Real Time Network Threat Monitoring using Wireshark

CSCE 522 - Fall 2024

Rhaynie Bongiorno - bongiors@email.sc.edu
Baker Womack - bwomack@email.sc.edu
Andy Smith - ats29@email.sc.edu
Jacob Sherer - jsherer@email.sc.edu
Ryan Malone - rpmalone@email.sc.edu
Shawn Tan - sktan@email.sc.edu
Bradley Zenker - bzenker@email.sc.edu
Nazire Brown - nlb1@email.sc.edu
Christine Steege - csteege@email.sc.edu
Cameron Washington - csw11@email.sc.edu

December 10th, 2024

Executive Summary

Our group was tasked with identifying a current limitation in security technologies, understanding the impact of this limitation, and developing a solution. Computer networks are the informational conduits that support a majority of personal and business activities. This makes networks a critical point of security. Effective network security relies on the ability to quickly detect and respond to threats. Currently, the response of security operators is limited by knowledge disparity and volume of traffic to inspect. This project seeks to improve these issues by providing tooling to security analysts. Our solution consists of an automated traffic analysis script, utilizing Python's PyShark library to streamline the detection of malicious network activity such as DDoS and malware attacks. This tool improves the abilities of tools like Wireshark, aiming to increase the efficiency of analysts and network security.

Our program analyzes packet capture files and applies custom Wireshark filters to detect malicious patterns and reduce the amount of legitimate traffic that analysts have to sift through. The solution is modular, allowing new functions to be written to address evolving attacks and security concerns. The presented script supports two primary functions: identifying DDoS attacks and detecting malware. The "detect_ddos" function leverages Wireshark filters to target common DDoS attack vectors such as SYN flood, UDP flood, ICMP flood, and large frame attacks. If suspicious traffic exceeds analyst defined thresholds, an alert with relevant information will be generated. The "detect_mal_commands" function similarly applies filters to check for network activity consistent with malware. This includes anomalous TCP port usage, Cobalt Strike BEACON traffic patterns, and DNS queries to possibly malicious domains. The malware patterns are also modular to allow analysts to expand the program to detect emerging threats. This project was designed with scalability in mind. For example, the incorporation of PyShark's LiveCapture feature would allow the script to process real-time traffic, enabling security operators to continuously monitor specific segments of a network environment. Additionally, the program has the opportunity to be integrated into analysts' existing alerting system and offers thresholds that can be dynamically tuned to meet an organization or threat actors traffic patterns.

We have demonstrated that packet analysis tools like Wireshark can be automated to improve the efficacy of security analysts by enhancing network visibility. Filters reduce the volume of traffic that analysts are presented with and encapsulate an expert level analyst knowledge. Ongoing development could turn our approach into a comprehensive tool for detecting and responding to cyber threats, especially in cases of a high network traffic volume. Our project thrives in an environment where modules are continuously updated to improve the tools detection capabilities with threat intelligence feeds and advanced filtering techniques. Additional features could include machine learning-based anomaly detection, live traffic analysis, and automated incident reporting.

Introduction

The area of research that was pursued for this project was defensive packet sniffing. While in the world of cyber security there are a large number of attacks such as DNS spoofing, ARP spoofing, DDoS attacks and many more. Analysts use tools to be able to look at the individual packets to be able to determine if they are malicious. However, this can be an issue due to a knowledge gap for the researcher or potentially the size of the network and how much traffic is flowing. Through the analysis of these packets many defenders can detect attacks not only early, but can also uncover unseen attacks. The tool that is most commonly used is wireshark. Wireshark, a powerful, open-source network protocol analyzer, has become one of the most widely used tools for network security professionals and researchers and supports a large number of network protocols and applications. It allows users to capture, analyze, and apply filters to isolate malicious activities, which is critical for identifying security threats and ensuring network security. However, while widely used, Wireshark also faces various limitations when being implemented into real-world networks for real-time threat monitoring.

Related Work

There has been much research over the past decade examining Wireshark's capabilities in defensive packet sniffing, network forensics, intrusion detection, and real-time threat mitigation. By analyzing packets sent across a network, it is possible to detect an attack of intrusion early on, which could help prevent any long-lasting effects from occurring.

