use the test data (Shavlik et al., 2001).

The Shavlik et al. (2001) algorithm worked on computing the time taken between each keystroke and the amount of time it is held down. Their model then determined that if the keystroke time is less than a specified threshold of the user's profile, then it would

be considered the legitimate user who is typing on the host (Shavlik et al., 2001). If more keystrokes entered than the window width of keystrokes allowed, an alert would be raised, signifying that the user is malicious. Shavlik et al. (2001) saw that they could identify a substantial number of intrusions detected by their algorithm. The team defined intrusions as user *X* typing on user *Y*'s host with a shallow false-positive rate based on their algorithm. For example, user *Y* would open a file called 'resume.txt' in the terminal using the Vim text editor. User *X* would simply enter 'cat resume.txt' in the terminal to display the contents directly to the screen without using Vim, which is much faster and efficient than *Y*'s approach by using fewer keystrokes. This demonstrates two different approaches that achieve the same outcome of reading the contents of a file. The Shavlik team conceded that even with a low keystroke. The insider threat has the potential to cause significant damage. For example, a single line of code in a script can unpack malware and execute on a host (Shavlik et al., 2001).

The next characteristic that users demonstrated in their daily application usage habits. For example, a user may come to the office logon to their host. At this point, a user may open their email client to read. This may be a standard operating procedure for some workforce users (Davidson & Hirsh, 1998). Over time, these habits can establish a user profile on the network based on the data. For instance, a user who did not access a sensitive file or directory before now appears to do so. This presents a problem because users may now have access to information that they are not authorized to have before.

### **C. PREDICTIVE ANALYSIS WITH PATTERNS OF RAW INPUT**

Davidson and Hirsh (1998) at Rutgers University conducted research using neural networks<sup>2</sup> on command-line data generated by the user who interacted with the terminal (McKinney, 2008). This research profiled how certain users interacted with their Unix host and their applications to determine a user profile (McKinney, 2008). Their effort was

---

<sup>2</sup> "An artificial neural network learning algorithm, or neural network, or just neural net, is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form" (DeepAI, n.d.).

primarily driven to understand any given user's behavioral patterns over an amount of time to develop their pattern of behavior (McKinney, 2008).

In the Davison and Hirsh (1998) study, users entered over 2,000 command instances in their Unix hosts, with 77 distinct commands used. Davidson and Hirsch (1998) hypothesized that people tend to repeat daily habits and are repetitious in the workplace. They found the command-line histories stored in memory on the user's hosts showed repeated commands. For example, one user may prefer the 'cat' command to view a file's contents displayed to the terminal directly. In contrast, another user prefers opening the same file in a text editor such as Vim. The team observed 77 users within a period of two to six months. They collected over 168,000 commands executed on Unix terminals. The research population consisted of 70 undergraduate students in the Internet programming course, and the remaining seven were graduate students and faculty (Davison & Hirsh, 1998).

The team found a macro-average<sup>3</sup> of 8.4% of the commands that were used were new and not logged previously. The researchers noted that the micro-average<sup>4</sup> of new commands consisted of 3.6%, meaning "...the fact that smaller samples had larger numbers of unique commands" (Davison & Hirsh, 1998, p.2). Finally, Davison and Hirsh (1998) discovered that one out of five commands entered occurred twice by each user. Figure 1 displays a snippet of a user's command-line history; the 'BLANK' command marks a new session started by the same user. The user entered 'cd' twice in the command line, which is the command to change to a different location on the file system's directory. The user also started a new session later and entered 'emacs', a text editor similar to Vim. As mentioned previously, the user entered the same command twice and as predicted, may use the 'cd' and 'emacs' command in the future.

---

<sup>3</sup> "Macro-averaged results compute statistics separately for each user, and then averages these statistics overall users." (Davidson & Hirsch, 1998, p.2).

<sup>4</sup> "Micro-averaged results compute an average over all data, determining the number of correct predictions made across all users divided by the total number of commands for all users combined." (Davidson & Hirsch, 1998, p.2).

```
...
96102513:34:49 cd
96102513:34:49 ls
96102513:34:49 emacs
96102513:34:49 exit
96102513:35:32 BLANK
96102513:35:32 cd
96102513:35:32 cd
96102513:35:32 rlogin
96102513:35:32 exit
96102514:25:46 BLANK
96102514:25:46 cd
96102514:25:46 telnet
96102514:25:46 ps
96102514:25:46 kill
96102514:25:46 emasc
96102514:25:46 emacs
96102514:25:46 cp
96102514:25:46 emacs
...
```

Figure 1. Snippet of a User's Command-Line History

#### **D. PERIPHERAL ENTRY DATA IN SUPPORT VECTOR MACHINE**

Researchers in India at the University of Hyderabad conducted a study on observing users masquerading or conducting impersonation. The user's goal was to gain access to confidential data or ultimately escalate user privileges on the network while conducting themselves as legitimate users (Bhukya, Kommuru, & Negi, 2007).

The team determined that any user over time would establish predictive actions by interacting with their host. In one example, users may prefer to interact with their host using the command-line to make changes to their files in the directory, while another user prefers using the desktop graphical user interface (GUI) to access their files (Bhukya et al., 2007). The research team hypothesized that when an individual interacts with a host and their behavior does not fit the behavior profile of the owner of the host, it would indicate that the user is a masquerader (Bhukya et al., 2007).

The Bhukya team observed eight different users with their in-house developed GUI event collector and discovered that the collector could efficiently detect masquerading users. Bhukya et al. (2007) classified a legitimate session of a user as a "positive" classification and a masquerade session by an illegitimate user as a "negative" classification. The data generated from the "positive" session would be used to develop the

One-class Support Vector Machine (SVM) (Bhukya et al., 2007). SVM is a supervised training model in the ML realm that analyzes data for classification and regression (SciKit Learn, 2019). Classification in ML is “the process of predicting the class of given data points” (Asiri, 2018, p. 2). In SVM, regression provides flexibility in how much acceptable error is in a model and then determines the hyperplane to fit that data (Sharp, 2020).

Bhukya et al. (2007) used legitimate sessions to develop their model's training data as their One-class SVM. The illegitimate data that the masquerading user-generated became the test data. In the One-class SVM, only one classification determined what regular activity looked like, such as a legitimate user and the session they interacted with their host. When another classification became available (i.e., a masquerading individual), it was compared against the regular activity. The results showed a distinct difference between the two classifications by illustrating the outlier (i.e., a masquerader's activity). Additionally, the team determined each user would develop a set number of his/her sessions as positive data. The remaining sessions of each user were then used to test against legitimate users as negative data (Bhukya et al., 2007).

The Bhukya research team used detection systems that collected the session data from the users to develop the behavioral profile. As a result, the team built individual behavioral profiles (Bhukya et al., 2007). After training their detection systems with the legitimate data, the team was able first to decipher regular activity and then later identify malicious activity when illegitimate data was introduced to it (Bhukya et al., 2007). Bhukya et al. (2007) used a method in Unix to leverage the ‘*acct*’ utility, which records users' commands in the terminal. Additionally, the team captured Windows Operating System (OS) logs to establish profiles. During their study, Bhukya et al. (2007) discovered that little research was conducted on observing Unix and the variant flavors of Linux that capture the user's interaction with the GUI desktop environment.

At the time of their study, Bhukya et al. (2007) observed the number of mouse clicks, the mouse movement's coordinates, and the keys pressed during the user's session. Since there was no logging method to collect their GUI events when the team conducted their research, they developed a custom logging tool to collect the events in-house (Bhukya et al., 2007).

Bhukya et al. (2007) observed the mouse enter and exit data recorded by the number of times the pointer entered and exited the application window, the average number of keys entered, and the average number of keyboard shortcuts entered. Bhukya et al. (2007) then assigned values to the following parameters to develop the standard deviation and the mean of each user. Mouse Clicks consisted of left-clicking, right-click, and double-click equaled 3, mouse entering and exiting an application equaled 2, wheel rotation horizontal and vertical movement equaled 2, keys pressed equaled 1, and keyboard shortcuts equaled 1. This data was applied to a sliding window<sup>5</sup> technique to generate tuples to serve as input values into their SVM (Bhukya et al., 2007).

After capturing the unique data of each user, their session data were split up. For example, User D created eight sessions, and four of those are used for training to develop a profile as a legitimate user. The other four sessions created the test data as a masquerader. (Bhukya et al., 2007). The team discovered that the average hit rate (i.e., detection rate) was 94.88% for User D, who acted as a masquerader on another user's host. In other words, out of 100 sessions where User D acted as a masquerader, 94 of those sessions would be detected by a cyber defender using the research team's One-class SVM method. In comparison, only 6 out of those 94 sessions would be undetectable by cyber defenders. Figure 2 depicts the results for User D in the One-class SVM.

Result of One-class SVM for User-D

| User    | Hits (%) | False Positive Rates (FPR) |
|---------|----------|----------------------------|
| User E  | 97.24%   | 2.75%                      |
| User F  | 95.00%   | 5.00%                      |
| User G  | 94.87%   | 5.12%                      |
| User H  | 92.42%   | 7.8%                       |
| Average | 94.88%   | 5.16%                      |

Figure 2. One-Class SVM of User D

---

<sup>5</sup> Sliding window algorithm is where the "data analysis is limited to an interval over the entire set of collected data, and, periodically, newly produced inputs are appended to the window and older inputs are discarded from it as they become less relevant to the analysis" (Bhatotia et al., 2014).

Bhukya et al. (2007) then compared the same users using the Two-class SVM approach. A Two-class SVM is where two classifications are trained and then compared together. Bhukya et al. (2007) trained both the positive and negative sessions from the users and then compared them with the One-class SVM method results. The findings showed less success in detection, averaging 53.08% compared to the One-class SVM, which resulted in 94.88% detecting the masquerader (Bhukya et al., 2007). Figure 3 displays the results of User D in the Two-class SVM method. Out of 100 sessions, User D would have been detected in 53 of those sessions. This means that User D would be detected if masquerading in 57 out of 100 sessions. The high detection rate posed a significant risk to cyber defenders because the reality of having a large percentage of not detecting a malicious user could lead to devastating results on the network. For example, data theft, manipulation, and administrator account would be compromised. The data suggests low confidence in detecting the malicious user.

Result of Two-Class SVM for User-D

| User    | Hits (%) | FPR    |
|---------|----------|--------|
| User E  | 56.86%   | 8.49   |
| User F  | 52.48%   | 0.00%  |
| User G  | 60.38%   | 0.00%  |
| User H  | 42.61%   | 28.87% |
| Average | 53.08%   | 9.34%  |

Figure 3. Two-Class SVM of User D

| User | User A | User B | User C | User D | Average |
|------|--------|--------|--------|--------|---------|
| A    | -      | 0.7190 | 0.4890 | 0.5010 | 0.5696  |
| B    | 0.7192 | -      | 0.9051 | 0.8624 | 0.8295  |
| C    | 0.4890 | 0.9051 | -      | 0.2665 | 0.5535  |
| D    | 0.5010 | 0.8643 | 0.2665 | -      | 0.5439  |

Figure 4. ROC Scores for Users A, B, C, & D Detected as Masqueraders

The Bhukya team noted that using the Two-class SVM method detected a wide range of probability of the ROC curve, with hit rates ranging from 26.64% to 90.51%; thus, deeming it inferior to the One-class SVM method because of the inconsistencies. Figure 4 displays the ROC scores for Users A, B, C, and D. The data displayed range from 26% to 90% when detecting the masquerading user. Unlike the ROC scores, the One-class SVM recorded by users E, F, G, and H scored 90% or higher across the board in detecting the masquerader and establishing consistency. The ROC scores of A, B, C, and D were sporadic, inconsistent, and would not help cyber defenders detect and prevent insider threats.

Bhukya et al. (2007) concluded that the One-class SVM was better equipped in analyzing user behavior and detecting individuals who were masquerading. They noted that using the One-class SVM method was efficient in training the model and developing the test data to detect users who were masquerading with it. Finally, the team determined that analyzing the keyboard and mouse movements on a GUI-based system effectively detected users who were masquerading because of the unique activities that each user demonstrated. Bhukya et al. (2007) noted that there was no need to train two classes because only using the one class approach was sufficient to detect the legitimate user. Any profile that falls outside of the behavioral profile is an illegitimate session, indicating the user is a masquerader.

#### **E. CAPTURING SYSTEM PROCESSES WITH CLASSIFICATION USING K-MEANS**

Akaninyene Walter Udoeyop submitted his thesis at the University of Tennessee, Knoxville, in 2010 with the goal of profiling users' activity and deriving their behavioral profile. Udoeyop approached the study in three phases: data acquisition, leveraging unsupervised data cluster analysis and using the KDE method (2010). Udoeyop would then use the results from the data to identify abnormal patterns of behavior that may indicate insider threat activity. Udoeyop classified the insider threat activity as either 1) a user with legitimate access violating security policy (e.g., inserting an unauthorized Universal Serial Bus (USB) device into a company-owned computer) or 2) a user violating access control

policy through unauthorized access (e.g., accessing someone's files without their consent or knowledge).

Udoeyop's research begins with acquiring data that a user-generated when interacting with their host. The data consisted of logs from system processes, hardware performance, file system manipulation, and network traffic logs (2010). This data was analyzed using the K-means (Garbade, 2018) algorithm to "learn" a user's normal behavior.

Udoeyop's (2010) thesis observed methods and actions before the insider threat could achieve their goal. For example, Udoeyop (2010) provided a possible scenario of an insider threat where a user typically displayed a behavior by only working with a given set of files and directories congruent with their user behavior profile. The same user observed copying files into another directory that was not accessed before. This activity may indicate suspicious or malicious activity and may lead the user to violate company policies.

Udoeyop (2010) classified behavior as abnormal or normal; both abnormal and normal behavior can experience threatening and non-threatening activities. Udoeyop (2010) informed his test subjects that they were subjected to monitoring with software installed and running in the background on their hosts. Udoeyop (2010) conceded that the test subjects were notified due to legal and privacy matters and, in doing so, which may skew the results of his research. Suppose a user is made aware that they are being monitored on the network. In that case, they may not engage in their everyday activities that may violate company policy, such as posting personal views about political issues on social media or shopping online while on company time.

The Microsoft .NET framework was used to collect data in Udoeyop's (2010) research. .NET supports a range of functions such as retrieving event logs and can capture the state of software processes and hardware performances (Udoeyop, 2010). Udoeyop used the .NET framework and the C# programming language to analyze the data that the users generated (2010). Udoeyop also collected processes by their identification number. These processes show which programs and services were running and what resources that they were consuming. Furthermore, when monitored over a given period, these processes

can develop a model and observe thread activities (Udoeyop, 2010). Threads are an executable sequence of codes associated with the operating system, and a process can spawn multiple threads (Udoeyop, 2010).

Udoeyop's research also examined and monitored both the central processing unit (CPU) and memory usage using the .NET framework (2010). Over time, this monitoring capability by .NET would provide Udoeyop a reliable baseline of hardware performance as he conducted his research. He also used the IPGlobalProperties in .NET to analyze the network traffic generated by the users. For example, .NET can provide which users accessed websites or connected to a file transport protocol (FTP) server (Udoeyop, 2010).

Finally, Udoeyop (2010) analyzed and mapped the file system behaviors to the user's activity profile. For example, the computer stores and organizes files and folders on the system. The user and system can create, delete, move, copy, and modify these system folders and files. Udoeyop (2010) used the FileSystemWatcher .NET class that includes subroutines and other functions to monitor system folder activities and data events. Udoeyop (2010) monitored the user's activities by observing the following:

- File Created, which creates a new file on the host.
- File Changed, which modifies or resaving a file to the host.
- File Renamed, that changes the name of the file on the host.
- File Delete, by removing an existing file from the host.

The date the file was accessed on, its absolute file path, and the file name were all recorded by the users to facilitate the user profile development (Udoeyop, 2010). Udoeyop noted that collecting this information over time can develop the user's profile and the host (2010).

Udoeyop (2010) developed a histogram (Figure 5) that was produced by the Kernel Density Estimation<sup>6</sup> (KDE), which separated the finite number of subintervals. Deciding to choose either the discrete KDE or continuous KDE method, Udoeyop (2010) chose the former because continuous KDE requires the probability of calculating a new data point. Even though

---

<sup>6</sup> "The Kernel Density Estimation is a mathematic process of finding an estimate probability density function of a random variable. The estimation attempts to infer characteristics of a population, based on a finite data set" (DeepAI, n.d.).

continuous KDE is more precise than the discrete method, it is more feasible and acceptable to utilize the discrete KDE method due to the amount of data accumulated during the research.

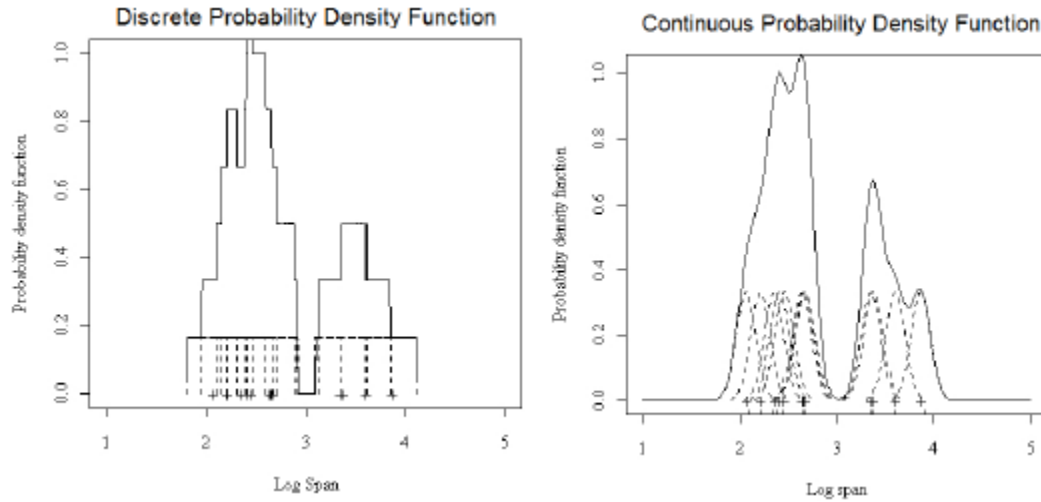


Figure 5. Discrete KDE (left) versus Continuous KDE (right)

The third phase included using Kernel Density Exploration (KDE) method to complete his behavioral analysis of users. The data stored in a structure called Data Grids that resides within the .NET framework.

