Intrusion Detection, Firewall, and Network Analysis

- **SEC290 Final Project**
- Hector Acosta
- **Course**: SEC290
- **Professor**: Larry Burnette

Introduction to the Project

Objective:

This project covers the following key areas of network security and defense:

- **Intrusion Analysis** using tools like **Wireshark** and **Snort**.
- **Firewall Configuration** with **iptables** to secure network traffic.
- **Live Memory Analysis** using tools like **Volatility**, **Process Hacker**, and **Process Monitor**.
- Applying techniques such as:
    - Time-based access control
    - Blackhole routing
    - Bogon filtering
    - Stateful firewalls
    - Dynamic NAT
    - Brute force protection for SSH

Intrusion Analysis with Wireshark

Objective:

Conduct protocol analysis and detect network attacks using **Wireshark**.

- **Key Activities**:
- **Basic Protocol Analysis**:
  - Capturing and analyzing ICMP packets.
  - Identifying **Type** and **Hexadecimal values** in ICMP packets.
- **Using Display Filters**:
  - Filtering TCP and HTTP traffic.
  - Analyzing HTTP streams for clear-text vulnerabilities.
- **Common Attacks**:
  - Performing **Nmap Scans** (Ping Scan, SYN Scan).
  - Identifying artifacts of recon activities in captured traffic.
- **Basic Attack Analysis**:
  - Analyzing a **pcap file** to detect suspicious activities (e.g., TTL differences, flag settings).

Intrusion Analysis with Wireshark

Basic attack analysis

1. Look at captures no. 20 and 22. (You can use the “Go” link at the top of the Wireshark screen to quickly go to a specific capture) Both packets are ICMP traffic but there are subtle differences between them. Compare the time-to-live and data field sizes in the two packets. What differences do you see? Packets 20 and 22 are both ICMP echo requests, but they differ in Time-To-Live (TTL) and Data field size. Packet 20 has a TTL of 128 and a data size of 74 bytes, while Packet 22 has a TTL of 64 and a data size of 98 bytes. The difference in TTL suggests they may come from different operating systems (TTL 128 is typical for Windows, and TTL 64 for Linux/Unix). The larger data size in packet 22 might indicate added information or padding. These variations could point to differences in system configurations or packet handling along the network path.

2. Do a little Internet research to discover which operating systems use the specific values in their ping commands. What operating system generated the echo request in capture 20? In packet capture 20, the ping request has a Time-To-Live (TTL) of 128, which is a default value used by Windows operating systems like Windows 7, 10, or 11. This suggests that the echo request in capture 20 likely came from a Windows computer.

3. Review packet no. 37 and beyond, what do you think is taking place here? In packet 37 and the following packets, it looks like 192.168.25.200 is performing a port scan on 192.168.25.1. The source sends SYN packets (connection requests) to different ports on the destination, trying to see if any ports are open. Each time, 192.168.25.1 responds with a RST (Reset), which rejects the connection. This pattern sending SYNs and receiving RSTs indicates a TCP SYN scan, a common method to check which ports on a computer are open or closed. This type of scan is often used to gather information about a network.

4. Look at capture 22846. What is suspicious about the flag settings in this packet? In capture 22846, the packet has only the FIN flag set, with no other flags like ACK or SYN. This is unusual because, normally, a FIN flag is used with an ACK to politely close a connection. A packet with just the FIN flag, without any acknowledgment, can be suspicious. This type of packet is sometimes used in FIN scans, where someone tests if a port is open in a way that might avoid detection. Attackers sometimes use FIN scans to gather information without setting off alarms.

5. What is the IP address of the host being targeted? The IP address of the host being targeted in this capture is 192.168.25.1.

OpenSSL Analysis

Objective:

- Explore SSL/TLS analysis by creating and testing encryption keys and certificates with OpenSSL.
- **Key Activities**:
- **Creating SSL/TLS Keys and Certificates**:
    - Generated a self-signed certificate and RSA key pair.
    - Created two files: `server.crt` (certificate) and `server.pem` (private key).
- **Running an SSL Server**:
    - Set up a local SSL server using the self-signed certificate and key.
    - Verified the connection using Firefox by accessing https://localhost:4433 and accepting the self-signed certificate warning.
- **Wireshark Capture and Decryption**:
    - Captured SSL traffic on the loopback interface.
    - Added the private key in Wireshark to decrypt the SSL traffic.
    - Viewed the decrypted GET request in Wireshark.