This idea of searching for attacks when they first take place is known as an intrusion detection system (IDS). Since there are so many attacks that are possible over a network, the intrusion detection system may need to look for multiple different things to determine if an attack could be taking place. By analyzing things on a network such as port selection, exploit utilization, network functionality disruptions, and any system manipulation via backdoors, the IDS can identify an attack (these technical details are what make up an attack's “signature”) (Beridze & Donadze, 2024). Once an attack is identified, the administrators can work to put a stop to it. By inspecting the attack, it is possible to determine what the attack is doing, how it’s behaving, and what it is searching for (or its main goal). With this information, private data and networks can be kept safe by securing what the attack is after. Wireshark can be used to identify and inspect these types of packets and behaviors. By seeing the “flow” of packets across a network, any suspicious, or abnormal, activities can be flagged and result in a quicker response rate to an active attack (Shaw & Parveen, 2024). This ensures that less data is taken and evasive actions can be dispatched before an attack has too much time to wreak havoc on a network or system.

Wireshark’s powerful packet-capturing and filtering capabilities make it indispensable for identifying and mitigating network attacks. Wireshark has been proven to be effective in identifying ongoing attacks, such as ARP or DNS spoofing, by applying filters to detect abnormal traffic patterns in real-time, and its open-source nature allows customization, making it adaptable for real-time alerts and integration with other security tools (Banerjee et al., 2010; Iqbal & Naaz, 2019). Several other types of network threats have also been explored, such as port scanning, ICMP-based attacks, covert FTP and IRC channels, and denial-of-service (DDoS) attacks (Mabsali et al., 2023; Ndatinya et al., 2015). By using Wireshark's packet filtering features, network administrators can monitor and analyze packets that reveal signs of these attacks. For example, port scanning, which attackers use to identify vulnerable services on a network, can be detected by Wireshark through specific patterns of SYN packets that do not complete a full TCP handshake (Ndatinya et al., 2015).

In addition to detecting direct attacks, research has also demonstrated how Wireshark can be instrumental in enhancing LAN security by identifying potential vulnerabilities in protocols like HTTP. By capturing and analyzing HTTP packets, Wireshark can help mitigate credential theft by identifying potential network vulnerabilities or security breaches (Hussain et al., 2024). This proactive analysis allows security teams to address misconfigurations or poor implementations before they are exploited by bad actors, further strengthening the overall security of networks.

More recent studies have also evaluated the effectiveness of combining Wireshark's data ingestion and packet analysis capabilities with machine learning algorithms. The addition of machine learning enabled the ability for systems to detect previously unknown attacks by learning from network traffic patterns, which resulted in greater accuracy in identifying attacks such as DDoS attacks (Chandravathi et al., 2024).

Overall, Wireshark's versatility in network forensics, intrusion detection, and vulnerability identification, positions it as a key tool in cybersecurity, with the potential for further advancements to improve its handling of large-scale traffic for future implementation in real-time threat detection.

Background Information

Defensive packet sniffing is a critical component of network security, especially in today's increasingly hostile digital environment. The ability to detect and analyze network traffic in real-time is important for identifying threats like DNS spoofing, ARP spoofing, and Distributed Denial of Service (DDoS) attacks. Despite how useful packet sniffing tools such as Wireshark are for analyzing network traffic, it is not an IDS. By default, the software is not able to determine whether or not traffic is malicious. That determination is up to the security analyst and the protocols or filters they implement. Furthermore, Wireshark is not built for automation or long-term monitoring. Packet monitoring sessions have to be initiated by the analyst or by a script using the Command Line Interface (CLI) tool. The tool is limited by the filters and protocols set by the analyst which still require manual intervention.

Group Research

Our project leverages and expands on Wireshark's capabilities to filter out malicious activities more efficiently and handle large volumes of network traffic and aid in the real-time detection of suspicious traffic patterns. We also demonstrate how Python can be used to automate the process of capturing and analyzing packets. To successfully complete our project for commercial use, we would require a few resources, including developers who can automate the reading, analysis, and detection of attacks through software (such as a more robust Python script), cybersecurity experts which are essential to ensure we develop comprehensive filters that can detect a wide range of traditional and sophisticated attacks, infrastructure to support our continuous updates to filters, automation software, and detection algorithms. In addition, we would need financial contributions to get our idea fully up and running. The cost of hiring all of these subject-matter experts would be high, and hosting updates can quickly become substantial. In addition, it would be smart for us to invest in a legal department that ensures we will not be held liable for any misuse of the software. Because detecting and analyzing packets could put personal information in jeopardy, and the risk for misuse on networks owned by other organizations is *not* negligible, it's important that there are many disclaimers in our user agreements to ensure we will not be held liable for misuse of our software.