Udoeyop (2010) performed his behavioral analysis on the users being observed based on the following variables that were generated with their hosts:

- The number of File System Events versus Directory whereas “the number of file creations, modifications, and deletions, along with the files that are renamed based on the directory” (Udoeyop, 2010, p. 41).
- Transport Control Protocol (TCP) Connections versus IP addresses whereas “the amount of TCP connections established from the local host to a remote machine” (Udoeyop, 2010, p. 41).
- The number of threads versus the parent process, whereas “the number of active threads running per process; each process is grouped by a parent process” (Udoeyop, 2010, p. 41).

- The number of handles versus processes, whereas “the number of active handles per process. A handle is a token, typically a pointer that enables a program to access a resource, such as a library function” (Udoeyop, 2010, p. 41).
- Hard drive usage, whereas “the ratio of bytes stored on the hard drive vs. the total number of bytes on the hard drive” (Udoeyop, 2010, p. 41).
- File Read Byte per second, whereas “the amount of memory that a read operation is performed on per second” (Udoeyop, 2010, p. 41).

In Figure 6, the data relation between the parent process ID on the x-axis concerning the threads generated by the parent process ID on the y-axis is displayed. For clarity, Microsoft defines processes and threads as “A process, in the simplest terms, is an executing program. One or more threads run in the context of the process. A thread is a basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread” (“Processes and Threads”, 2018).

In Figure 6, the parent process ID (Identification) 0 (zero), which is the idle system process, has spawned up to 70 threads, which can be expected based on the user’s activity. Suppose an application that is running on the host has increased inactivity. In that case, the process idle will drop in threads due to system resources such as the CPU processor being used to support the application. If the user is inactive on the host, the idle system process will rise. The idle system process is consuming host resources, which is normal behavior. Relating the parent process ID to the insider threat, each user may develop their process ID profile using the different applications they use frequently. For example, a user who has never used FTP will most likely not generate a process ID for it.

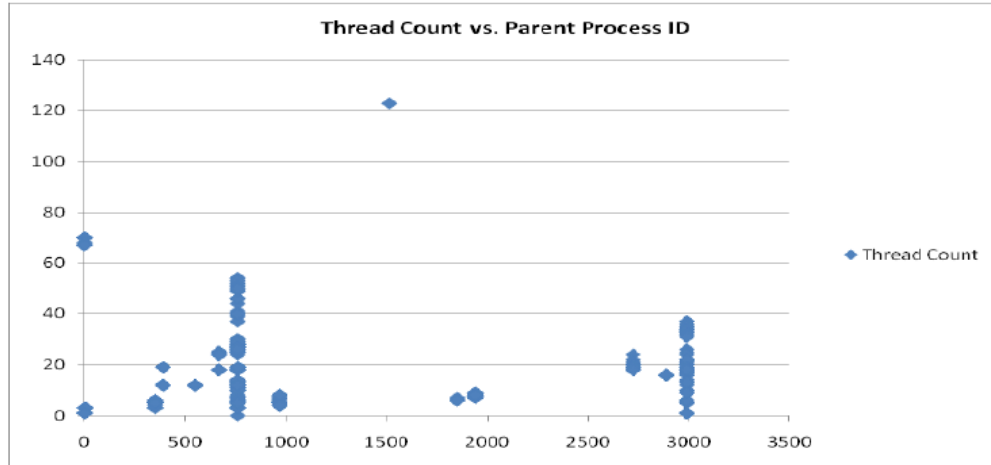


Figure 6. Thread Count (x-axis) versus Process ID (y-axis)

Clusters are data points grouped on a graph related to one another—for example, looking at home prices grouped by neighborhoods displayed on a city map. One could assume that by looking at the city map, one may identify where the low, middle, and upper-class neighborhoods based on the range of home values. We can relate this type of data visualization to cybersecurity reasonably well; we can observe which processes started and how many threads created from those processes. The processes with the most threads would have a denser and more populated cluster. Other processes are as much would display a dispersed cluster.

Figure 7, Graph 1 depicts data points clustered together. Udoeyop (2010) selected 60 processes he were utilized in his experiment and divided the group in half to come up with his initial 30 data points (60/2). These randomly selected data points served as his initial clusters, in which “K” represents the number of clusters (i.e.,  $K = 30$  in the K-means clustering algorithm).

The other 30 data points that were not converted to clusters were then measured and assigned to the nearest cluster. Once this process was completed, Udoeyop (2010) then measured and clustered the data points again using the mean value of the distance between the clusters. This process continued until the clusters no longer changed, which signified the convergence of the clusters. Udoeyop (2010) then identified the center of each of the 30 clusters to determine its centroids. A centroid is defined as either an actual or imaginary

point that lies at the center of a cluster (Garbade, 2018). Figure 7 shows the centroids on Graph 1 and a heat map in Graph 2 that displays the KDE statistical distribution. In other words, the KDE illustrates the density of the data points by using a heat map; the brighter the color, such as white, the denser the cluster is.

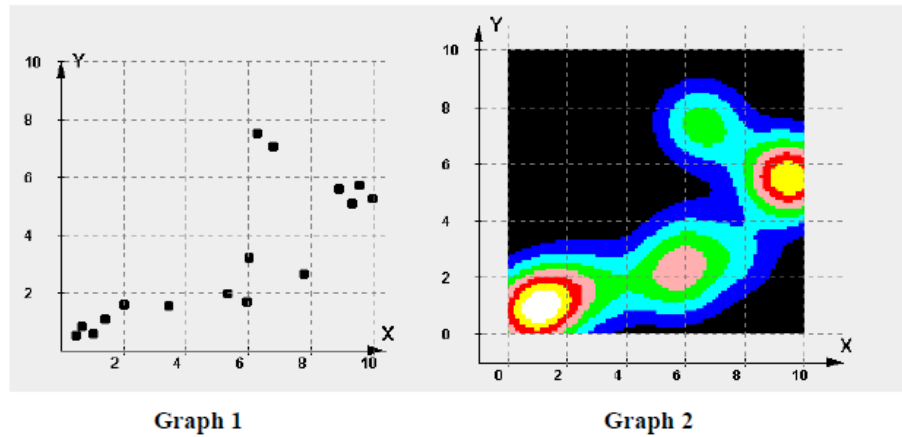


Figure 7. Centroid Graph

After developing the user behavior profile, Udoeyop (2010) used a thread bomb technique that spawns numerous threads of a system process, such as creating multiple instances of Notepad or Internet Explorer. This method draws on additional resources such as memory and processing power to run multiple instances of Notepad and Internet Explorer; thus, it creates more system process identifiers. Thread bombing (often referred to as fork bombing) is a malicious code that causes a denial-of-service (DoS) attack by consuming vast amounts of host resources such as memory. This prevents other legitimate processes and applications from running and denying or disrupting the user access to the system (Varmazyar, n.d.).

Udoeyop (2010) recorded an average user behavior probability from his user's thread usage pool at .631. Udoeyop (2010) then introduced numerous thread bombs to the host unbeknownst to the users to create abnormal activity. Udoeyop theorized that the thread bomb should create a significant behavior change but only dropped the thread usage slightly from .631 to .448. This showed a marginal change in behavior by 30%. Udoeyop

(2010) speculated that the thread bomb did not achieve the desired effect because the thread bomb only impacted one of the processes; all the other processes were unaffected.

One of the key findings in Udoeyop research showed that comparing the behavioral profile of the IP of a user's host to sites they visited frequently, or rarely visited, was difficult to analyze. This is because of the nature of how the network interface card (NIC) functions. Udoeyop (2010) stated that a network card does not know the difference between IP addresses and websites associated with it. Furthermore, to analyze the user's network activity's behavior, he analyzed the geographical IP locations, which he conceded was time and resource-consuming. The average IP behavioral profile before foreign websites was accessed was .353, and the average IP behavioral profile after foreign websites were accessed, such as NorthKorea.com, was .349 (Udoeyop, 2010). Figure 8 illustrates the difference between normal and abnormal network activity, depicting a difference of only 1% change in behavior, considering that the typical NIC in today's standard can transmit 1-Gigabit per second.

| <b>Network IP Profile</b>     | <b>Average Nominal Behavior Probaility</b> |
|-------------------------------|--------------------------------------------|
| Before Foreign Website Access | .353                                       |
| After Foreign Website Access  | .349                                       |

Figure 8. Network Behavioral Profile Normal versus Abnormal Activity

The next analysis conducted was the port behavioral profile that a user would be accustomed to accessing. Udoeyop (2010) discovered a significant change in port behavioral profile when users accessed foreign ports (see Figure 9). For example, most users access websites using port 80 and port 443 (unsecured and secured web browsing protocols, respectively); however, it is rare for users to access ports such as port 23, which is associated with Telnet. Telnet is an application used to access a host remotely using clear text credentials and is not widely used anymore because of the lack of security measures. It is used for the initial configuration of network devices. Udoeyop (201) found that the average profile was .894 and dropped to .104, equating to an 89% change in profile behavior when users connected to a foreign port. It is important to note that this effort only

considered accessing destination ports and not source ports. Source ports on a host are pseudo-randomly generated and allocated from 1024 to 65,535, making it nearly impossible to copy.

| <b>Network Port Profile</b>      | <b>Average Nominal Behavior Probability</b> |
|----------------------------------|---------------------------------------------|
| Before Access to Different Ports | 0.894                                       |
| After Access to Different Ports  | 0.104                                       |

Figure 9. Port Behavioral Profile Normal versus Abnormal Activity

The hard drive and the CPU processing profile behavior provided insightful information about the abnormal activity. Udoeyop (2010) found that the average processor behavior for a given user was .603. After the abnormal activity was introduced, the host recorded a significant drop in profile behavior to .258, resembling a 58% change in CPU behavior, as shown in Figure 10.

| <b>Process Thread Profile</b>  | <b>Average Nominal Behavior Probability</b> |
|--------------------------------|---------------------------------------------|
| Before New Program Utilization | 0.603                                       |
| After New Program Utilization  | 0.258                                       |

Figure 10. CPU Behavioral Profile Normal versus Abnormal Activity

Furthermore, Udoeyop (2010) also analyzed the behavioral profile's hard drive characteristics and found shocking results, illustrated in Figure 11. Udoeyop (2010) conducted two hard drive tests. The first test analyzed and determined average hard drive behavioral profile and then added 10 gigabytes (GB) of data to simulate abnormal behavior. Udoeyop (2010) found that the average behavioral profile during the baseline test introduced a score of .659 probability rate. Injecting 10GB of data dropped the average standard behavioral profile to .145, a 78% change in hard drive behavior (Udoeyop, 2010).

Udoeyop (2010) introduced and removed 1GB, 5GB, and 15GB of data onto the user's hard drives as a second test. Udoeyop (2010) observed that the results were similar to the first hard drive test, where the probability score of normal behavior dropped significantly. Each user having scored a .993 standard behavior profile, and after adding 1GB, the probability score dropped to .013, equating to a 99% change in hard drive behavior. Then, adding 5GB dropped to 0.00, and 15GB also dropped to 0.00, virtually a 100% change for both 5GB and 15GB behavior change. Udoeyop (2010) recorded that users generally added about 1-megabyte (MB) worth of data daily, so an addition of 1GB was significantly larger; thus, escaped the average standard behavior profile that one might predict.

| <b>Hard Drive Profile</b> | <b>Average Nominal Behavior Probability</b> |
|---------------------------|---------------------------------------------|
| <b>Before Adding 1 GB</b> | <b>0.993</b>                                |
| <b>After Adding 1 GB</b>  | <b>0.013</b>                                |

| <b>Processor Profile</b>         | <b>Average Nominal Behavior Probability</b> |
|----------------------------------|---------------------------------------------|
| <b>Before Increased Workload</b> | <b>0.659</b>                                |
| <b>After Increased Workload</b>  | <b>0.145</b>                                |

| <b>Hard Drive Profile</b> | <b>Average Nominal Behavior Probability</b> |
|---------------------------|---------------------------------------------|
| <b>Before Adding 5 GB</b> | <b>0.993</b>                                |
| <b>After Adding 5 GB</b>  | <b>0.000</b>                                |

Figure 11. Hard Drive Behavior Profile Normal versus Abnormal Activity

Udoeyop (2010) finally concluded that every test dropped the probability rate of profile behavior as theorized. It is important to note that the ports and hard drive profile behaviors displayed the most change in the behavior profile during Udoeyop's research (2010).

## **F. ANALYZING MALWARE WITH CUCKOO SANDBOX AND LEARNING WITH WEKA**

Arshi Dhammi and Maninder Singh, a research team at Thapar University in India, tackled the problem of proactively identifying malware based on static and dynamic analysis using the Cuckoo sandbox (2015). Dhammi and Singh (2015) approached the problem by using ML from the data that was generated from Cuckoo. The Cuckoo Sandbox is equipped with native applications built inside of it and open-source tools such as Wireshark, Process Explorer, and Regshot, which allow the user to observe what is occurring. For example, observing application programming interfaces (API) and binary strings that executed with the associated malware (Dhammi & Singh, 2015).

Dhammi and Singh (2015) collected 1,270 malware and benign files from online resources to serve as samples in their ML environment. The team also retrieved benign data from a clean Microsoft Windows 8.1 64-bit version after a fresh install (Dhammi & Singh, 2015). Furthermore, the clean Windows installation data includes formats such as Portable Executables, HyperText Markup Language (HTML) files, executables, compressed files, and much more (Dhammi & Singh, 2015). The output presented by Cuckoo can be delivered in multiple formats such as HTML, JSON, and MongoDB (Dhammi & Singh, 2015).

Dhammi and Singh (2015) defined static analysis of malware by observing the software without running it. This is accomplished by unpacking malware to view the source code and investigate how it functions when it executes. Dhammi and Singh (2015) leveraged a method developed by researchers at the University of Ballarat, Australia. The researchers in Australia developed a system that unpacks malware and automatically dissect, observe, and compare benign code against it.

Dhammi and Singh (2015) approached their research using six steps to determine malware activity:

1. Collect benign and malicious data.
2. Audit behavior activity.
3. Generate reports.

4. Process data from the reports.
5. Develop classification and learning using WEKA (Waikato Environment for Knowledge Analysis), an open-source machine learning and data mining platform.
6. Conduct validation of the results and verification.

The team then processed the data produced by Cuckoo in a Comma Separated Value (CSV) file (Dhammi & Singh, 2015). The team then converted the CSV file to an Attribute Relation File Format (ARFF), an acceptable format that the WEKA platform can ingest and conduct ML analysis. The ARFF was then fed into WEKA, which implemented a 10-fold cross-validation technique that can be used to conduct data mining tasks, which is illustrated in Figure 13.

Table 1. Classification Results before Feature Selection

| Classifier | TPR    | FPR   | Precision | Recall | Accuracy |
|------------|--------|-------|-----------|--------|----------|
| LMT        | 73.393 | 11.7  | 73.389    | 73.389 | 73.4289  |
| NaïveBayes | 68.2   | 16.9  | 68.19     | 68.20  | 68.2315  |
| SVM        | 71.18  | 14.31 | 71.04     | 71.04  | 71.0131  |
| Ridor      | 53.47  | 18.88 | 54.19     | 54.20  | 54.2692  |
| kNN        | 61.37  | 22.82 | 61.38     | 61.38  | 61.397   |

The classification consists of five columns (True Positive Rate, False Positive Rate, Precision, Recall, and Accuracy<sup>7</sup>) and five rows (Logistic Model Trees, Naïve Bayes, Support Vector Machine, Ridor, and K-nearest neighbors) (Dhammi & Singh, 2015). The research team noted that malware signatures such as SHA1 and MD5 hash (i.e., signature) algorithms were the least productive in their analysis. Though Dhammi and Singh did not

---

<sup>7</sup> The definitions for True Positive Rate, False Positive Rate, Precision, Recall, and Accuracy can found in Chapter IV, Figures 30 through 33.

explicitly state this in their research, any file can generate a new hash value by simply adding a commented-out period (.) or any other character in the source code.

The team highlighted that file types such as registry keys, resource names, and IP addresses ranked higher in priority concerning their analysis (Dhammi & Singh, 2015). Changing the attributes in these file types can disrupt the operating system and someone is attempting to change the system's functionality, which serves as an indicator of compromise. Dhammi and Singh (2015) discovered that the Logistic Model Trees was the most accurate at identifying malware at 98% because it demonstrated the ability to classify benign files and malware with a 7.7% false-positive rate (FPR). The low FPR equates to having cyber defenders spending less time determining if a file is malicious or not. This means when cyber defenders identify malware, they have a 98% chance of confirmation that it is malware that they were investigating. The team also observed that the recall, true positives, and precision were 98.3%, as shown in Table 2. The K-nearest neighbor classifier with accuracy was rated the lowest at 81.6% (Dhammi & Singh, 2015).

Table 2. Classification Results after Feature Selection

| <b>Classifier</b> | <b>TPR</b> | <b>FPR</b> | <b>Precision</b> | <b>Recall</b> | <b>Accuracy</b> |
|-------------------|------------|------------|------------------|---------------|-----------------|
| <b>LMT</b>        | 98.3       | 7.7        | 98.3             | 98.3          | 98.2857         |
| <b>NaïveBayes</b> | 96         | 13         | 95.59            | 96            | 96              |
| <b>SVM</b>        | 97.7       | 7.8        | 97.7             | 97.7          | 97.7143         |
| <b>Ridor</b>      | 94.3       | 15.8       | 94.2             | 94.3          | 94.2857         |
| <b>kNN</b>        | 81.7       | 8.19       | 81.8             | 81.8          | 81.6379         |

The clustering results of the unsupervised data, such as IP and file types that the team analyzed, found that 53% were portable executable files, followed by PDFs and zip files commonly shared and created by users. This may indicate a significant surface threat to the network (Dhammi & Singh, 2015). An important aspect to note is that 18% of the malware files analyzed had an IP address associated with them. This indicates enabling a network connection to an unauthorized server that may reside outside of the private network for an adversary to gain access through (Dhammi & Singh, 2015).