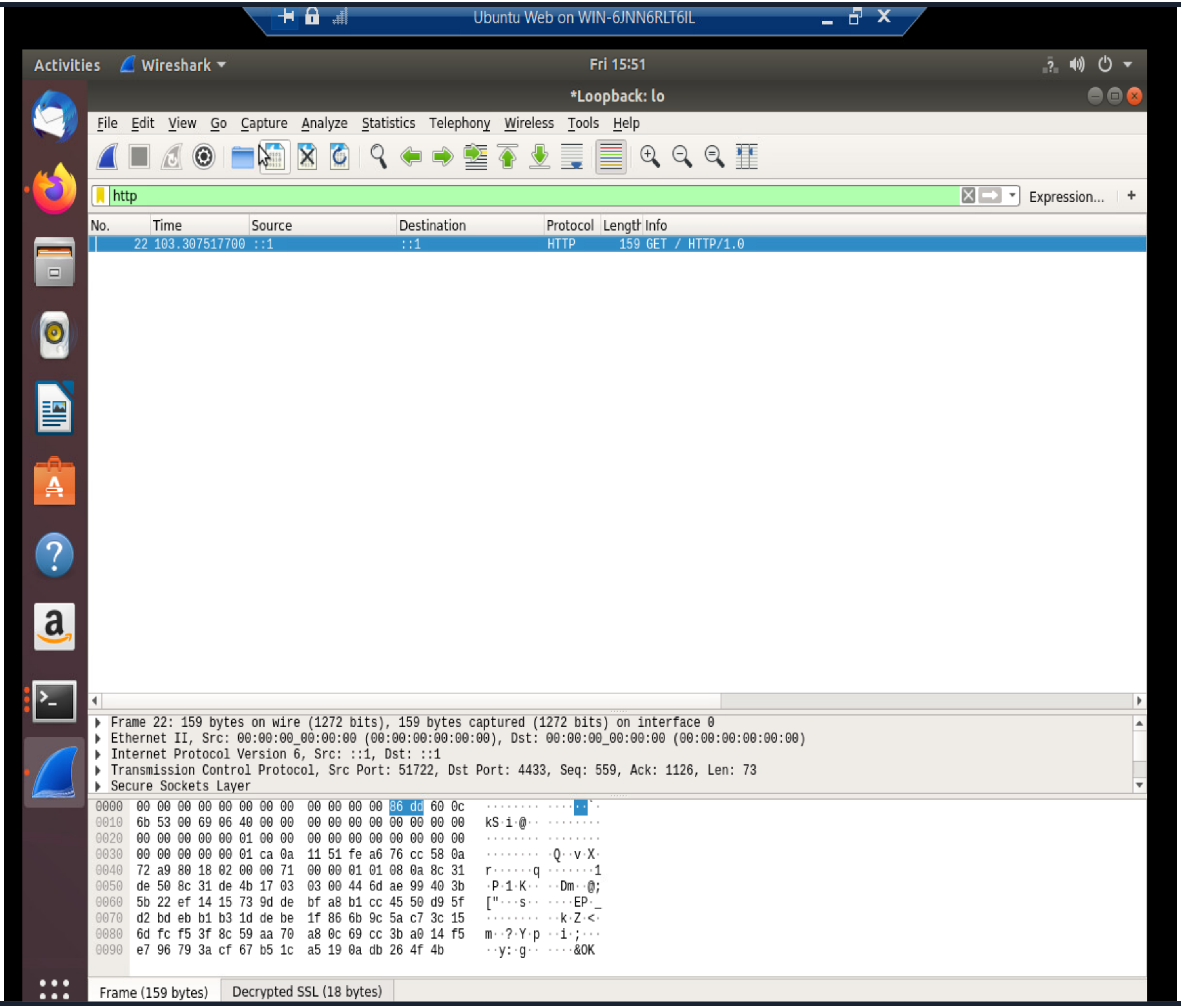
Creating and testing an SSL/TLS file

This slide shows the process of creating and testing an **SSL/TLS file** using **Wireshark** for network analysis. The image displays the captured network packets during an SSL/TLS connection.

- **Tool**: Wireshark
- **Details**: Network traffic captured on **port 443** (HTTPS) using **SSL/TLS encryption**.
- **Packet Content**: Shows encrypted data exchanged between the client and server.