WireShark Filters

Our solution relies on several different powerful technologies to streamline the detection of potential network attacks. The foundation of our solution are the filters that were originally developed to work with WireShark. These filters will take all of the packets that were found on the network and filter through them to display only the ones that would be relevant to specific types of attacks. These filters are unique in that they will help analysts quickly identify an attack among possible thousands of different packets on a network at any given time. Using these filters, it can be demonstrated how it can filter for two attacks, DDOS and malware.

DDOS

DDOS attacks are a critical concern for network security analysts, with high amounts of large traffic going across the network to overwhelm systems. These attacks frequently leverage common network protocols such as the TCP protocol's handshake to establish authentication, when this is leveraged it is a SYN flood. Other variations include UDP (User Data Protocol) floods using large UDP packets to overwhelm a target system or ICMP (Internet Control Message Protocol) floods where the attacker overwhelms a target device with large numbers of pings.

In this analysis, the PCAP file used to create and test the filter has normal traffic mixed with these DDOS attacks. This filter identifies 1658 packets of the 4975 total as potentially relevant DDOS packets.

(tcp.flags.syn == 1 && tcp.flags.ack == 0) || (udp && frame.len > 1000) || icmp.type == 8 || icmp.type == 0) || frame.len > 1500

This filter specifically searches for SYN, UDP, and ICMP-based DDOS attacks in a given PCAP. The first set identifies TCP SYN flood attacks which consist of a malicious user initiating large numbers of TCP handshakes, which consist of a client sending a SYN packet to a server which responds with a SYN-ACK packet leaving a connection open waiting on the final ACK packet from the client. The attack leverages this by never completing the handshake, leaving the ports open and waiting for a final ACK packet from the clients. This filter identifies any TCP handshake packets with the SYN flag that never had a corresponding ACK flag returned.

The second set in the filter identifies UDP flood DDOS attacks. UDP, as a connectionless protocol, does not leave connections open like TCP, meaning that if a UDP request can't be fulfilled the server sends a destination unreachable message through ICMP back to the client. A malicious actor exploits this by sending large UDP packets to overwhelm the storage and processing power of a network device along with the many ICMP destination unreachable messages that would need to be sent. This filter identifies this attack by looking at UDP packets of an unusually large size of greater than 1000 bytes.

The third set identifies ICMP flood DDOS attacks. The attack uses ICMP echo requests specifically which are used in the ping command. These echo requests are calls and responses between 2 network devices. The attack has a malicious actor send large quantities of ICMP echo requests which all need to have a requisite echo reply which overloads the server resources by needing to generate and send large numbers of packets. This filter identifies all ICMP echo requests (type 8) and ICMP echo replies (type 0).

The final part of this filter identifies any giant frames being sent as a final catch-all of resource-based DDOS attacks. Frames larger than 1500 bytes are flagged, as this is an unusual size and would likely be indicative of a volumetric attack on the network. This is a good check for an additional ICMP DDOS attack known as the ping of death.

Malware

Malware is a constant cat-and-mouse game, with most serving as a delivery method for dropping additional payloads, which are the larger issue. These payloads often will include command and control beacons (C2 beacons), which are going to be essential for attackers to establish persistence within the victim's computer. Within the realm of these beacons, the most common are going to be from a framework known as Cobalt Strike.

In this analysis, the specific PCAP that was used to assist with the filter creation and testing has roughly half normal traffic and half C2-related beacons. As such the filter that was used compressed the PCAP from 15105 down to 7338 relevant or suspicious packets.

```
((tcp.port == 80 || tcp.port == 443 || tcp.port == 2222 || tcp.port == 4444 || tcp.port == 8080 ||
tcp.port == 8443) && (frame.time_delta > 45 && frame.time_delta < 75)) ||
(ssl.handshake.extensions_server_name contains "cobalt" || ssl.handshake.type == 1 ||
ssl.record.content_type == 23) || (http.user_agent contains "Cobalt" || http.user_agent
contains "Mozilla" || http.request.uri contains "beacon" || http.request.uri contains "index" ||
http.request.uri contains "post" || http.request.uri contains "payload") || (frame.len > 600 &&
tcp.payload && tcp.stream) || (dns.qry.name contains ".onion" || dns.qry.name contains
".xyz" || dns.qry.name contains ".top" || dns.qry.name contains ".ru" || dns.qry.name contains
".club") || (tcp.window_size == 1024) || (tcp.analysis.retransmission)
```