Dhammi and Singh (2015) concluded their research by indicating that the Simple K-means clustering algorithm provided optimal results in differentiating malware in different classes. Furthermore, Dhammi and Singh (2015) foresee that even zero-day attacks may be easily detected than previously when applying more than one ML algorithm to the system behavior.

## **G. CONCLUSION**

Observing the different methods such as studying the threads being created or the preference of one user using GUI applications instead of the Command Line Interface (CLI) terminal, user behavior can determine malicious activity with the appropriate machine language. Additionally, the most critical aspect of this thesis is to research the most appropriate data available to conduct analysis with and identify the insider threat. Furthermore, extreme care will be used to safeguard user data such that it does not infringe on personal privacy or violate organizational policy.

After studying the previous researchers' methods of identifying the insider threat, our contribution would be to spotlight user behavior accessing a file system. The previous studies focused on system performances, malware signatures, and user input behavior, but none addressed file access types. In our research, we will home in on developing and identification classification types based on file access. For instance, understanding uploading, downloading, lateral copying, and deleting files allows cybersecurity professionals to classify such activities through the data generated on the network. In recent years, the DOD continues to be targeted with data theft. According to Jed Pressgrove (2020), a contributor for Military.com, said:

Last year was notable for the number of data breaches, according to multiple sources. Earlier this month, company Risk Based Security stated that 15.1 billion records were reportedly exposed by breaches in 2019, a 284 percent jump from 2018. A report from nonprofit Identity Theft Resource Center estimates that the U.S. government and military experienced 83 data breaches in 2019, which accounted for 5.6 percent of the year's total breaches and resulted in the exposure of 3.6 million sensitive records.

THIS PAGE INTENTIONALLY LEFT BLANK

### III. EXPERIMENTAL DESIGN

This chapter explains the overall design of the experimentation that was conducted. The design process consists of three phases. In Phase 1, we prepared a test network with dummy files that mimic a business-like environment to analyze traffic. The results will shape the following phase. In Phase 2, we have created a test network to generate both normal and malicious traffic that uses ML models to classify each. We then analyze the ML model's ability to identify user and device actions on the test network. These metrics will be used to measure the effectiveness of the model to detect abnormal user behaviors on a network. Finally, in Phase 3, we conduct our model validation using regression in order to determine the type of classification based on file access behavior on the SMB file server.

Figure 12 shows how the experiment was designed and generated, along with how we evaluated the network traffic. This is what we considered as our matured implementation, which will be discussed later in this chapter.

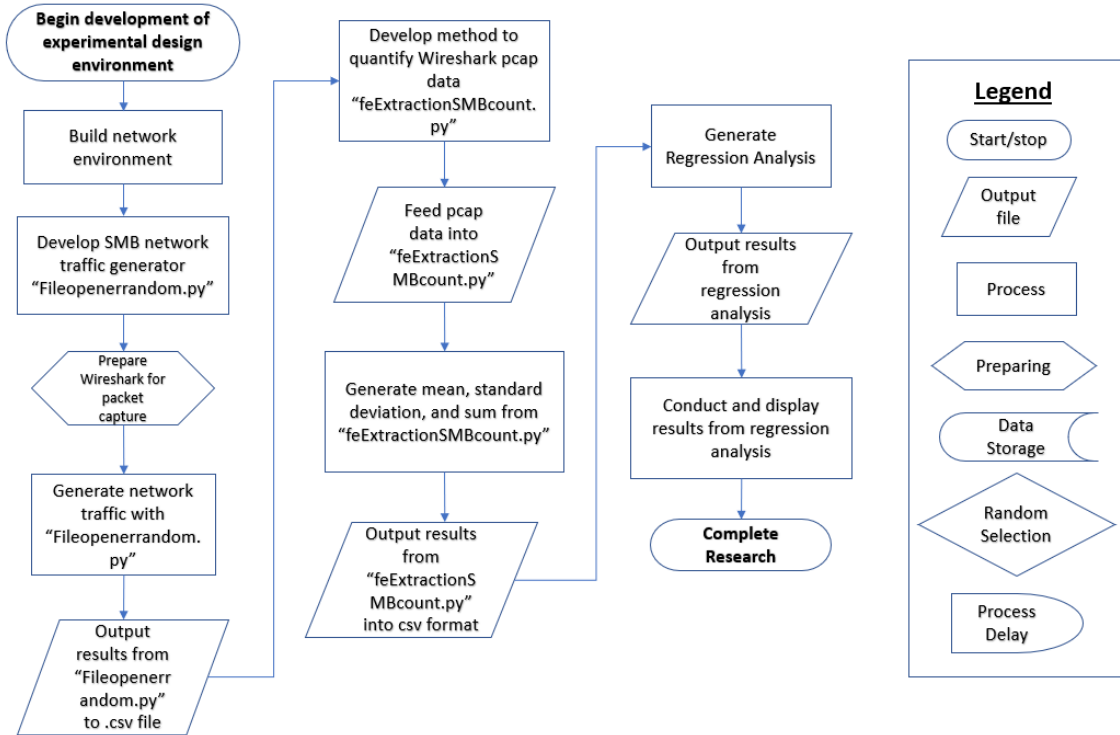


Figure 12. Overall Matured Experimental Design Flowchart

## A. PHASE 1: CREATION OF THE TESTING ENVIRONMENT

We start with Phase 1, which is where the test network was designed for traffic analysis. This test network includes dummy files to mimic normal network traffic and data usage. The network designed to support the experimentation consist of the following devices:

1. 1x NetGear ProSafe GS108 Gigabit Switch
2. 4x Category 5e Cables (5 feet each)
3. 2x Raspberry Pi (Generation 1), 32GB SD Cards
4. 1x Intel NUC (NUC10FNK)
5. 1x Dell Precision 7720 Laptop
6. 1x Linksys EA7500 Route

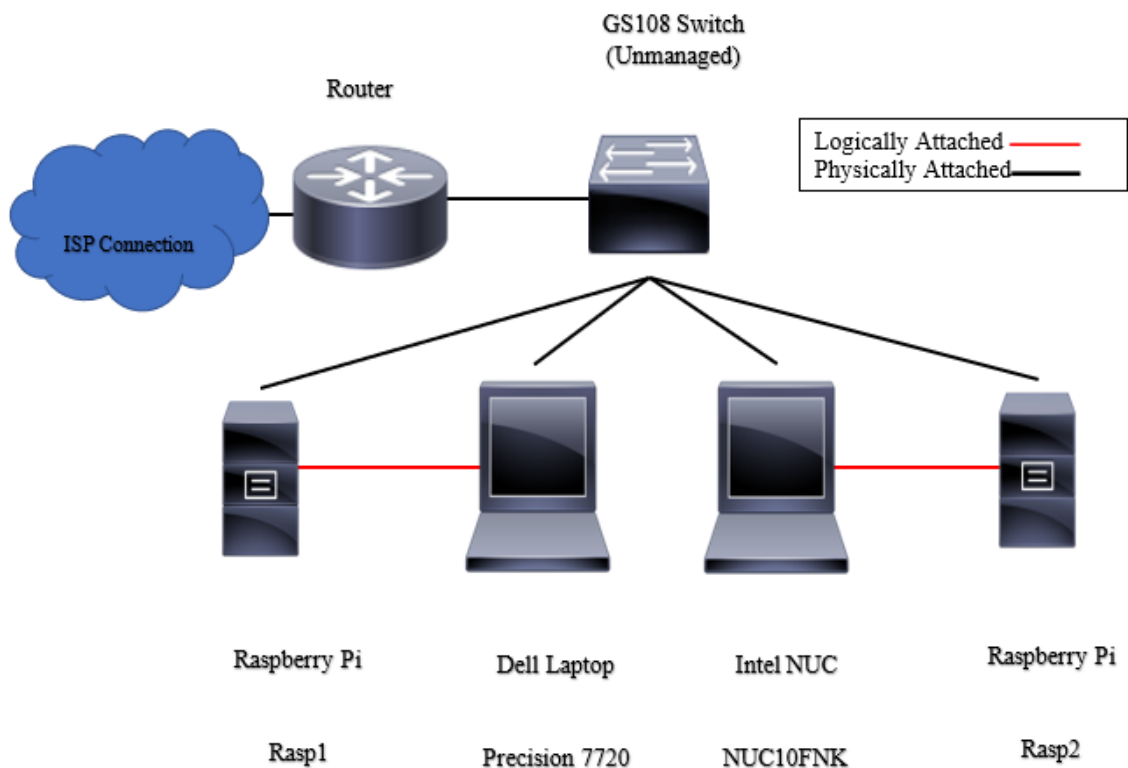


Figure 13. Logical Network Diagram of Experimental Environment

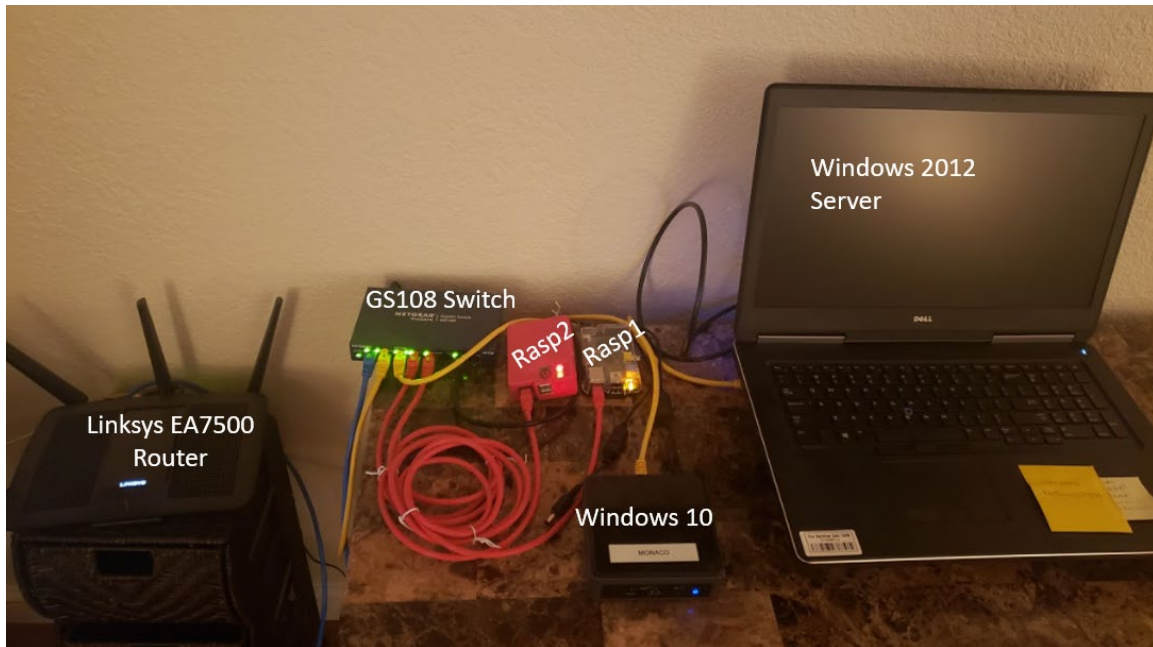


Figure 14. Actual Experimental Equipment Used

## 1. Network Configuration

The Raspberry Pis and the Intel NUCs (both referred to as end devices hereinafter) were connected to the NetGear switch with the Category 5e cables. Figure 15 shows the IP addresses for the associated devices in this experiment:

| Host                                                    | IP Address   |
|---------------------------------------------------------|--------------|
| Windows 2019 Server                                     | 192.168.1.50 |
| Windows 10                                              | 192.168.1.51 |
| Raspberry Pi (Rasp1)<br>Attached to Windows 2019 Server | 192.168.1.61 |
| Raspberry Pi (Rasp2)<br>Attached to Windows 10          | 192.168.1.62 |

Figure 15. Host and IP Diagram

To access and manage the network environment remotely, the NoMachine<sup>8</sup> application was used to serve as a remote desktop application. We then configured the router to use single port forwarding to allow external access to the internal local area network (LAN) of our experimental network environment.

The IP addresses were statically rather than dynamically assigned to ensure that the packet source and destination remained consistent. One NUC had Windows 2019 Server installed on it and the other NUC had Windows 10. Both NUCs did not receive any patches. See Annex 3 for a complete list of software installed.

The primary application that was used and installed on the Raspberry Pis for this experiment was Samba. The Raspberry Pis were configured to serve as the SMB servers and were configured with SMB version 2 (SMB2). The NUCs connect to the SMB servers by mapping to their SMB share file location. In Windows Explorer, you can view the SMB share drive attached to the NUC<sup>9</sup> in Figure 16. Natively, SMB2 does not implement encryption by default. To enable encryption, the configuration in the SMB configuration file to enable encryption to occur. The encrypted SMB traffic introduces challenges to classify SMB actions on the network because it encrypts the SMB commands end-to-end on the traffic. Wireshark sniffs the encrypted traffic and cannot identify the SMB commands in their encrypted state. This is true for network sensors that are tapped in-line with the physical and/or logical connection.

Our initial attempt to create an environment used the same hardware for both Windows 10 and Windows Server 2019. Our initial setup for Windows Server 2019 was not compatible with the NUC due to Intel not providing driver support for Windows Server 2019. This prevented us to configure the NUC to receive network access. We then switched the NUC for a Dell Precision 7720 laptop and installed Windows Server 2019. After

---

<sup>8</sup> NoMachine must be installed on all machines in order to access them by remote. Information for NoMachine can be found at [www.nomachine.com](http://www.nomachine.com).

<sup>9</sup> Operations such as adding and viewing SMB drives are identical on both NUCs; therefore, only one screenshot will provide evidence for the sake of brevity. Hereinafter, this will be a common occurrence since we conduct similar SMB operations within our networking environment.

verifying that SMB file sharing had occurred, we moved onto Phase 2 of the experimentation designed with Python (programming language) and Wireshark.

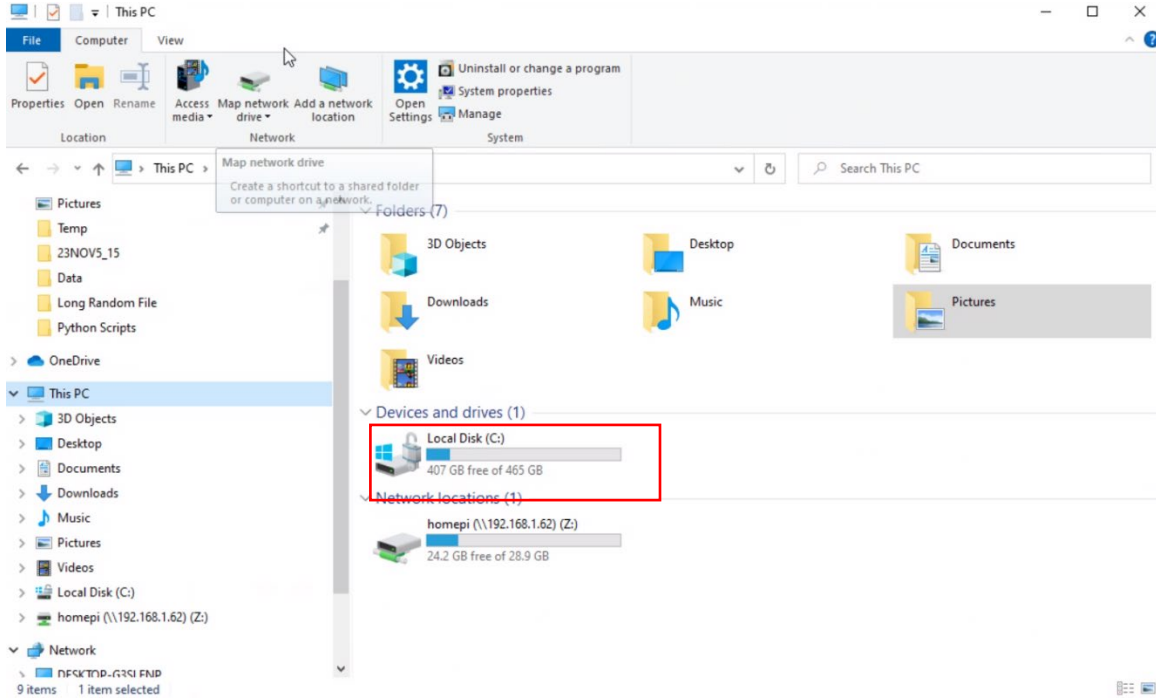


Figure 16. SMB Drive Attached on the NUC

## B. PHASE 2: SMB TRAFFIC GENERATION AND PACKET CAPTURE

In Phase 2, we were able to record packet captures (PCAP) with SMB traffic from the production network. We generated our own SMB network traffic using Python scripts that replayed several file access behaviors. This enabled us to conduct analysis of our approach and then transition to our mature process that was found to be best suited for SMB traffic generation and packet capture.

### 1. Early Implementation of SMB Traffic Generation with Python

Initially, a program was developed that opened and closed random files on the SMB servers (Rasp1 and Rasp2). These files are charted below in table one and two.