The filter works by firstly, finding all of the most common ports for services, with the addition of commonly used ports from C2 beacons. Next, the filter checks frame times, which is how often the frame is sent from the previous spot. Many beacons are going to have a set interval, most commonly between these two times. Next, it finds different parts of the packet that contain cobalt strike names such as cobalt, beacon, or payloads. Specifically, the filter is going to check for the request URI, the user agent, and the SSL certificate and handshake. Any mention of this would be something that defenders will want to check. Searching for frames with a larger size helps to find when data is being sent back and forth. Finally, the filter searches for DNS names that are commonly used to host command and control servers. The top-level domain names or TLD, such as .onion refer to websites that are a part of the Tor network which allows for more anonymous hosting. Additionally, all of the other TLDs common for simple hosting are known to be cheap.

Potential for Automation Addressing the Research Gap

These Wireshark filters are crucial in helping to narrow down network traffic to identify specific attacks. However, the filters would need to be manually put into the Wireshark program, and it is still up to the analyst to go through the data and make a determination. To further optimize the analysis process and make the use of the filters far more practical, our team has constructed a potential automation solution.

To demonstrate this solution, our team wrote a simple Python script that utilizes the PyShark library, a Python wrapper for Wireshark's command-line counterpart, Tshark. This library allowed us to read network traffic through .pcap files, and apply the Wireshark filters that were developed to preprocess the information for only packets that may be relevant to specific types of attacks. Using the filtered results, we developed simple algorithms to check for evidence of an attack. For the sake of demonstration, if the software has detected that an attack may have taken place, it prints out the results, however, it could easily be expanded to log its findings, or could even notify individuals or departments of the results automatically.

By combining Python's versatility and WireShark's filtering system, our solution will enable fast and automated threat detection. Rather than the analyst having to manually apply each filter for each attack type, going through each of the filtered packets (of which could be thousands of packets), and determining if the activity is normal, software could do it automatically in a matter of seconds. With the time it takes to detect an attack reduced, and efficiency of detection rising, organizations could save both time and resources by acting early on any attack that they are subjected to.

In the future, our script could easily be slightly modified to run on a server that is online 24x7, making it possible to use resources such as the PyShark library's LiveCapture feature to automatically collect packets in real time for a certain time interval. The script would follow the same process of applying the Wireshark filters to the packets to determine whether an attack is or was taking place. To ensure the appropriate departments or individuals are notified promptly, the script could send automated emails whenever an attack is detected. This enhancement would significantly reduce the workload on analysts, as they would no longer need to manually collect packets and import them into the script. Instead, the script would operate continuously throughout the day, ensuring consistent monitoring of network traffic and proactively identifying any packets that may indicate an attack.

Python Script

Packages and Usage

We import the PyShark library to read in our PCAP file. We also use the sys library to take in arguments from the command line. This means the analyst will just have to write the name of the PCAP file they want to analyze when they run the script like so: `python3 test.py file.pcap`.

Script Entry Point:

The entry point of the script ensures that a valid PCAP file name is provided as input. The script passes the provided file to methods for both types of attacks we are targeting for this project (DDOS attacks and malware). Each method applies the relevant Wireshark filters, analyzes the results, and potentially notifies if an attack was detected. If no threats are detected, it outputs a message indicating no malicious patterns were found. To expand the program to run 24/7 with real, live network traffic rather than an existing PCAP file, we could simply capture packets for a predefined duration utilizing Pyshark's LiveCapture feature and pass packets captured for the duration to each attack detection method, and continuously looping through this process.

```
if __name__ == '__main__':
   if len(argv) != 2:
        print(f'usage: {argv[0]} <pcap file name>')
        exit()

    pcap = argv[1]

    if not os.path.exists(pcap):
        print("File not found")
        exit()

    # Send the pcap file to the functions
    detect_ddos(pcap)
    detect_mal_commands(pcap)
    if not detect_ddos and not detect_mal_commands:
        print("No malicious traffic patterns detected!")
        exit()
```

detect_ddos Function:

The detect_ddos method utilizes the DDOS attack filters designed in Wireshark. It uses the PyShark library to apply these filters to the provided PCAP file and analyze network packets based on specific criteria associated with common DDOS attack patterns. To automate the detection of each individual DDOS attack, the Wireshark filter was broken down into four filters to capture the different types of traffic: SYN flood attacks, characterized by TCP packets with the SYN flag set and the ACK flag unset; UDP flood attacks, identified by large UDP packets exceeding 1000 bytes; ICMP flood attacks, which involve ICMP echo requests or replies; and large frame attacks, signified by network frames exceeding 1500 bytes.

For each filter, the method applies it to process the PCAP file, counts the matching packets, and compares the count against predefined thresholds. If any threshold is exceeded, the method extracts the source IPs associated with the packets and their frequency, and prints an alert with this information. Here is where we could expand the program to both automatically send an email to the relevant parties so the terminal would not need to be constantly monitored and to extract and send any other information from the packets that may be relevant depending on the type of attack. Finally, After processing all filters, the method returns the attack flag, indicating whether a potential DDoS attack was detected.

Of course, under operation with real-time network traffic, the thresholds for detecting the attacks would vary depending on the specific network environment, normal traffic patterns, and the sensitivity of the monitoring system. If we were to expand our automation program, these thresholds could theoretically be set automatically by implementing a calibration method which could observe and learn the normal network behavior over a defined period and dynamically establish the thresholds based on some statistical analysis.

```
def detect_ddos(pcap):
    attack = False

    syn_flood_filter = 'tcp.flags.syn == 1 && tcp.flags.ack == 0'
    udp_flood_filter = 'udp && frame.len > 1000'
    imcp_flood_filter = 'icmp.type == 8 || icmp.type == 0'
    large_frame_filter = 'frame.len > 1500'

    def extract_ips(cap):
        ips = Counter()
        for packet in cap:
            ip = packet.ip.src
            if ip:
                ips[ip] += 1
        return ips

    syn_flood_cap = pyshark.FileCapture(pcap, display_filter=syn_flood_filter)
    syn_count = len([packet for packet in syn_flood_cap])
    if syn_count > SYN_FLOOD_THRESHOLD:
        print(f"Potential SYN Flood attack detected! (SYN count: {syn_count})")
        syn_ips = extract_ips(syn_flood_cap)
        print("Suspicious IPs involved:")
        for ip, count in syn_ips.most_common(5):
            print(f"  {ip} - {count} packets")
        attack = True

    syn_flood_cap.close()

    udp_flood_cap = pyshark.FileCapture(pcap, display_filter=udp_flood_filter)
    udp_large_count = len([packet for packet in udp_flood_cap])
    if udp_large_count > UDP_LARGE_PACKET_THRESHOLD:
        print(f"Potential UDP Flood attack detected! (Large UDP packets:
{udp_large_count})")
        udp_ips = extract_ips(udp_flood_cap)
        print("Suspicious IPs involved:")
        for ip, count in udp_ips.most_common(5):
            print(f"  {ip} - {count} packets")
        attack = True
    udp_flood_cap.close()

    imcp_flood_cap = pyshark.FileCapture(pcap, display_filter=imcp_flood_filter)
    icmp_count = len([packet for packet in imcp_flood_cap])
    if icmp_count > ICMP_THRESHOLD:
       print(f"Potential ICMP Flood attack detected! (ICMP packets:
{icmp_count})")
```

```
        icmp_ips = extract_ips(udp_flood_cap)
        print("Suspicious IPs involved:")
        for ip, count in icmp_ips.most_common(5):
            print(f"  {ip} - {count} packets")
        attack = True
    imcp_flood_cap.close()

    large_frame_cap = pyshark.FileCapture(pcap, display_filter=large_frame_filter)
    large_frame_count = len([packet for packet in large_frame_cap])
    if large_frame_count > 0:
        print(f"Potential Large Frame attack detected! (Large frames:
{large_frame_count})")
        large_frame_ips = extract_ips(large_frame_cap)
        print("Suspicious IPs involved:")
        for ip, count in large_frame_ips:
            print(f"  {ip} - {count} packets")
        attack = True
    large_frame_cap.close()

    return attack

        attack = True
    large_frame_cap.close()

    return attack
```

detect_mal_commands Function:

The detect_mal_commands function uses the Wireshark filter designed to detect network activity seen in commonly seen malware. As with the DDOS function, the PyShark library is used to analyze the provided packet capture file.