Table 3. Windows 10 and Rasp1 File Types

| Total Files 229 / Total Size: 382 MB |          |             |              |            |
|--------------------------------------|----------|-------------|--------------|------------|
| File Type                            | Quantity | Lowest Size | Highest Size | Total Size |
| .pdf                                 | 80       | 30 KB       | 14,172 KB    | 62.6 MB    |
| .docx                                | 31       | 12 KB       | 276 KB       | 1.26 MB    |
| .gif                                 | 2        | 6 KB        | 10 KB        | 308 KB     |
| HTML                                 | 6        | 203 KB      | 306 KB       | 1.40 MB    |
| .xls                                 | 17       | 1 KB        | 3,681 KB     | 7.05 MB    |
| .ppt                                 | 13       | 68 KB       | 9,360 KB     | 27.3 MB    |
| .doc                                 | 2        | 42 KB       | 44 KB        | 85.5 KB    |
| .mod                                 | 9        | 5 KB        | 55,857 KB    | 71.1 MB    |
| .mw                                  | 2        | 351 KB      | 799 KB       | 1.12 MB    |
| .png                                 | 63       | 7 KB        | 9,872 KB     | 27.8 MB    |
| .txt                                 | 3        | 2 KB        | 30 KB        | 32.4 KB    |

Table 4. Server 2019 and Rasp2 File Types

| Total Files 271 / Total Sizes 354 MB |          |             |              |            |
|--------------------------------------|----------|-------------|--------------|------------|
| File Type                            | Quantity | Lowest Size | Highest Size | Total Size |
| .acsm                                | 1        | 2 KB        | 2KB          | 12 KB      |
| .csv                                 | 12       | 1 KB        | 62,623 KB    | 129 MB     |
| .doc                                 | 2        | 42 KB       | 44 KB        | 85.5 KB    |
| .gif                                 | 3        | 6 KB        | 10 KB        | 20.8 KB    |
| .html                                | 6        | 203 KB      | 306 KB       | 1.40 MB    |
| IE shortcut                          | 8        | 1 KB        | 1 KB         | 8 KB       |
| .jpg                                 | 4        | 2,1246 KB   | 4,208 KB     | 11.9 MB    |
| Movieclip                            | 7        | 5 KB        | 8 KB         | 45 KB      |
| .mp4                                 | 2        | 17,528 KB   | 55,857 KB    | 71.1 MB    |
| .mw                                  | 2        | 351 KB      | 789 KB       | 1.12 MB    |
| .xml                                 | 35       | 12 KB       | 317 KB       | 1.65 MB    |
| .one                                 | 1        | 203 KB      | 203 KB       | 203 KB     |
| .odt                                 | 1        | 11 KB       | 11 KB        | 11 KB      |
| .pdf                                 | 97       | 30 KB       | 14,172 KB    | 81.5 MB    |
| .png                                 | 63       | 7 KB        | 9,072 KB     | 27.8 MB    |
| .pptx                                | 14       | 9,360 KB    | 42 KB        | 27.4 MB    |
| Shortcut                             | 1        | 1 KB        | 1 KB         | 1 KB       |
| .txt                                 | 4        | 2 KB        | 30 KB        | 34.8 KB    |
| .xlsx                                | 8        | 9 KB        | 275 KB       | 374 KB     |

The program would open and close each file in a synchronous order with sleep time<sup>10</sup> enabled to prevent applications from crashing. As each file opened and then closed on the SMB drive, the program would output when the file opened, closed, and the file name itself. We introduced the random function in Python to select files randomly in the

---

<sup>10</sup> Sleep time pauses the processes with the Python script for a specified or random amount of time, and then will continue operations. This is necessary for other file or system operations to complete prior to the Python script to carry on in its operations.

folder so that predictability would be minimized when we introduced the output to a .txt file to the ML algorithm. This process would continue until all the files have been opened and read. Figure 20 depicts a high-level view of the *fileopenerrandom.py* process.

We identified issues earlier in our early implementation of the SMB traffic generation. One problem that we experienced, was that as a file opened and then closes, the processing time-lagged and caused the *fileopenerrandom.py* to crash. This is due to attempts to close the application after its been closed. To address this issue, we reconfigured the script to use the *try and except* function, whereas the *try* would attempt to close out the application that the file opened with. With the *except* function, we implemented the *pass* feature that continues the process without crashing. Typically, we would use the *except* feature to catch the error; however, in this case, our goal was to continue to open files on the SMB share without disruption.

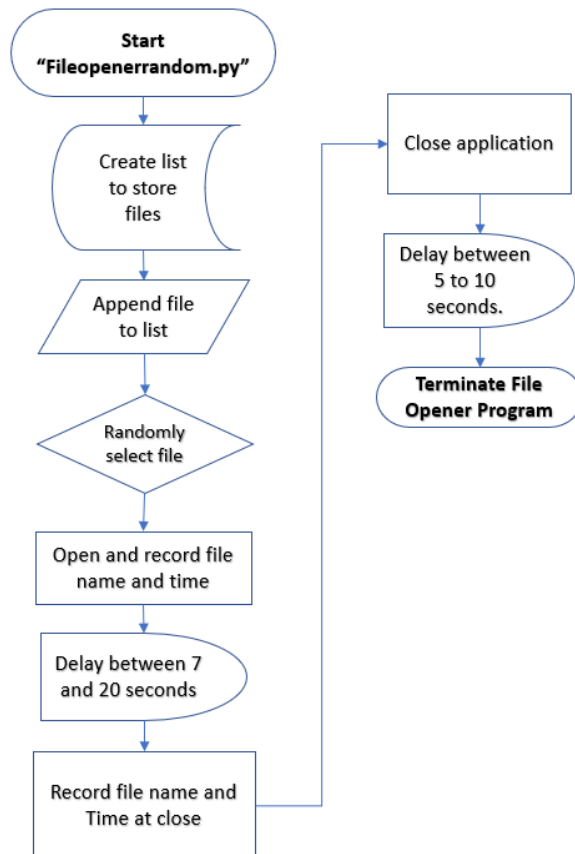


Figure 17. Random File Opener *Fileopenerrandom.py* Flowchart

## 2. Applying the Wireshark Filter

Before executing the program, we prepared Wireshark to look for traffic only using the following filter: IP Host 192.168.1.51 and IP Host 192.168.1.62 on the Ethernet interface (Figure 18). This enabled us to focus on just the network traffic between Rasp2 and Windows 10 when we executed the *fileopenerrandom.py* script. It is important to note that we chose to use Windows 10 and Rasp2 for our experimentation. Furthermore, the same operations conducted on Windows 10 and Rasp2 were executed on Windows Server 2019 and Rasp1. We recorded epoch timestamps for all scripts (see Figure 22)<sup>11</sup>.

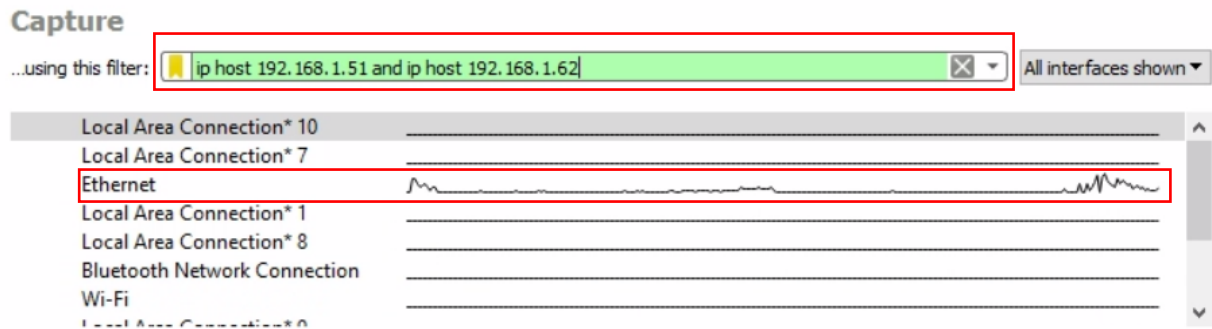
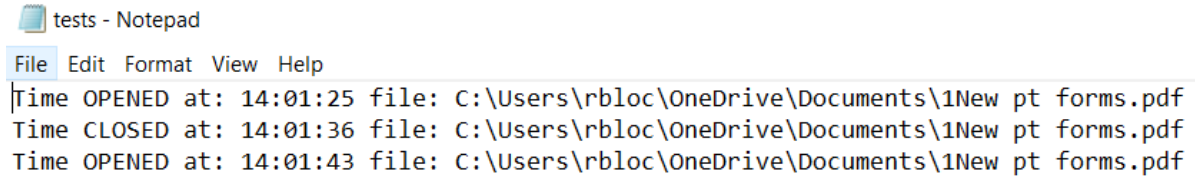


Figure 18. Wireshark Filter Settings

One of the limitations of *fileopenerrandom.py* is that it does not accurately capture the SMB commands that were being used initially. For example, a file opened on the SMB server but not edited was referred to as “Open”, when in actuality the SMB command is “Read”. The output from the Python script created a .txt file. We later changed the output to a .csv format to enable effortless ingestion into the ML algorithm.

<sup>11</sup> Epoch time is recorded in seconds since 00:00:00 UTC, January 1, 1970.



```
tests - Notepad
File Edit Format View Help
Time OPENED at: 14:01:25 file: C:\Users\rbloc\OneDrive\Documents\1New pt forms.pdf
Time CLOSED at: 14:01:36 file: C:\Users\rbloc\OneDrive\Documents\1New pt forms.pdf
Time OPENED at: 14:01:43 file: C:\Users\rbloc\OneDrive\Documents\1New pt forms.pdf
```

Figure 19. Output from fileopenerrandom.py

After trial-and-error, we matured our network traffic generating script as well as our Wireshark packet capture extraction in an effort to further optimize our workflow and process.

### 3. Matured Implementation of SMB Traffic Generation with Python

The next iteration of Python scripting captured the SMB commands accurately and reflects the commands in Wireshark, including additional information and creating an output to a .csv format. We revised our Python traffic generator, *SMBGenerator.py*, to mimic very basic user behavior. We also incorporated the “Read, Write, Download, Upload, and Lateral Copy”.

For example, random files open and close for random lengths of time from 5 to 15 minutes. “Random” .docx and .txt files can be written to at random when selected, but not every .docx or .txt document will be written to, and the number of characters that are written in the documents is random as well. The “Download” SMB command is a function where a file is downloaded from the Rasp2 to Windows 10 local system. The “Upload” SMB command is the function where a local file on the Windows 10 systems uploads a file to Rasp2. Finally, “Lateral Copy”, is the SMB command where a file from one directory on Rasp2 is moved to another directory on Rasp2.

Figure 20 depicts a high-level view of *SMBGernator.py* on how it processes basic user behavior and generator SMB traffic.

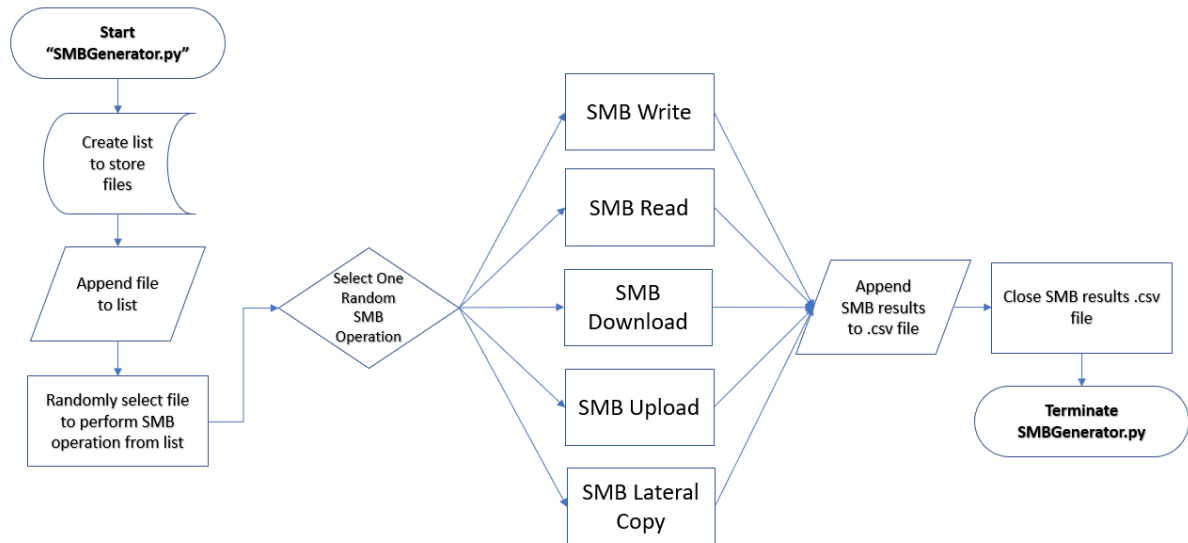


Figure 20. *SMBGenerator.py* High-Level Flowchart

The output from the *SMBGenerator.py* captures all the necessary data that we require to conduct our regression analysis with ML. Most importantly, was capturing the epoch time of when the SMB command was sent by *SMBGenerator.py*. To do so, we ensure that we use the *time()* function in Python, and return the results to the “sysTime” identifier every time an SMB command is executed. Figure 21 illustrates the SMB operation being conducted, one file at a time, one process at a time until all files in the targeted directory have been processed.

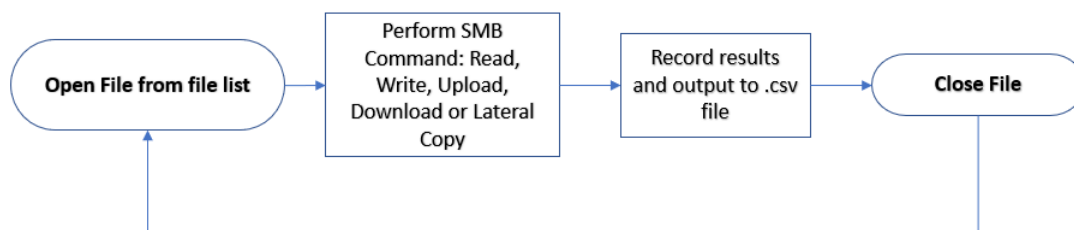


Figure 21. Single SMB Operation Flowchart

Figure 22 displays a sample output from *SMBGenerator.py* that created a .csv file. This file records the following information:

Filename: Records the filename used during the SMB process.

SysTime: The recorded epoch time when the SMB process started.

FileAction: The SMB command “Read, Write, Download, Upload or Lateral Copy” that was randomly selected to use for the process..

FileStatus: “A” - Last time file was accessed/read. “M” - Last time file was modified. “C” - Last time metadata was changed on file (Linux), or the creation time of the file (Microsoft).

FileMode: Records the inode (which is the metadata that includes file creation and access time, for example) of the file such as “Read, Write and Access” permission groups.

FileSize: Records the file size in bytes.

CreateTime: Records the time the file was created from the metadata in epoch time.

AccessTime: Records the time the file was last accessed in epoch time.

ModifyTime: Records the time the file was last modified in epoch time.

|    | A                                                                            | B                 | C            | D          | E        | F        | G                 | H                 | I                 |
|----|------------------------------------------------------------------------------|-------------------|--------------|------------|----------|----------|-------------------|-------------------|-------------------|
|    | Filename                                                                     | SysTime           | FileAction   | FileStatus | FileMode | FileSize | CreateTime        | AccessTime        | ModifyTime        |
| 1  | Z:\Documents\Data\1.txt                                                      | 1606201394.350100 | Download     | M          | 33206    | 6268628  | 1605728823.895080 | 1605728823.895080 | 1606196934.358780 |
| 2  | Z:\Documents\Data\2.txt                                                      | 1606201395.359710 | Read         | M          | 33206    | 321943   | 1605729192.484720 | 1605729192.484720 | 1606197042.722540 |
| 3  | Z:\Documents\Data\328BitFloat.gif                                            | 1606201407.573020 | Read         | M          | 33206    | 345249   | 1604604336.043580 | 1604604336.043580 | 1606191421.831930 |
| 4  | Z:\Documents\Data\4E7WGU.docx                                                | 1606201423.102400 | Download     | M          | 33206    | 9513     | 1606191966.653780 | 1606191966.653780 | 1606192801.613810 |
| 5  | Z:\Documents\Data\4Z2M17FHSZ.docx                                            | 1606201423.161100 | Write        | M          | 33206    | 39375    | 1606191966.703780 | 1606191966.703780 | 1606201432.052830 |
| 6  | Z:\Documents\Data\5WS09LUQR3.docx                                            | 1606201445.582830 | Read         | M          | 33206    | 51345    | 1606191966.723780 | 1606191966.723780 | 1606200255.590840 |
| 7  | Z:\Documents\Data\62ROG2HBTI.docx                                            | 1606201451.995370 | Read         | M          | 33206    | 92484    | 1606191966.803780 | 1606191966.803780 | 1606200258.252930 |
| 8  | Z:\Documents\Data\7EAFPE.docx                                                | 1606201457.437760 | Write        | M          | 33206    | 21735    | 1606191966.753780 | 1606191966.753780 | 1606201467.918390 |
| 9  | Z:\Documents\Data\8BitFixedPoint.gif                                         | 1606201480.382850 | Lateral Copy | M          | 33206    | 204718   | 1604604336.173580 | 1604604336.173580 | 1606192291.412860 |
| 10 | Z:\Documents\Data\8BitFixedPoint.png                                         | 1606201480.551970 | Read         | M          | 33206    | 353154   | 1604604336.203580 | 1604604336.203580 | 1606192297.109770 |
| 11 | Z:\Documents\Data\96CN7D5.docx                                               | 1606201494.134460 | Read         | M          | 33206    | 51471    | 1606191966.823780 | 1606191966.823780 | 1606197165.888700 |
| 12 | Z:\Documents\Data\Additional Notes on Cumulative Distribution Functions.docx | 1606201500.548320 | Download     | M          | 33206    | 455374   | 1604604336.243580 | 1605745716.527380 | 1606195018.615220 |
| 13 | Z:\Documents\Data\AEK57GEKT.docx                                             | 1606201500.648560 | Upload       | A          | 33206    | 0        | 1606191966.843780 | 1606191966.843780 | 1603305188.505880 |
| 14 | Z:\Documents\Data\AJ0Q0RE2.docx                                              | 1606201500.795920 | Upload       | M          | 33206    | 26019    | 1606191966.873780 | 1606191966.873780 | 1606195608.508870 |
| 15 |                                                                              |                   |              |            |          |          |                   |                   |                   |

Figure 22. SMBGenerator.py Sample Output

The output from *SMBGenerator.py* serves two purposes; it records every action that occurred on the SMB sever; and it is our experimental designs’ ground truth of our research. The data that is recorded by *SMBGenerator.py* uses the local system time in

epoch format. We accounted for the network time and other information using Wireshark PCAP, which we will discuss next.

#### 4. Wireshark Packet Capture Output

Figure 23 illustrates the first few packet captures from Wireshark that were generated from network activity by *SMBGenerator.py* Python script. This will be used to facilitate our regression analysis in Chapter 4. The blue arrow indicates that the rows and columns continue as a single packet.

|   | A   | B                 | C         | D        | E            | F        |
|---|-----|-------------------|-----------|----------|--------------|----------|
| 1 | No. | Time              | DeltaTime | Src Port | Destination  | Dst Port |
| 2 | 1   | 1606201392.724400 | 0         | 49763    | 192.168.1.62 | 445      |
| 3 | 2   | 1606201392.725520 | 0.001122  | 445      | 192.168.1.51 | 49763    |
| 4 | 3   | 1606201394.169490 | 1.443975  | 49763    | 192.168.1.62 | 445      |
| 5 | 4   | 1606201394.170270 | 0.000778  | 445      | 192.168.1.51 | 49763    |
| 6 | 5   | 1606201394.174470 | 0.004196  | 445      | 192.168.1.51 | 49763    |
| 7 | 6   | 1606201394.174720 | 0.000254  | 49763    | 192.168.1.62 | 445      |
| 8 | 7   | 1606201394.175920 | 0.001194  | 445      | 192.168.1.51 | 49763    |

| G        | H          | I       | J         | K      | L                                                    | M | N | O | P |
|----------|------------|---------|-----------|--------|------------------------------------------------------|---|---|---|---|
| Protocol | Packet Len | TCP Len | SMB CMD   | Window | Info                                                 |   |   |   |   |
| NBSS     | 55         | 1       |           | 6178   | NBSS Continuation Message                            |   |   |   |   |
| TCP      | 66         | 0       |           | 22691  | 445 > 49763 [ACK] Seq=1 Ack=2 Win=22691 Len=0 SLE=1  |   |   |   |   |
| SMB2     | 258        | 204     | Create    | 6178   | Create Request File: Documents\\Data                 |   |   |   |   |
| TCP      | 60         | 0       |           | 22691  | 445 > 49763 [ACK] Seq=1 Ack=206 Win=22691 Len=0      |   |   |   |   |
| SMB2     | 298        | 244     | Create    | 22691  | Create Response File: Documents\\Data                |   |   |   |   |
| SMB2     | 260        | 206     | Find,Find | 6178   | Find Request File: Documents\\Data SMB2_FIND_ID_BOTH |   |   |   |   |
| TCP      | 60         | 0       |           | 22691  | 445 > 49763 [ACK] Seq=245 Ack=412 Win=22691 Len=0    |   |   |   |   |

Figure 23. Wireshark Packet Capture Sample in CSV Format

The following fields were selected to be used for our regression analysis:

Table 5. Regression Analysis Fields

| Field Name  | Definition                                                                                                        |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| No          | The number of the packet in the sequence of order that was captured, beginning at one (1).                        |
| Time        | The epoch time recorded at the time of the SMB operation occurred on the network.                                 |
| Delta Time  | The change in the time between the last packet capture and the current capture.                                   |
| Source      | Source IP address of the end device that originated the SMB connection.                                           |
| Src Port    | Source port number that the SMB connection originated from.                                                       |
| Destination | The destination IP address to where the SMB connection is going.                                                  |
| Dst Port    | The destination port that the request is being made to.                                                           |
| Protocol    | The protocol that the packet in question is using.                                                                |
| Packet Len  | The packet length recorded in bytes.                                                                              |
| TCP Len     | The TCP length recorded in bytes.                                                                                 |
| SMB CMD     | The SMB command is used in the packet. It is important to note, that not every packet will record an SMB command. |
| Info        | The condensed field displays some of the data in the previous fields in brevity.                                  |

After exporting the PCAP to a .csv format, it is important to delete everything after the last “Close” SMB command, under the SMB command field in the PCAP .csv file before conducting any further analysis. If *SMBGenerator.py* processes all the files first, the script will complete and stop. However, Wireshark will continue to capture packets on the network because the client (NUC) will continue to send *keep-alive* packets to maintain the socket connection to the SMB server. This is not desirable because it can dilute the data that we are trying to analyze. Figure 24 illustrates the last “Close” SMB command and the *keep alive packets*. As shown in Figure 24, there are multiple “Close” commands associated with the PCAP. We are only interested in the last one because it signifies the

last file in the target shared folder that has been processed and closed; therefore, we delete all the packets after the last “Close” SMB command.

|        |        |          |              |       |              |       |      |     |     |       |                              |
|--------|--------|----------|--------------|-------|--------------|-------|------|-----|-----|-------|------------------------------|
| 234482 | 234481 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49763 | SMB2 | 182 | 128 | Close | Close Response               |
| 234483 | 234482 | 1.61E+09 | 192.168.1.51 | 49763 | 192.168.1.62 | 445   | TCP  | 54  | 0   |       | 49763 > 445 [ACK] Seq=356    |
| 234484 | 234483 | 1.61E+09 | 192.168.1.51 | 49763 | 192.168.1.62 | 445   | SMB2 | 146 | 92  | Close | Close Request File: Documer  |
| 234485 | 234484 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49763 | TCP  | 60  | 0   |       | 445 > 49763 [ACK] Seq=112    |
| 234486 | 234485 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49763 | SMB2 | 182 | 128 | Close | Close Response               |
| 234487 | 234486 | 1.61E+09 | 192.168.1.51 | 49763 | 192.168.1.62 | 445   | TCP  | 54  | 0   |       | 49763 > 445 [ACK] Seq=356    |
| 234488 | 234487 | 1.61E+09 | 192.168.1.51 | 49763 | 192.168.1.62 | 445   | TCP  | 55  | 1   |       | [TCP Keep-Alive] 49763 > 445 |
| 234489 | 234488 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49763 | TCP  | 66  | 0   |       | [TCP Keep-Alive ACK] 445 >   |
| 234490 | 234489 | 1.61E+09 | 192.168.1.51 | 49763 | 192.168.1.62 | 445   | TCP  | 55  | 1   |       | [TCP Keep-Alive] 49763 > 445 |
| 234491 | 234490 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49763 | TCP  | 66  | 0   |       | [TCP Keep-Alive ACK] 445 >   |
| 234492 | 234491 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49839 | TCP  | 60  | 0   |       | 445 > 49839 [ACK] Seq=112    |
| 234493 | 234492 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49839 | TCP  | 60  | 0   |       | [TCP Dup ACK 234491#1] 445   |
| 234494 | 234493 | 1.61E+09 | 192.168.1.62 | 445   | 192.168.1.51 | 49733 | TCP  | 60  | 0   |       | 445 > 49733 [ACK] Seq=112    |

Figure 24. Last Close SMB Command and Keep-Alive Packets

After the final SMB “Close” command and removing the excess *keep alive* packets, the PCAP .csv file will then be used as the input data for the *PcapExtraction.py* Python script.

## 5. Quantifying the Wireshark Packet Capture Data

To conduct our classification analysis, we had to create a python script that would take the data from the packet capture and present them to the regression algorithm. To do so, we created the *feExtractionSMBcount.py*. This program extracts features from a non-overlapping sliding window which we set to 30 packets. The following independent variables were calculated for every window:

Delta\_Time\_Mean: Calculated mean for delta time.

Delta\_Time\_Std: Calculated standard deviation delta time.

Window\_Size\_Mean: Calculated mean for window size.

Window\_Size\_Std: Calculated standard deviation for window size.

Packet\_Length\_Mean: Calculated mean for packet length.

Packet\_Length\_Std: Calculated standard deviation for packet length.

TCP\_Len\_Mean: Calculated mean for TCP length.

TCP\_Len\_Std: Calculated standard deviation for TCP length.

The following SMB commands are calculated using the “Sum” function, which tallies the amounts that were called or executed. We are primarily focused on understanding the SMB network behavior of the “Create, Read, Write, and Close” SMB commands.

Create: SMB command that is created by the client to access a file or creates a file.

Read: SMB command that opens a file or folder.

Write: SMB command that indicates data has been written to file.

Close: SMB command that closes the file or the SMB session.

Session\_Setup: SMB command that setups the SMB session by the client.

Session\_Logoff: SMB command that logs the client off the SMB server.

Tree\_Connect: SMB command request to mount to the SMB drive.

GetInfo: SMB command to retrieve the metadata on the selected file.

Find: SMD command to locate the requested file.

Create\_Ioctl: SMB command for Input/Output System control calls

No\_Op: No operation conducted.

It is important to note that all though we incorporate other SMB commands, the “Read, Write, Close, and Create” are of most interest as these reflect to some extent the action a user is performing. Figure 25 illustrates a high-level view of how *feExtractionSMBcount.py* quantifies the PCAP data. Figure 26 represents a sample from the *feExtractionSMBcount.py* output. in .csv format.

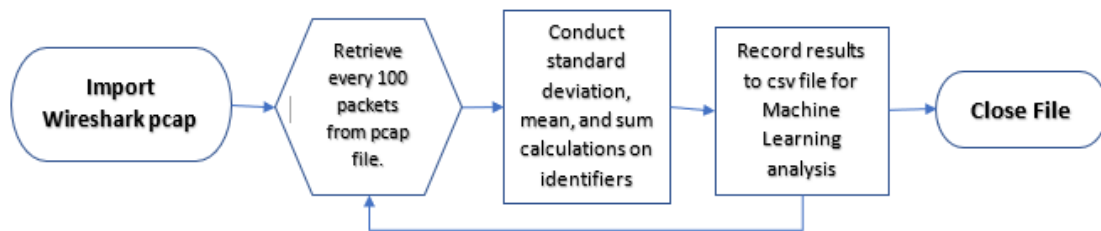


Figure 25. feExtractionSMBCount.py High-Level Process

The connecting arrows indicate that it is a continuation of the output.

|   | A      | B               | C              | D                | E               | F                  | G                 |
|---|--------|-----------------|----------------|------------------|-----------------|--------------------|-------------------|
|   | window | Delta_Time_Mean | Delta_Time_Std | Window_Size_Mean | Window_Size_Std | Packet_Length_Mean | Packet_Length_Std |
| 1 |        |                 |                |                  |                 |                    |                   |
| 2 | 0      | 0.0165131       | 0.145115455    | 17242.75         | 7802.25005      | 835.5              | 703.1581212       |
| 3 | 1      | 0.01651476      | 0.145115265    | 17407.88         | 7740.202946     | 850.09             | 701.9352627       |
| 4 | 2      | 0.01650455      | 0.145116395    | 17407.88         | 7740.202946     | 864.57             | 700.530899        |
| 5 | 3      | 0.0020648       | 0.016381785    | 17573.01         | 7674.066115     | 869.85             | 697.8977722       |
| 6 | 4      | 0.00205728      | 0.016382554    | 17407.92         | 7740.144336     | 869.79             | 697.9683547       |
| 7 | 5      | 0.00201757      | 0.01638213     | 17242.83         | 7802.135472     | 868.52             | 699.1338433       |

|  | H            | I           | J             | K              | L            | M       | N               | O                       | P      | Q    | R     | S     |
|--|--------------|-------------|---------------|----------------|--------------|---------|-----------------|-------------------------|--------|------|-------|-------|
|  | TCP_Len_Mean | TCP_Len_Std | Session_Setup | Session_Logoff | Tree_Connect | GetInfo | GetInfo_GetInfo | GetInfo_GetInfo_GetInfo | Create | Read | Write | Close |
|  | 780.9        | 703.8280251 | 0             | 0              | 0            | 6       | 2               | 0                       | 6      | 1    | 0     | 4     |
|  | 795.49       | 702.6189176 | 0             | 0              | 0            | 6       | 2               | 0                       | 6      | 1    | 0     | 4     |
|  | 810.09       | 701.0894422 | 0             | 0              | 0            | 6       | 2               | 0                       | 6      | 1    | 0     | 4     |
|  | 815.37       | 698.4620863 | 0             | 0              | 0            | 6       | 2               | 0                       | 5      | 2    | 0     | 4     |
|  | 815.37       | 698.4620863 | 0             | 0              | 0            | 6       | 2               | 0                       | 5      | 2    | 0     | 4     |
|  | 814.1        | 699.6259823 | 0             | 0              | 0            | 6       | 2               | 0                       | 4      | 3    | 0     | 4     |

|  | T    | U         | V              | W            | X              | Y                      | Z                | AA    |
|--|------|-----------|----------------|--------------|----------------|------------------------|------------------|-------|
|  | Find | Find_Find | Create_SetInfo | Create_Ioctl | Create_GetInfo | Create_GetInfo_GetInfo | Create_Find_Find | No_Op |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |
|  | 4    | 2         | 0              | 0            | 2              | 2                      | 0                | 0     |

Figure 26. feExtractionSMBCount.py Sample Output

## 6. Preparing to Test for SMB Behavior Using Machine Learning

With our experimental network completed, we can move forward with classifying SMB network behavior with ML. With the data generated, we will train and then test our

model. This will enable us to predict the type of SMB operation that may occur given the data that is generated.

### **C. PHASE 3: MODEL VALIDATION**

In Phase 2, we analyzed the capability of the ML model to classify malicious versus benign traffic in the network. We will now look at metrics and measures of effectiveness to determine if the ML model is accurate.

After extracting the dataset using the *feExtractionSMBcount.py* script, we split our dataset in half; the first half is to train the model and the second half is to test the model. It is important to note that the window size is set to 30 which represents the number of instances that are being evaluated in the ML at a time. As each window is being processed, the sum of each SMB action is calculated as well as the data in the TCP header. For example, the labels (independent variables) were identified such as the Window Size, Packet Length, and TCP length. The TCP header is calculated to derive the mean and standard deviation and the SMB commands are being tallied.

As each window is being processed, the features (or dependent variables) are being calculated to classify and predict the type of SMB command is be taken such as Create, Write, Read and, Close.

## IV. DATA ANALYSIS AND RESULTS

In this chapter, we will present our analysis of the classification process. We conducted our experiment using SMB operations: the download, upload, lateral copy, and delete operations to help classify and predict the type of file access behavior. Additionally, our experimentation utilized different file types and sizes, found in Chapter II tables 1 and 2, and was conducted using two other operating systems to add complexity and depth to our model.

The team conducted the following SMB operations using every file in our target directory. Our first SMB operation completed lateral copying of the files from one file directory on the SMB server to another location on the same server. Then we downloaded all of the files from the SMB server to the client. We followed up by uploading all of the files from the client to the SMB server. Lastly, we deleted all the files in our target directory on the SMB server.

### A. CLASSIFICATION METRICS

It is essential to understand the numbers behind the metrics to quantify the SMB commands<sup>12</sup> that classify file access behavior. The metrics provide us with the accuracy and precision data required to explain the SMB prediction and classification in our model. This model analyzed the network packets we collected over the associated SMB operation with a window size set to 30. With our sliding window set to analyze 30 network packets simultaneously, multiple SMB command features were identified and then associated with the SMB operation. Our model created a multi-label classification from the data that resulted from the sliding windows. The multi-label classifications use the read, write, close, and create labels based upon the SMB commands used. Then, the precision, recall, support, and the f1-score were associated with the labels to describe their accuracy. Figures 27 through 33 will illustrate the significance of these values.

---

<sup>12</sup> SMB Commands consisted of Read, Write, Create, and Close.

Our model's predictive accuracy depends on understanding the confusion matrix's values and their associated SMB commands. We must first understand what True Positive, True Negative, False Positive, and False Negatives are. These values will determine the Precision and Recall that will help predict the user's action.

|                  |          | Actual Values       |                     |
|------------------|----------|---------------------|---------------------|
|                  |          | Negative            | Positive            |
| Predicted Values | Negative | True Negative (TN)  | False Positive (FP) |
|                  | Positive | False Negative (FN) | True Positive (TP)  |

Figure 27. Confusion Matrix

True Positive (TP): “Is an outcome where the model *correctly* predicts the *positive* class” (Google Developers, 2020).

False Positive (FP): “Is an outcome where the model *incorrectly* predicts the *positive* class” (Google Developers, 2020).

False Negative (FN): “Is an outcome where the model *incorrectly* predicts the *negative* class” (Google Developers, 2020).

True Negatives (TN): “Is an outcome where the model *correctly* predicts the *negative* class” (Google Developers, 2020).

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Figure 28. Recall Formula

Recall (also known as sensitivity): percentage of correct predictions for the actual positives<sup>13</sup>. Figure 28 illustrates the formula to derive the recall value.

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Figure 29. Precision formula

Precision: percentage value of those examples that are identified as positive was confirmed to be true. Figure 29 illustrates the formula for precision.

$$f1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 30. f1-score

F1-score: “The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0” (sklearn.metrics.f1\_score”, n.d.).

Support: total number of occurrences found in our target features (also known as our dependent variables) (“sklearn.metrics.f1\_score”, n.d.).

---

<sup>13</sup> More information about Recall, Precision, and F1-score can be found at [https://www.scikit-yb.org/en/latest/api/classifier/classification\\_report.html](https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html).

Micro-average: “Calculate metrics globally by counting the total true positives, false negatives and false positives” (“sklearn.metrics.f1\_score”, n.d.).

Macro-average: “Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account” (“sklearn.metrics.f1\_score”, n.d.).

Weighted average: “Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters ‘macro’ to account for label imbalance; it can result in an F-score that is not between precision and recall” (“sklearn.metrics.f1\_score”, n.d.).

Sample Average: Calculate metrics for each instance, and find their average” (“sklearn.metrics.f1\_score”, n.d.).

With the metrics identified for our classification analysis, we will cover the study of SMB operations. It is important to note that we relate these operations to what an insider threat may do on a network in terms of file access by moving (or removing) large quantities of files around and off the network.