The filter is divided into three distinct sections. The TCP filter (tcp_filter) looks for TCP packets being directed to ports commonly closed to TCP. The filter for Cobalt Strike (cobalt_filter) filters out packets that have characteristics common with the penetration tool Cobalt Strike and the BEACON malware payload. The malicious DNS filter (malicious_dns_filter) filters out packets that have DNS queries from domains commonly used to distribute malware. If any of these filters contain one packet, that means the filter detected at least one malicious packet. If true, the script prints a message for the analyst and the function returns True.

```
def detect_mal_commands(pcap):
    # Fill in the function to detect other attack
    tcp_filter = '((tcp.port == 80 || tcp.port == 443 || tcp.port == 2222 ||
tcp.port == 4444 || tcp.port == 8080 || tcp.port == 8443 || tcp.port > 1024) &&
(frame.time_delta > 45 && frame.time_delta < 75))||(tcp.window_size == 1024) ||
tcp.analysis.retransmission || (frame.len > 600 && tcp.payload && tcp.stream)'
    cobalt_filter = '(ssl.handshake.extensions_server_name contains "cobalt" ||
ssl.handshake.type == 1 || ssl.record.content_type == 23) || (http.user_agent
contains "Cobalt" || http.user_agent contains "Mozilla" || http.request.uri
contains "beacon" || http.request.uri contains "index" || http.request.uri
contains "post" || http.request.uri contains "payload")'
    malicious_dns_filter = 'dns.qry.name contains ".onion" || dns.qry.name
contains ".xyz" || dns.qry.name contains ".top" || dns.qry.name contains ".ru" ||
dns.qry.name contains ".club"'

    # Checks for calls over commonly closed TCP ports, or large frame size
    tcp_cap = pyshark.FileCapture(pcap, display_filter=tcp_filter)
    tcp_frame_count = len([packet for packet in tcp_cap])
    if tcp_frame_count > 1:
        print("Potentially malicious activity over TCP")
        tcp_cap.close()
        return True

    tcp_cap.close()

    # checks for activity related to Cobalt Strike RAT and BEACON
    cobalt_cap = pyshark.FileCapture(pcap, display_filter=cobalt_filter)
    cobalt_frame_count = len([packet for packet in cobalt_cap])
    if cobalt_frame_count > 1:
        print("Potential malicious network activity related to Cobalt Strike")
      cobalt_cap.close()
       return True

    cobalt_cap.close()

    # Check if malicious dns queries
    dns_cap = pyshark.FileCapture(pcap, display_filter=malicious_dns_filter)
    dns_frame_count = len([packet for packet in dns_cap])
    if dns_frame_count > 1:
        print("DNS Queries from commonly malicious domain")
       dns_cap.close()
       return True

    dns_cap.close()

    return False
```

Conclusion

This project highlights the potential to enhance the capabilities of traditional packet sniffing and analysis tools such as Wireshark through automation and advanced filtering. By developing and leveraging customized and adaptable filters, packet sniffing software can be used to effectively locate malicious traffic patterns on a network, such as those associated with DDOS attacks and malware. This is enabled by the use of Wireshark's robust built-in filtering capabilities as well as Python scripts and libraries such as PyShark. These prototypes were developed with the intent of beginning the automation of the detection process. This approach not only streamlines the identification of potential threats, but also would increase the efficiency and accuracy of security analysts allowing them to focus on mitigating attacks instead of sifting through packet capture files.

Our findings demonstrate the viability of combining real-time packet analysis software with custom made automated detection to create a more robust network security solution. The scalability of this solution would be accomplished through continuous development to create more filters and Python scripts for any type of attack in the quickly changing cybersecurity landscape. The foundation laid by this project illustrates how automation of packet monitoring can be a valuable investment for cybersecurity personnel. With additional time and resources, this solution could become a powerful tool for cybersecurity professionals to detect and respond to network threats efficiently.

Group Members' Contributions

The group member's split up their responsibilities and contributions into 2 teams, analyst team and engineering team. The analyst team was responsible for implementing the filters in WireShark using the publicly available packet capture files, documenting and showing the results of the filters, and providing an overview of multiple attacks. The team members on the analyst team were Baker Womack, Nazire Brown, Jacob Sherer, Andy Smith, Cameron Washington, and Bradley Zenker. The engineering team was responsible for developing the sudo-code of the python scripts to take the filters and run them automatically to notify the user and writing the documentation of the Python Scripts. The team members on the engineering team were Rhaynie Bongiorno, Ryan Malone, Christine Steege, and Shawn Tan. Any other responsibilities were evenly distributed and split between the group members.