For example, a malicious user may want to steal all the files on a specific directory by downloading all files. A malicious user may also try to move the contents from one network directory to another location (i.e., lateral copy). They may want to delete all the files in a directory on the network for an arbitrary reason. Malware can also conduct these types of actions; therefore, it is essential to classify and understand what uploading, downloading, lateral copy, and deleting files look like through ML. We selected all files in a target directory and not just individual files or a selected group of files through this research. Although these actions do not necessarily disrupt the SMB server, they do present the issue of disrupting business operations where critical documents can be stolen, corrupted, and removed altogether.

Finally, Windows 10 used a smaller number of files to conduct SMB operations instead of the same files on Windows Server 2019 (Server 2019), which had many files. The analysis data will have different ratings in their features, but overall, the classification

should be similar when comparing histograms. Windows 10 and Server 2019 used other numbers of files and file sizes analyzed with our classification model. We chose this method because regardless of the number of files and their size, and the operating system, our model, can generalize the classifications of the type of file access behavior.

## **B. SMB CLASSIFICATIONS AND RESULTS**

The results from our model are displayed in Figures 34 through 41. The histograms are the graphical representations of what our classification model predicted based upon the packet captures when SMB operations occurred. The tables associated with the histograms are the actual values that our classification model determined, such as the precision, recall, and f1-scores. Through the remainder of this chapter, we will refer to the f1-score more often because it takes in both and strikes the middle ground between the precision and recall scores. Therefore, we selected the f1-score as our metric when weighing the relevancy of our classification model's features.

For clarity, when we discuss SMB commands such as “write,” “read,” “create,” and “close,” we are explicitly stating the actual command sent to the SMB server. The SMB commands are different than when we discuss the Write, Read, Create, and Close features associated with the statistical data that our classification model derived. For example, the precision, recall, f1-score, and support values are associated with the features previously named.

Additionally, we defined our classification (Upload, Download, Lateral Copy, and Delete) based upon the group of features. For instance, when we reference the Upload classification, the values in the features (Create, Read Write, and Close) make up its classification. Each classification will have the same set of features, but each feature's values will be different for each classification, which gives the unique characteristic of each classification.

Recall Raspberry Pis are functioning as SMB servers. Windows 10 and Server 2019 are the clients. When we discuss the clients, we are explicitly discussing Windows 10 and Server 2019 together, unless otherwise stated. Finally, Windows 10 used 229 files with a combined size of 382MB. Server 2019 used 271 files with a total size of 354MB. Having

different files is essential because even though both clients have different numbers of files and file sizes, the classifications that we defined and the associated features will display similar accuracy results.

### **1. Upload SMB Operation**

The results from uploading files from Windows 10 to the SMB server are in Figure 31. The analysis displays the amount of support (or the frequency occurrence of the targets). Out of the four SMB features, the Create feature occurred the most with 395 occurrences, and according to the recall, it carried a score of 95%. Server 2019 performed very similarly with the Create feature support of 486 shown in Figure 32.

In Figure 32, the Write feature recorded the second-highest amount of actual feature occurrence with 463.

A unique characteristic of the Upload classification for Windows 10 and Server 2019 is that the f1-score and support score for the Read feature was null. The f1-score indicated no SMB “read” command during the upload operation as ground truth because the user did not access any files on the SMB server. The user placed the files on the SMB server instead. According to Figures 31 and 32, both operating systems made very small predictions in the SMB “read” commands.

Overall, both Windows 10 and Server 2019 feature predictions appear nearly identical to each other regardless of the files processed in the Upload classification regarding the histograms. It is important to remember that Windows 10 and Server 2019 processed different amounts of files. The visual comparisons of the histograms follow the same pattern between one another. One characteristic to note is that the Write feature predictions for Windows 10 in Figure 31 are smaller than Server 2019. We assumed that the Upload classification's characterization is that the Create feature will be the dominant, followed by either the Close or Write as the second feature and the Read feature being a non-dominate feature.

## **2. Download SMB Classification**

Windows 10 and Server 2019 SMB Download classification look nearly identical according to the histograms in Figures 33 and 34. In terms of ground truth, SMB "read" command for Windows 10 and Server 2019 in the SMB Download classification score are high. In Figure 33, Windows 10 recorded an f1-score of 97%, and Server 2019's f1-score recorded a score of 99%, shown in Figure 34. The f1-scores indicate that the Read feature will be the most prominent in the Download classification regardless of the operating system.

Another clear distinction is that both operating systems recorded nearly no SMB "write" commands when downloading files. Windows 10 recorded support of one according to Figure 33, and Server 2019 recorded zero support in the Write feature shown in Figure 34. We can generalize that the SMB Download classification will consist of little to no SMB "write" commands. We can infer the SMB "write" command will have little to no presence in support value associated with the SMB Download classification.

## **3. Lateral Copy SMB Classification**

Both operating systems reported the same support value for the Write feature when they conducted the lateral copy of files on the SMB server. This Write feature closely resembles the characteristics of the Write feature found in the SMB Download classification.

In Figure 35, Windows 10 has a much higher f1-score in the Read feature at 91%, whereas Server 2019 in Figure 36 reports an f1-score of 83%. The Read feature has a lower score on the SMB server than on a client machine because files did not transfer to another server/client.

Overall, both Windows 10 and Server 2019 exhumed higher f1-scores collectively than any other classification when we only consider the Create, Read, and Close features. The score of the features is a distinct characterization of the lateral copy classification.

#### **4. Delete SMB Classification**

Of all the classifications, the SMB Delete classification marked the highest f1-scores, with Windows 10 having 100% for both Read and Close features shown in Figure 37. Server 2019 has f1-scores of 97% the Create feature and 95% in the Close feature shown in Figure 38.

Both Windows 10 and Server 2019 recorded the same f1-score with 0% regarding the Write feature. Server 2019 recorded an f1-score of 96% in Figure 38 for the Read feature. It is unclear why Server 2019 would conduct SMB "read" commands for an SMB Delete classification. It may be operating in a server role with different system processes that Windows 10 does not have or use that influenced the behavior. It is important to note that there have been no additional group policy settings made or any other server configurations adjusted other than allowing the SMB server to be logically attached to Windows 10 and Server 2019

#### **5. Overall SMB Feature Model Performance**

The classification model performed very well in its prediction and determined the data behind the features associated with the type of SMB operations used. Referencing Figures 39 and 40, we relate the success of our classification model's accuracy using the f1-score because it represents the harmonic balance between the precision and recall scores (i.e., the f1-score is the middle ground between precision and the recall scores). The overall results for Windows 10 in Figure 39 and Server 2019 are in Figure 40.

With Windows and Server 2019 performing SMB operations with different quantities of files (Windows 10 having 229 and Server 2019 having 271), both overall provided similar f1-scores with their features. For example, with Windows 10, Figure 39 shows an f1-score of 92% for the Create feature and an f1-score of 94% for the Read feature. Similarly, Server 2019 in Figure 40 also recorded the Create feature with an f1-score of 92% and a Read feature with 92%. The marginal difference between Windows 10 and Server 2019 Create feature was 0.2% (Windows 10 having 92.4% and Server 2019 has 92.2%). The 0.2% difference is critical because regardless of the files used, both clients

behaved similarly, making the classification model suitable to be used in an enterprise environment.

Additionally, the f1-score for the Write feature was nearly identical in overall performance for both clients, which lends credence that this classification model provides consistency in predicting and determining the file access behavior.

Furthermore, Windows 10 and Server 2019's Read feature had a wider marginal difference, but still very close with a 2% gap between the f1-scores. As with observing the near-identical performance with the Create feature for Windows 10 and Server 2019, we can argue that both systems perform nearly identical concerning the Read feature when we analyzed it with our model.

Windows 10 and Server 2019 also performed similarly to our classification model when we observed the Write feature. Windows 10 recorded an 86% f1-score concerning the Write feature, while Server 2019 recorded an f1-score of 86%. The marginal difference that our model determined between Windows 10 and Server 2019 is 0.3%.

Finally, our model showed similar results with Windows 10 and Server 2019, analyzing the data from the SMB "close" commands. Windows 10 recorded an f1-score of 89%, shown in Figure 39, while Server 2019 recorded an f1-score of 90%, as shown in Figure 40. The marginal difference between Windows 10 and Server 2019 is 1%, which indicates that our model performed consistently with both systems.

Suppose the overall Read, Create, Close, and Write features were inconsistent between Windows 10 and Server 2019 by not having similar scores. In that case, our model may not support an enterprise environment well enough to predict and determine the type of SMB classifications that would occur.

Considering that both Windows 10 and Server 2019 used different data quantities to process their SMB operations, our model predicted accurately and closely resembled one another in terms of metrics. Though our classifications can be generalized, the features that our model determined displayed similar characteristics to what Windows 10 and Server 2019 had.

## 6. Classification Figure Results

### Windows 10 Upload Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.947     | 0.957  | 0.952    | 395     |
| Read         | 0.000     | 0.000  | 0.000    | 0       |
| Write        | 0.925     | 0.830  | 0.875    | 253     |
| Close        | 0.914     | 0.914  | 0.914    | 349     |
| micro avg    | 0.903     | 0.910  | 0.907    | 997     |
| macro avg    | 0.697     | 0.675  | 0.685    | 997     |
| weighted avg | 0.930     | 0.910  | 0.919    | 997     |
| samples avg  | 0.828     | 0.824  | 0.814    | 997     |

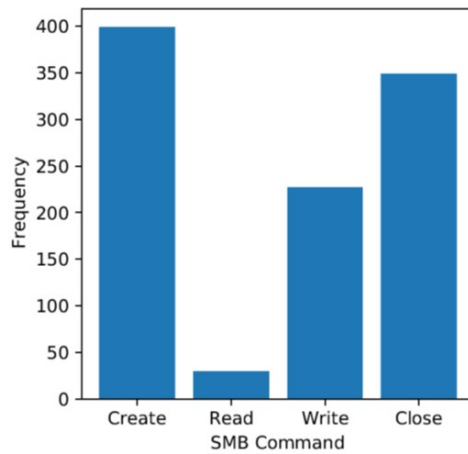


Figure 31. Windows 10 Massive Upload Chart and Histogram

## Server 2019 Upload Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.933     | 0.862  | 0.896    | 486     |
| Read         | 0.000     | 0.000  | 0.000    | 0       |
| Write        | 0.930     | 0.808  | 0.865    | 463     |
| Close        | 0.890     | 0.890  | 0.890    | 453     |
| micro avg    | 0.897     | 0.853  | 0.875    | 1402    |
| macro avg    | 0.688     | 0.640  | 0.663    | 1402    |
| weighted avg | 0.918     | 0.853  | 0.884    | 1402    |
| samples avg  | 0.791     | 0.766  | 0.765    | 1402    |

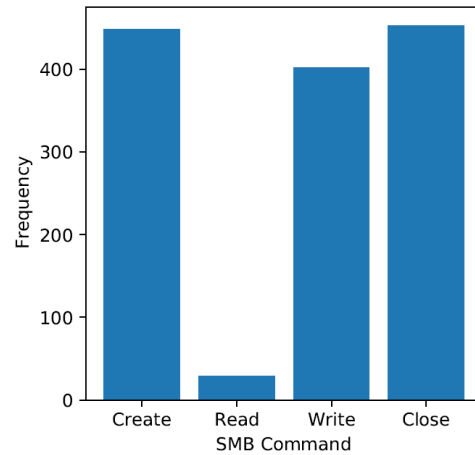


Figure 32. Server 2019 Massive Upload Chart and Histogram

## Windows 10 Download Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.812     | 0.867  | 0.839    | 135     |
| Read         | 0.983     | 0.962  | 0.973    | 928     |
| Write        | 0.000     | 0.000  | 0.000    | 1       |
| Close        | 0.789     | 0.759  | 0.774    | 133     |
| micro avg    | 0.942     | 0.928  | 0.935    | 1197    |
| macro avg    | 0.646     | 0.647  | 0.646    | 1197    |
| weighted avg | 0.942     | 0.928  | 0.935    | 1197    |
| samples avg  | 0.173     | 0.173  | 0.171    | 1197    |

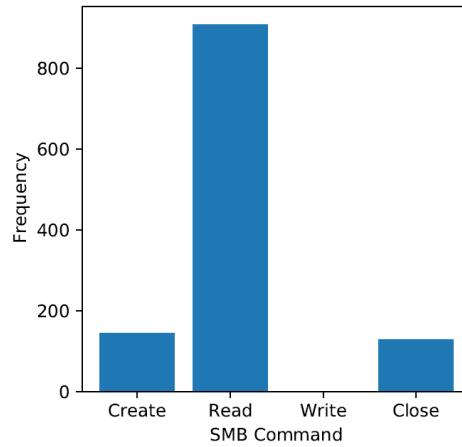


Figure 33. Windows 10 Massive Download Chart and Histogram

## Server 2019 Download Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.914     | 0.943  | 0.928    | 192     |
| Read         | 0.996     | 0.990  | 0.993    | 960     |
| Write        | 0.000     | 0.000  | 0.000    | 0       |
| Close        | 0.921     | 0.879  | 0.899    | 198     |
| micro avg    | 0.973     | 0.967  | 0.970    | 1350    |
| macro avg    | 0.708     | 0.703  | 0.705    | 1350    |
| weighted avg | 0.973     | 0.967  | 0.970    | 1350    |
| samples avg  | 0.138     | 0.138  | 0.137    | 1350    |

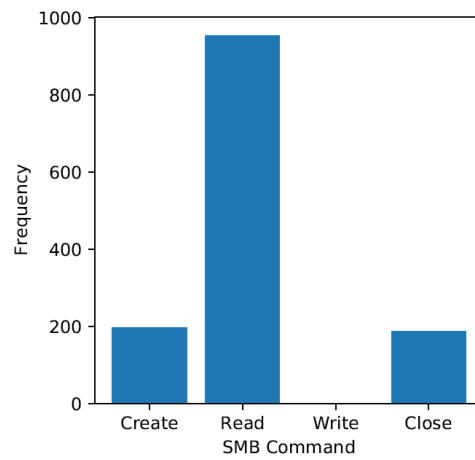


Figure 34. Server 2019 Massive Download Chart and Histogram

## Windows 10 Lateral Copy Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.928     | 0.896  | 0.912    | 443     |
| Read         | 0.901     | 0.921  | 0.911    | 482     |
| Write        | 0.000     | 0.000  | 0.000    | 0       |
| Close        | 0.891     | 0.893  | 0.892    | 401     |
| micro avg    | 0.905     | 0.904  | 0.905    | 1326    |
| macro avg    | 0.680     | 0.678  | 0.679    | 1326    |
| weighted avg | 0.907     | 0.904  | 0.905    | 1326    |
| samples avg  | 0.792     | 0.790  | 0.778    | 1326    |

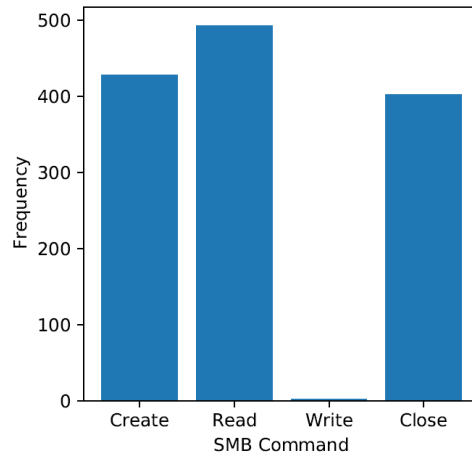


Figure 35. Windows 10 Massive Lateral Copy Chart and Histogram

## Server 2019 Lateral Copy Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.943     | 0.923  | 0.933    | 854     |
| Read         | 0.870     | 0.803  | 0.835    | 651     |
| Write        | 0.000     | 0.000  | 0.000    | 0       |
| Close        | 0.892     | 0.911  | 0.902    | 844     |
| micro avg    | 0.905     | 0.885  | 0.895    | 2349    |
| macro avg    | 0.676     | 0.659  | 0.667    | 2349    |
| weighted avg | 0.904     | 0.885  | 0.894    | 2349    |
| samples avg  | 0.463     | 0.464  | 0.457    | 2349    |

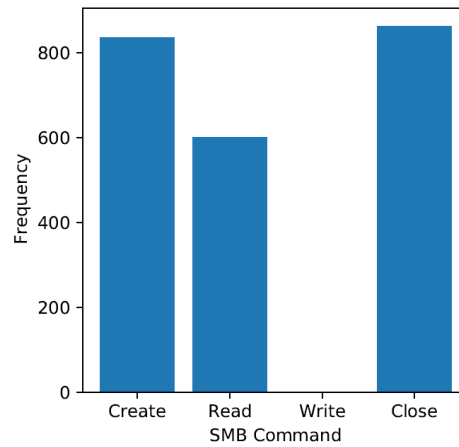


Figure 36. Server 2019 Massive Lateral Copy Chart and Histogram

# Windows 10 Delete Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 1.000     | 1.000  | 1.000    | 54      |
| Read         | 0.000     | 0.000  | 0.000    | 1       |
| Write        | 0.000     | 0.000  | 0.000    | 1       |
| Close        | 1.000     | 1.000  | 1.000    | 54      |
| micro avg    | 0.991     | 0.982  | 0.986    | 110     |
| macro avg    | 0.500     | 0.500  | 0.500    | 110     |
| weighted avg | 0.982     | 0.982  | 0.982    | 110     |
| samples avg  | 0.994     | 0.991  | 0.990    | 110     |

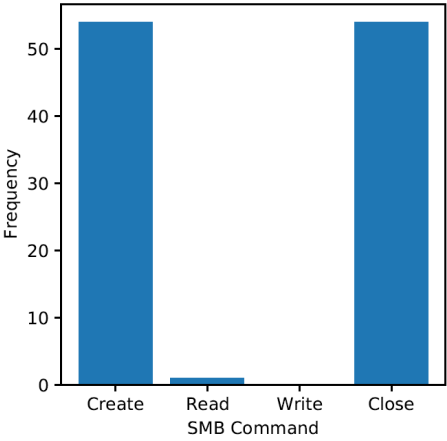


Figure 37. Windows 10 Massive Delete Chart and Histogram

## Server 2019 Delete Classification

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Create       | 0.978     | 0.978  | 0.978    | 89      |
| Read         | 0.980     | 0.941  | 0.960    | 51      |
| Write        | 0.000     | 0.000  | 0.000    | 0       |
| Close        | 0.921     | 1.000  | 0.959    | 82      |
| micro avg    | 0.956     | 0.977  | 0.967    | 222     |
| macro avg    | 0.720     | 0.730  | 0.724    | 222     |
| weighted avg | 0.957     | 0.977  | 0.967    | 222     |
| samples avg  | 0.119     | 0.121  | 0.120    | 222     |

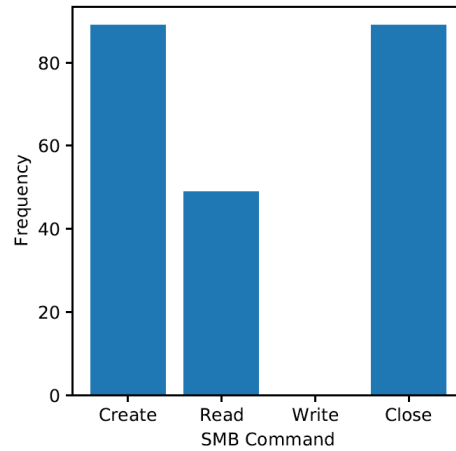


Figure 38. Server 2019 Massive Delete Chart and Histogram

Windows 10 Overall Classification

|        | <b>Precision</b> | <b>Recall</b> | <b>F1-Score</b> | <b>Support</b> |
|--------|------------------|---------------|-----------------|----------------|
| Create | 0.923            | 0.921         | 0.922           | 1027           |
| Read   | 0.934            | 0.948         | 0.941           | 1411           |
| Write  | 0.917            | 0.824         | 0.868           | 255            |
| Close  | 0.892            | 0.888         | 0.890           | 937            |

Figure 39. Windows 10 Overall SMB Chart

Server 2019 Overall Classification

|        | <b>Precision</b> | <b>Recall</b> | <b>F1-Score</b> | <b>Support</b> |
|--------|------------------|---------------|-----------------|----------------|
| Create | 0.938            | 0.910         | 0.924           | 1621           |
| Read   | 0.931            | 0.915         | 0.923           | 1662           |
| Write  | 0.930            | 0.808         | 0.865           | 463            |
| Close  | 0.896            | 0.906         | 0.901           | 1577           |

Figure 40. Server 2019 Overall SMB Chart

## V. CONCLUSION

We found that identifying file access behavior may not have been widely researched before. Our goal for this research was to determine if we can predict and classify file access behavior using the packet headers found in the network traffic. Using ML to analyze network packets to classify would enable cybersecurity professionals to be proactive in defending the network rather than reactive.

By researching commercial network traffic generated solutions, experimental scripts, and packet carvers, we considered an auto-user process to assist SMB generation. Our approach was to use ML and study the network traffic generated by predicting what the user was doing.

### A. SIMULATING SMB NETWORK TRAFFIC

The *"Fileopenrandom.py"* Python script performed very well when generating SMB traffic. However, it did not account for a few factors, such as a user running multiple instances of Word or Excel files concurrently. It did not account for writing to different files simultaneously or having numerous files opened for prolonged periods. Our model might have determined different results if this was the case. For example, the *Fileopenrandom.py* script opened files for a set amount of time; we chose a random time between 5 to 15 minutes. In a production environment, documents can be left open for hours at a time, and this can cause additional data to be generated in the packet capture, potentially skewing results.

The network environment is straightforward, using the SMB servers, clients, and the network switch. In this type of experimental environment, there was little change to the packet headers. In a production environment, the packet headers may change because additional network devices are used, such as firewalls and routers.

Another aspect that limited our research was declining the usage of PySMB<sup>14</sup>, which is an experimental SMB/CIFS (Common Internet File System). We opted to simulate user behavior through our script and not introduce another capability such as PySMB that we were unfamiliar with regarding how it interacted with the host and network.

We also did not select network traffic generators such as Smartbear, AutoIT, and Sikuli. These applications can accept and record user actions on a host and simulate network traffic. We chose not to use these because we did not fully understand what was occurring “under the hood” or its effect on the network.

Finally, we also chose not to use packet carvers<sup>15</sup>. Opting to select a packet carver and manipulate network traffic would not lend credence to establishing the ground truth by working the dependent variables within our research. Packet carvers may still be an asset in future research methods. For example, is it possible to carve a packet to resemble an SMB download command? Can ML determine the difference between an authenticate SMB packet compared to a carved SMB packet?

## **1. Limitations with the Classification Model with Encryption**

The DoDIN uses numerous IDS/IPS solutions and encryption methods that change some of the packet headers' information. Some solutions use a brokerage method to break the encryption chain and inspect network traffic. The effects of the brokerage can alter the information in the packet headers after it is re-encrypted. Baselineing the network with devices that broker network connections may still allow for ML to determine classification.

Furthermore, the security appliances that the DoDIN uses when access resources that are geographically separated. These packets will go through multiple network security devices and require a robust effort to achieve a baseline.

---

<sup>14</sup> More information about PySMB can be found at <https://pypi.org/project/pysmb/>.

<sup>15</sup> Packet carvers are applications that allows a user to adjust and make modifications to network packets to create a desired effect. Typically, this done to test network appliances for network attacks such as observing a Tear Drop attack.

This leads to another dimension to consider when carrying on this research further. Observing SMB version 3 (SMB3) uses end-to-end encryption to protect the payload's confidentiality, but not the TCP headers. The encryption prevents reading the SMB commands in the PCAP. In SMB2 traffic, we observed the various SMB commands used. For instance, the Read, Write, IOTCL, GetInfo, Find, etc., are present in the PCAP. SMB3 encrypts the network traffic, and the SMB commands are no longer visible; therefore, leveraging supervised learning was not achievable for the targets that we were observing.

## **2. Classification usage in an Enterprise Environment**

ML can use PCAPs from enterprise file-sharing solutions to provide classifications as well. For instance, cloud storage applications like Google Drive, Dropbox, and Microsoft SharePoint can provide classifications from their PCAPs. These services may or may not use the SMB protocol; however, it is possible to understand what primary file operations are being used by observing targets in a classification model such as uploading and downloading files.

Connecting to these services may occur over Transport Layer Security (TLS) when using port 443 (Hypertext Transfer Protocol Secure, or HTTPS) when using web browsers. As discussed earlier, TLS uses encryption to protect the confidentiality of the payload.

Google Drive and Dropbox can also be installed on local machines as applications to allow users to upload and download documents from their desktop. Some users may use this method to circumvent the security policies that prevent accessing private file-sharing sites. These applications may perform differently from SMB and web-based applications, but they provide another aspect to consider when observing file-sharing behavior on the network.

Data security is the focal point for the DOD. The DOD heavily relies upon file-sharing capabilities like SharePoint when it comes to data sharing and file collaboration. Observing SharePoint activity and correlating that with PCAPs could increase security awareness for malicious behavior. For example, a disgruntled user downloading numerous documents can aggregate the documents' overall classification, potentially putting national security at risk.

Cybersecurity professionals would cheerfully accept ML as a new tool to predict malicious behavior before the incident occurs. Combing through security logs and conducting deep packet inspection is tedious. Still, leveraging ML would alleviate a severe amount of work effort to allow these individuals to maintain focus on defending the network.

## **B. CONCLUSION**

Our research using ML to observe and predict SMB operations was exciting to conduct and watch. The DOD can significantly enhance any network operations center and provide additional coverage for security managers to achieve rapid incident handling and response. More importantly, the time it takes to perform deep packet inspection may not have to occur as often with manual labor as when using ML.

As the DOD moves forward and continues to develop Artificial Intelligence and Machine Learning (AIML) techniques and procedures, this capability will continue to expand further and provide invaluable services. More importantly, seeing the adoption of ML in the Cyber Mission Force<sup>16</sup> will significantly enhance the United States' posture in the Cyber domain for many years to come. The DOD should continue to invest in training and produce cyber warriors with technical skills and education that will exploit the usage of ML to defend the DoDIN and U.S. national interest at home and abroad.

---

<sup>16</sup> The Cyber Mission Force is the U.S. Cyber Command's operational arm that performs offensive and defensive operations and activities in and through cyberspace.

## APPENDIX. SOURCE CODES

### A. SMB GENERATIONS

```
import os, psutil, random, shutil, csv
from smb.SMBHandler import SMBHandler
from random import randint
from time import sleep, time
from datetime import datetime

#appendgarbage function is just to call when a file needs to have
#garbage padded to it for it to increase in size
def appendgarbage():
    padchar
    ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789") =

    return padchar

#lastfileaction function is used to determine the last action on a file.
# "A" - Last time file was accessed/read
# "M" - Last time file was modified
# "C" - Last time meta data was changed on file (Linux), or
# "C" - The creation time of the file (Microsoft)
#Note, if the creation time is greater than the accessed time, it possibly means
#that the file was copied from another location and then pasted in which it would
#get a new time creation stamp.
def lastfileaction(mystat):
    if mystat.st_atime >= mystat.st_ctime and mystat.st_atime >= mystat.st_mtime:
        LastFileStatus = "A"
    elif mystat.st_mtime >= mystat.st_ctime and mystat.st_mtime >= mystat.st_atime:
        LastFileStatus = "M"
    elif mystat.st_ctime >= mystat.st_atime and mystat.st_ctime >= mystat.st_mtime:
        LastFileStatus = "C"
    else:
        LastFileStatus = "NOOP"
    return LastFileStatus

#timeopen function captures the current time when called
#for operations that are opened
def timeopen():
    timenow = datetime.now()
    currenttime = timenow.strftime("%H:%M:%S")
    return currenttime
```

```

#timeclosed function captures the current time when called
#for operation to close out
def timeclose():
    timenow = datetime.now()
    currenttime = timenow.strftime("%H:%M:%S")
    return currenttime

def closefile():
    #try closing randomFile after being opened
    try:
        #kill MS applications that are opened after the timer expires
        for process in (process for process in psutil.process_iter()):
            if process.name()=="WORDPAD.EXE":
                process.kill()
            elif process.name()=="WINWORD.EXE":
                process.kill()
            elif process.name()=="Microsoft.Photos.exe":
                process.kill()
            elif process.name()=="msedge.exe":
                process.kill()
            elif process.name()=="Video.UI.exe":
                process.kill()
            elif process.name()=="POWERPNT.EXE":
                process.kill()
            elif process.name()=="EXCEL.EXE":
                process.kill()
            elif process.name()=="notepad.exe":
                process.kill()
            elif process.name()=="AcroRd32.exe":
                process.kill()
        #if process does not exist anymore (program quit), then pass
        except psutil.NoSuchProcess:
            pass

    return

#fileopener function random selects an operation (read, open, download, upload & lateral
copy
#to simulate user operations on the SMB share drive.
#mystat.st_ctime is creation time (windows) or meta data change (unix)
#mystat.st_atime is last time file was read/accessed but not changed
#mystat.st_mtime is last time file was changed
#mystat.st_size is Size of the file in bytes, if it is a regular file or a symbolic link.
#mystat.st_mode is file type and file mode bits (permissions)
#LastFileStatus shows if file was modified, accessed/read, or created (windows)/meta
changed (unix)

```

```

#print(randomFile, LastFileStatus, mystat.st_mode, mystat.st_size, mystat.st_ctime,
mystat.st_mtime, mystat.st_mtime)
#APPEND TO CSV FILE
def fileopener(logFile, StartTime, CloseTime):

    #for loop through specified directory and copy append names of files into list
    for root, dirs, files in os.walk(r"Y:\Documents\Data", topdown=True):
        #go through each file
        for file in files:

            randomFile = os.path.join(root, file)

            #identify the file extension to be used for the next block of code
            fileType = os.path.splitext(randomFile)

            #random choice determines the type of operation to be conducted
            randomChoice = [1,2,3,4,5]

            #Read and/or write from SMB share drive. If .txt or .docx, write to file, else
            #read from file
            if random.choice(randomChoice) == 1:
                #if extension is .txt or .docx, it will be padded with garbage data
                #simulate a user adding data to it.
                if fileType[-1] == ".txt" or fileType[-1] == ".docx":
                    #set time now
                    sysTime = time()
                    #random value will determine how much garbage data is written to file
                    randomLimit = random.randrange(50,500)
                    xCount = 0
                    #while loop will call function to append garbage data, then close
                    while xCount <= randomLimit:
                        addchar = appendgarbage()
                        updateFile = open(randomFile, 'a')
                        updateFile.write (addchar)
                        updateFile.write ("\n")
                        xCount += 1
                    updateFile.close()
                    #open file through its native application and record action
                    fileAction = "Write"
                    os.startfile(randomFile)
                else:
                    fileAction = "Read"
                    #set time now and open file to read
                    sysTime = time()
                    os.startfile(randomFile)

```

```

#get file status after data has been read/written to
mystat = os.stat(randomFile)

#return last action on that file (R accessed, M modified, C created)
LastFileStatus = lastfileaction(mystat)

#time set to leave application open for random sleep timer
sleep(randint(StartTime, CloseTime))
#print results to csv file
with open(logFile, 'a', newline='') as csvfile:
    filewriter = csv.writer(csvfile,
delimiter=",", quotechar='"', quoting=csv.QUOTE_MINIMAL)

filewriter.writerow([randomFile,sysTime,fileAction,LastFileStatus,mystat.st_mode,mysta
t.st_size, \
                    mystat.st_ctime,mystat.st_atime,mystat.st_mtime])
#print results to terminal
print(randomFile, sysTime, fileAction, LastFileStatus, mystat.st_mode,
mystat.st_size,mystat.st_ctime, \
      mystat.st_atime, mystat.st_mtime)
#close csv file
closefile()

#read operation - open file from SMB share drive
elif random.choice(randomChoice) == 2:
    sysTime = time()
    os.startfile(randomFile)
    fileAction = "Read"

#get file status after data has been written to it
mystat = os.stat(randomFile)

#return last action on that file (R accessed, M modified, C created)
LastFileStatus = lastfileaction(mystat)

#time set to leave Word open for 7 to 20 seconds
sleep(randint(StartTime, CloseTime))
#write results to csv file
with open(logFile, 'a', newline='') as csvfile:
    filewriter = csv.writer(csvfile,
delimiter=",", quotechar='"', quoting=csv.QUOTE_MINIMAL)

filewriter.writerow([randomFile,sysTime,fileAction,LastFileStatus,mystat.st_mode,mysta
t.st_size, \
                    mystat.st_ctime,mystat.st_atime,mystat.st_mtime])

```

```

        #print results to terminal
        print(randomFile, sysTime, fileAction, LastFileStatus, mystat.st_mode,
mystat.st_size,mystat.st_ctime, \
            mystat.st_atime, mystat.st_mtime)
        #close csv file
        closefile()

    #Later copy operation, move one file from SMB share drive location to another on
    SMB share drive
    elif random.choice(randomChoice) == 3:
        sysTime = time()
        shutil.copy2(randomFile,r'Y:\Documents\Upload')
        fileAction = "Lateral Copy"

        #pull os.stat on randomFile after processing
        mystat = os.stat(randomFile)

        #return last action on that file (R accessed, M modified, C created)
        LastFileStatus = lastfileaction(mystat)

        #write results to csv file
        with open(logFile, 'a', newline=") as csvfile:
            filewriter = csv.writer(csvfile,
delimiter=",", quoting=csv.QUOTE_MINIMAL)

        filewriter.writerow([randomFile,sysTime,fileAction,LastFileStatus,mystat.st_mode,mysta
t.st_size, \
            mystat.st_ctime,mystat.st_atime,mystat.st_mtime])
        #print results to terminal
        print(randomFile, sysTime, fileAction, LastFileStatus, mystat.st_mode,
mystat.st_size,mystat.st_ctime, \
            mystat.st_atime, mystat.st_mtime)

    #Download operation - download file from SMB share drive to local drive and get
    system time
    elif random.choice(randomChoice) == 4:
        sysTime = time()
        shutil.copy2(randomFile,r'C:\Temp')
        fileAction = "Download"

        #pull os.stat on randomFile after processing
        mystat = os.stat(randomFile)

        #return last action on that file (R accessed, M modified, C created)
        LastFileStatus = lastfileaction(mystat)

```

```

        #write results to csv file
        with open(logFile, 'a', newline='') as csvfile:
            filewriter = csv.writer(csvfile,
delimiter=",",quotechar='"',quoting=csv.QUOTE_MINIMAL)

filewriter.writerow([randomFile,sysTime,fileAction,LastFileStatus,mystat.st_mode,mysta
t.st_size, \
                    mystat.st_ctime,mystat.st_atime,mystat.st_mtime])
        #print results to terminal
        print(randomFile, sysTime, fileAction, LastFileStatus, mystat.st_mode,
mystat.st_size,mystat.st_ctime, \
            mystat.st_atime, mystat.st_mtime)

#Upload operation - upload file from local drive to SMB share drive
elif random.choice(randomChoice) == 5:
    localFile = random.choice(os.listdir(r'C:\Temp'))
    #pull os.stat on randomFile after processing
    mystat = os.stat(randomFile)

    #copy from local drive to SMB share drive and get system time
    sysTime = time()
    shutil.copy2(os.path.join(r'C:\Temp', localFile),r'Y:\Documents\Upload')
    fileAction = "Upload"

    #return last action on that file (R accessed, M modified, C created)
    LastFileStatus = lastfileaction(mystat)

    #write results to csv file
    with open(logFile, 'a', newline='') as csvfile:
        filewriter = csv.writer(csvfile,
delimiter=",",quotechar='"',quoting=csv.QUOTE_MINIMAL)

filewriter.writerow([randomFile,sysTime,fileAction,LastFileStatus,mystat.st_mode,mysta
t.st_size, \
                    mystat.st_ctime,mystat.st_atime,mystat.st_mtime])
        #print results to terminal
        print(randomFile, sysTime, fileAction, LastFileStatus, mystat.st_mode,
mystat.st_size,mystat.st_ctime, \
            mystat.st_atime, mystat.st_mtime)

#time set to delay parent loop for 10 to 15 seconds to buffer out processing time
sleep(randint(5,30))