References

Banerjee, U., Vashishtha, A., & Saxena, M. (2010). Evaluation of the Capabilities of Wireshark as a Tool for Intrusion Detection. *International Journal of Computer Applications*, 6(7), 1-6. https://doi.org/10.5120/1092-1427

Baptist, W. (2023, May 12). *Simple Wireshark Scripts for Easy Network Forensics*. Medium. Retrieved November 17, 2024, from https://medium.com/@williambaptist/simple-wireshark-scripts-for-easy-network-forensics-ff17a36ea278

Beridze, B. O., & Donadze, M. V. (2024). Analysis of the security level of information systems and methods for detecting network anomalies. *Proceedings of the International Conference on Computer Systems and Technologies 2024*, 2, 10–15. https://doi.org/10.1145/3674912.3674924

Chandravathi, D., Praneeth, P. V. S. S., Sakina, A., Anjanaa, K., & Priyanka, B. (2024). Network Intrusion Detection Using Wireshark and Machine Learning. *International Journal of Telecommunications and Emerging Technologies*, 10(1), 23–31. https://journalspub.com/wp-content/uploads/2024/09/23-31-Network-Intrusion-Detection-Using-Wireshark-And-Machine-Learning.pdf

Diyeb, I. A. I., Saif, A., & Al-Shaibany, N. A. (2018). Ethical Network Surveillance Using Packet Sniffing Tools: A comparative study. *International Journal of Computer Network and Information Security*, 10(7), 12-22. https://doi.org/10.5815/ijcnis.2018.07.02

Green, D. (2023, April 26). *pyshark/readme.md*. github.com. Retrieved November 17, 2024, from https://github.com/KimiNewt/pyshark/blob/master/README.md

Hussain, A., Qadri, S., Razzaq, A., Nazir, H., & Ullah, M. S. (2024). Enhancing LAN Security by Mitigating Credential Threats via HTTP Packet Analysis with Wireshark. *Journal of Computing & Biomedical Informatics*, 6(2), 1-10. https://doi.org/10.56979/602/2024

Iqbal, H., & Naaz, S. (2019). Wireshark as a Tool For Detection of Various LAN Attacks. *International Journal of Computer Sciences and Engineering*, 7(5), 833-837. https://doi.org/10.26438/ijcse/v7i5.833837

Kulshrestha, A., & Dubey, S. K. (2014). A Literature Review on Sniffing Attacks in Computer Network. *International Journal of Advanced Engineering Research and Science*, 1(2), 67–73. https://ijaers.com/Paper-July%202014/IJAERS-JULY-2014-010.pdf

Mabsali, N. A. L., Jassim, H., & Mani, J. (2023). Effectiveness Of Wireshark Tool For Detecting

Attacks And Vulnerabilities In Network Traffic. In N. Bacanin & H. Shaker (Eds.), *Proceedings of the 2022 International Conference on Innovation and Information Technology in Business (ICIITB)* (pp. 114–135). Atlantis Press. https://doi.org/10.2991/978-94-6463-110-4_10

Ndatinya, V., Xiao, Z., Manepalli, V. R., Meng, K., & Xiao, Y. (2015). Network forensics analysis using Wireshark. *International Journal of Security and Networks*, 10(2), 91. https://doi.org/10.1504/ijsn.2015.070421

Shaw, R., & Parveen, S. (2024). Literature Review on Packet Sniffing: Essential for Cybersecurity & Network Security. *2024 5th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, https://doi.org/10.1109/ICICV62344.2024.00119

SYN Flood DDoS attack | Knowledge Base | MazeBolt. (n.d.). Knowledge Base. Retrieved November 19, 2024, from https://kb.mazebolt.com/knowledgebase/syn-flood/

West, M. (2019, May 11). *Capturing Packets Continuously Everyday Using PyShark and Cron*. Medium. Retrieved November 17, 2024, from https://medium.com/@mwester1111/capturing-packets-continuously-everyday-using-pyshark-and-cron-a7835bf1beb0

2021-05-13 (Thursday) - Hancitor with Ficker Stealer and cobalt strike. (2021, 5 13). malware-traffic-analysis.net. https://malware-traffic-analysis.net/2021/05/13/index.html