```

```

#####
#   Main Program
#####

#get input log file name from user
logFile = str(input("Enter name of CSV log file to output to (e.g. save.csv): "))

#get start and close sleep time from user that determines how long an application stays
open
print("Configure random time range in seconds...> ")
StartTime = int(input("Enter the start time of sleep: "))
CloseTime = int(input("Enter the stop time for sleep: "))

while StartTime >= CloseTime:
    print("Incorrect time range... ")
    StartTime = int(input("Please enter correct start sleep time: "))
    CloseTime = int(input("Please enter correct stop sleep time: "))

print
('Filename\t','SysTime\t','FileAct\t','Fstat\t','FMode\t','FSize\t','CreateTime\t','AccessTime
\t','ModifyTime')

#CREATE AND APPEND TO CSV FILE
with open(logFile, 'a', newline="") as csvfile:
    filewriter = csv.writer(csvfile,
delimiter=",", quotechar='|', quoting=csv.QUOTE_MINIMAL)

filewriter.writerow(['Filename','SysTime','FileAction','FileStatus','FileMode','FileSize','Cr
eateTime','AccessTime','ModifyTime'])

#pass in arguments and call function to process file opener
fileopener(logFile, StartTime, CloseTime)

```

## B. CLASSIFY ACTIONS

```
'Time_Delta_Prev_Displacement_mean',
'Time_Delta_Prev_Displacement_std',
'Window_Size_Mean',
'Window_Size_Std',
'Packet_Length_Mean',
'Packet_Length_Std',
'TCP_Len_Mean',
'TCP_Len_Std',
]
```

```
TARGETS = [
# 'Session_Setup',
# 'Session_Logoff',
# 'Tree_Connect',
# 'GetInfo',
# 'GetInfo_GetInfo',
# 'GetInfo_GetInfo_GetInfo',
'Create',
'Read',
'Write',
'Close',
# 'Find',
# 'Find_Find',
# 'Create_SetInfo',
# 'Create_Ioctl',
# 'Create_GetInfo',
# 'Create_GetInfo_GetInfo',
# 'Create_Find_Find',
# 'No_Op',
]
```

```
def main(seed=SEED):
    dfs = []
    for f in glob.glob('features/*.csv'):
        df = pd.read_csv(f)
        df['behavior'] = os.path.split(f)[1].split('_')[0]
        dfs.append(df)

    df = pd.concat(dfs, ignore_index=True)

    X = df[FEATURES].values
    y = df[TARGETS].values
    y_binary = (y > 0).astype(int)
    behaviors = df['behavior'].values
```

```

X_train, X_test, y_train_binary, y_test_binary, behavior_train, behavior_test =
train_test_split(X, y_binary, behaviors, train_size=0.5, random_state=seed)
clf = RandomForestClassifier(n_estimators=100, random_state=seed)
clf.fit(X_train, y_train_binary)
y_pred_binary = clf.predict(X_test)

report = classification_report(y_test_binary, y_pred_binary, digits=3,
target_names=TARGETS)
print(report)

with open('reports/overall_report.txt', 'w') as f:
    f.write(report)

for behavior in np.unique(behavior_test):
    idx = (behavior_test == behavior)
    X_test_behavior = X_test[idx]
    y_test_behavior = y_test_binary[idx]
    y_pred_behavior = clf.predict(X_test_behavior)

    report = classification_report(y_test_behavior, y_pred_behavior, digits=3,
target_names=TARGETS)
    print(report)

    with open('reports/%s_report.txt' % behavior, 'w') as f:
        f.write(report)

    freqs = y_pred_behavior.sum(axis=0)
    fig, ax = plt.subplots(1,1, figsize=(4,4))
    ar = np.arange(len(TARGETS))
    ax.bar(ar, freqs)
    ax.set_xticks(ar)
    ax.set_xticklabels(TARGETS)
    ax.set_xlabel('SMB Command')
    ax.set_ylabel('Frequency')
    plt.savefig('figures/%s_hist.pdf' % behavior)

if __name__ == '__main__':
    main()

```

## EXTRACT FEATURES

```
import os
import sys
import pandas as pd
```

```
WINDOW_SIZE = 30
```

```
def window_features(df):
    begin = df.iloc[0]['Time']
    end = df.iloc[-1]['Time']
```

```
    smbread = 'Read'
    smbwrite = 'Write'
    smbtreecon = 'Tree Connect'
    smbsessionsetup = 'Session Setup'
    smbsessionlogoff = 'Session Logoff'
    smbcreate = 'Create'
    smbgetinfo = 'GetInfo'
    smbgetinfo2 = 'GetInfo,GetInfo'
    smbgetinfo3 = 'GetInfo,GetInfo,GetInfo'
    smbfind = 'Find'
    smbfindfind = 'Find,Find'
    smbcreatesetinfo = 'Create,SetInfo'
    createioctl = 'Create,Ioctl'
    creategetinfo = 'Create,GetInfo'
    creategetinfo2 = 'Create,GetInfo,GetInfo'
    createfind2 = 'Create,Find,Find'
    smbclose = 'Close'
    noop = ''
```

```
f = {
    #'Time_mean': df['Time'].mean(),
    #'Time_std': df['Time'].std(),
    'Delta_Time_Mean': df['DeltaTime'].mean(),
    'Delta_Time_Std': df['DeltaTime'].std(),
    'Time_Delta_Prev_Cap_Frame_Mean': df['TimeDelta_prev_cap_frame'].mean(),
    'Time_Delta_Prev_Cap_Frame_Std': df['TimeDelta_prev_cap_frame'].std(),
    'Time_Delta_Prev_Disp_Frame_mean':
df['TimeDelta_prev_displayed_frame'].mean(),
    'Time_Delta_Prev_Disp_Frame_std': df['TimeDelta_prev_displayed_frame'].std(),
    'Window_Size_Mean': df['Window'].mean(),
    'Window_Size_Std': df['Window'].std(),
    'Packet_Length_Mean': df['Packet Len'].mean(),
    'Packet_Length_Std': df['Packet Len'].std(),
```

```

'TCP_Len_Mean': df['TCP Len'].mean(),
'TCP_Len_Std': df['TCP Len'].std(),
'Session_Setup': df['SMB CMD'].str.count(smbsessionsetup).sum(),
'Session_Logoff': df['SMB CMD'].str.count(smbsessionlogoff).sum(),
'Tree_Connect': df['SMB CMD'].str.count(smbtreecon).sum(),
'GetInfo': df['SMB CMD'].str.count(smbgetinfo).sum(),
'GetInfo_GetInfo': df['SMB CMD'].str.count(smbgetinfo2).sum(),
'GetInfo_GetInfo_GetInfo': df['SMB CMD'].str.count(smbgetinfo3).sum(),
'Create': df['SMB CMD'].str.count(smbcreate).sum(),
'Read': df['SMB CMD'].str.count(smbread).sum(),
'Write': df['SMB CMD'].str.count(smbwrite).sum(),
'Close': df['SMB CMD'].str.count(smbclose).sum(),
'Find': df['SMB CMD'].str.count(smbfind).sum(),
'Find_Find': df['SMB CMD'].str.count(smbfindfind).sum(),
'Create_SetInfo': df['SMB CMD'].str.count(smbcreatesetinfo).sum(),
'Create_Ioctl': df['SMB CMD'].str.count(createioctl).sum(),
'Create_GetInfo': df['SMB CMD'].str.count(creategetinfo).sum(),
'Create_GetInfo_GetInfo': df['SMB CMD'].str.count(creategetinfo2).sum(),
'Create_Find_Find': df['SMB CMD'].str.count(createfind2).sum(),
'No_Op': df['SMB CMD'].str.count(noop).sum(),

#look for rows that fall between beginning and end of window
#add #of file reads, writes, etc. columns (one column for #write)
#ensure the time units in the csv are the same in the log file****
}
return f

def main(fname_in, fname_out):
df = pd.read_csv(fname_in, index_col=0)

# upstream = df[df['Dst Port']==445]
# downstream = df[df['Src Port']==445]

features = []
for i in range(0, len(df)-WINDOW_SIZE, WINDOW_SIZE):
    features.append(window_features(df.iloc[i:(i+WINDOW_SIZE)]))

features = pd.DataFrame(features)
features.index.name = 'window'
features.to_csv(fname_out)
#print(features)

```

```

if __name__ == '__main__':
    main('2019MassiveDelete.csv', '2019MassiveDelete_features.csv')
    main('2019MassiveDownload.csv', '2019MassiveDownload_features.csv')
    main('2019MassiveLateralCopy.csv', '2019MassiveLateralCopy_features.csv')
    main('2019MassiveUpload.csv', '2019MassiveUpload_features.csv')

```

## FE EXTRACTION SMB COUNT

```

import os
import sys
import pandas as pd

#df = pd.read_csv(r"23NOV5_15pcap.csv")
#pd.options.display.max_columns = None
#print(df)

WINDOW_SIZE = 100

def window_features(df):
    begin = df.iloc[0]['Time']
    end = df.iloc[-1]['Time']

    smbread = 'Read'
    smbwrite = 'Write'
    smbtreecon = 'Tree Connect'
    smbsessionsetup = 'Session Setup'
    smbsessionlogoff = 'Session Logoff'
    smbcreate = 'Create'
    smbgetinfo = 'GetInfo'
    smbgetinfo2 = 'GetInfo,GetInfo'
    smbgetinfo3 = 'GetInfo,GetInfo,GetInfo'
    smbfind = 'Find'
    smbfindfind = 'Find,Find'
    smbcreatesetinfo = 'Create,SetInfo'
    createioctl = 'Create,Ioctl'
    creategetinfo = 'Create,GetInfo'
    creategetinfo2 = 'Create,GetInfo,GetInfo'
    createfind2 = 'Create,Find,Find'
    smbclose = 'Close'
    noop = ''

    f = {
        #'Time_mean':df['Time'].mean(),
        #'Time_std':df['Time'].std(),

```

```

'Delta_Time_Mean': df['DeltaTime'].mean(),
'Delta_Time_Std': df['DeltaTime'].std(),
'Time_Delta_Prev_Cap_Frame_Mean': df['TimeDelta_prev_cap_frame'].mean(),
'Time_Delta_Prev_Cap_Frame_Std': df['TimeDelta_prev_cap_frame'].std(),
'Time_Delta_Prev_Displ_Frame_mean':
df['TimeDelta_prev_displayed_frame'].mean(),
'Time_Delta_Prev_Displ_Frame_std': df['TimeDelta_prev_displayed_frame'].std(),
'Window_Size_Mean': df['Window'].mean(),
'Window_Size_Std': df['Window'].std(),
'Packet_Length_Mean': df['Packet Len'].mean(),
'Packet_Length_Std': df['Packet Len'].std(),
'TCP_Len_Mean': df['TCP Len'].mean(),
'TCP_Len_Std': df['TCP Len'].std(),
'Session_Setup': df['SMB CMD'].str.count(smbsessionsetup).sum(),
'Session_Logoff': df['SMB CMD'].str.count(smbsessionlogoff).sum(),
'Tree_Connect': df['SMB CMD'].str.count(smbtreecon).sum(),
'GetInfo': df['SMB CMD'].str.count(smbgetinfo).sum(),
'GetInfo_GetInfo': df['SMB CMD'].str.count(smbgetinfo2).sum(),
'GetInfo_GetInfo_GetInfo': df['SMB CMD'].str.count(smbgetinfo3).sum(),
'Create': df['SMB CMD'].str.count(smbcreate).sum(),
'Read': df['SMB CMD'].str.count(smbread).sum(),
'Write': df['SMB CMD'].str.count(smbwrite).sum(),
'Close': df['SMB CMD'].str.count(smbclose).sum(),
'Find': df['SMB CMD'].str.count(smbfind).sum(),
'Find_Find': df['SMB CMD'].str.count(smbfindfind).sum(),
'Create_SetInfo': df['SMB CMD'].str.count(smbcreatesetinfo).sum(),
'Create_Ioctl': df['SMB CMD'].str.count(createioctl).sum(),
'Create_GetInfo': df['SMB CMD'].str.count(creategetinfo).sum(),
'Create_GetInfo_GetInfo': df['SMB CMD'].str.count(creategetinfo2).sum(),
'Create_Find_Find': df['SMB CMD'].str.count(createfind2).sum(),
'No_Op': df['SMB CMD'].str.count(noop).sum(),

```

```

#look for rows that fall between beginning and end of window
#add #of file reads, writes, etc. columns (one column for #write)
#ensure the time units in the csv are the same in the log file****
}
print(f)
return f

```

```

def main(fname_in, fname_out):
df = pd.read_csv(fname_in, index_col=0)

# upstream = df[df['Dst Port']==445]

```

```

# downstream = df[df['Src Port']==445]

features = []
for i in range(len(df)-WINDOW_SIZE):
    features.append(window_features(df.iloc[i:(i+WINDOW_SIZE)]))

features = pd.DataFrame(features)
features.index.name = 'window'
features.to_csv(fname_out)
#print(features)

"""
if __name__ == '__main__':
    # usage: $ python extract_features.py <input> <output>

    fname_in = sys.argv[1]
    fname_out = sys.argv[2]
"""

fname_in = str(input("enter csv file: "))

fname_out = str(input("enter output file: "))

main(fname_in, fname_out)

```

## LIST OF REFERENCES

- Asiri, S. (2018). *Machine learning classifiers*. Towards Data Science. <https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623>
- Bhukya, W. N., Kommuru, S. K., & Negi, A. (2007). Masquerade detection based upon GUI user profiling in Linux systems. In *Annual Asian Computing Science Conference* (pp. 228-239). Springer, Berlin, Heidelberg. [https://link.springer.com/chapter/10.1007/978-3-540-76929-3\\_21](https://link.springer.com/chapter/10.1007/978-3-540-76929-3_21)
- Colbaugh, R. & Glass, K. (2011). Proactive defense for evolving cyber threats. *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics* (pp. 125-130). IEEE. <https://www.osti.gov/servlets/purl/1108387>
- Crumpler, W. & Lewis, J.A. (2019). *The cybersecurity workforce gap*. The Center for Strategic & International Studies. <https://www.csis.org/analysis/cybersecurity-workforce-gap>
- Davison, B. D. & Hirsh, H. (1998, July). *Predicting sequences of user actions* [Workshop]. Predicting the Future: AI Approaches to Time-Series Analysis (pp. 5-12). Rutgers, NJ, United States. <http://www.cs.cornell.edu/~hirsh/papers/1998/aaai-tsws1.pdf>
- DeepAI (n.d.a). *Kernel density estimation*. DeepAI. <https://deepai.org/machine-learning-glossary-and-terms/kernel-density-estimation>
- DeepAI (n.d.b). *Neural network*. DeepAI. <https://deepai.org/machine-learning-glossary-and-terms/neural-network>
- Dhammi, A. & Singh, M. (2015). Behavior analysis of malware using machine learning. In *2015 eighth international conference on contemporary computing (IC3)* (pp. 481-486). IEEE. <https://ieeexplore.ieee.org/document/7346730>
- Fraley, J. B. & Cannady, J. (2017). The promise of machine learning in cybersecurity. *SoutheastCon 2017* (pp. 1-6). IEEE. <https://ieeexplore.ieee.org/abstract/document/7925283>
- Garbade, M.J. (2018). *Understanding k-means clustering in machine learning*. Towards Data Science. <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>
- Google Developers (2020). *Classification: true vs. false and positive vs. negative*. [Ghttps://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative](https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative)

- McKinney, S. (2008). *Insider threat: user identification via process profiling* [Master's thesis, North Carolina State]. NC State Libraries: <https://repository.lib.ncsu.edu/handle/1840.16/1902>
- Masduki, B. W., Ramli, K., Saputra, F. A., & Sugiarto, D. (2015). Study on implementation of machine learning methods combination for improving attacks detection accuracy on intrusion detection system (IDS). *2015 International Conference on Quality in Research (QiR)* (pp. 56-64). <https://ieeexplore.ieee.org/document/7374895>
- Mayhew, M., Atighetchi, M., Adler, A., & Greenstadt, R. (2015). Use of machine learning in big data analytics for insider threat detection. *MILCOM 2015-2015 IEEE Military Communications Conference* (pp. 915-922). [https://www.researchgate.net/publication/281558896\\_Use\\_of\\_Machine\\_Learning\\_in\\_Big\\_Data\\_Analytics\\_for\\_Insider\\_Threat\\_Detection](https://www.researchgate.net/publication/281558896_Use_of_Machine_Learning_in_Big_Data_Analytics_for_Insider_Threat_Detection)
- Pressgrove, J. (2020). *DoD agency suffers data breach, potentially compromising SSNs*. Military.com. <https://www.military.com/daily-news/2020/02/25/dod-agency-suffers-data-breach-potentially-compromising-ssns.html> Microsoft (2018, May 31). *Process and threads*. <https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads?redirectedfrom=MSDN>
- Shavlik, J., Shavlik, M., & Fahland, M. (2001). *Abstract evaluating software sensors for actively profiling windows 2000 computer users* [Presentation]. Fourth International Symposium on Recent Advances in Intrusion Detection. <https://ftp.cs.wisc.edu/machine-learning/shavlik-group/shavlik.raid01.pdf>
- Sharp, T. (2020). *An introduction to support vector regression*. Towards Data Science. <https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2>
- SciKit Learn. (n.d.). *Support vector machines*. <https://scikit-learn.org/stable/modules/svm.html>
- Udoeyop, A. W. (2010). *Cyber profiling for insider threat detection* [Master thesis, University of Tennessee]. Tennessee Research and Creative Exchange. [https://trace.tennessee.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1709&context=utk\\_gradthes](https://trace.tennessee.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1709&context=utk_gradthes)
- Varmazyar, M. (2016). *What is fork bomb and how can you prevent this danger?* Unixmen. <https://www.unixmen.com/fork-bomb-can-prevent-danger/>

## **INITIAL DISTRIBUTION LIST**

1. Defense Technical Information Center  
Ft. Belvoir, Virginia
2. Dudley Knox Library  
Naval Postgraduate School  
Monterey, California