Why It Matters:

This helps in analyzing secure connections, identifying vulnerabilities, and ensuring encryption is properly implemented.



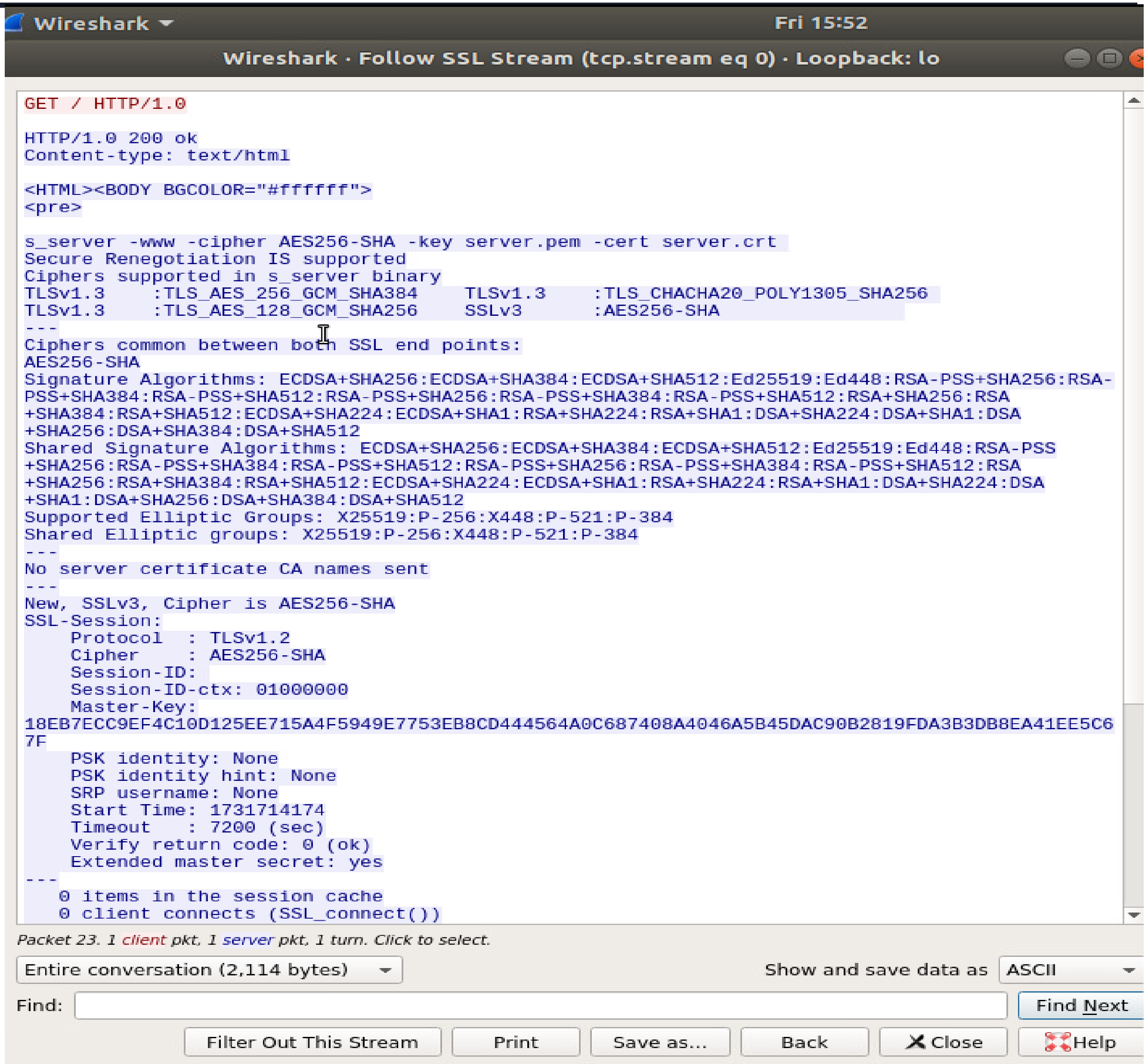
Creating and testing an SSL/TLS file cont'd

This slide shows the detailed analysis of an **SSL/TLS stream** in Wireshark. The image displays the **HTTP request and response** within the decrypted SSL/TLS stream.

- **Tool**: Wireshark
- **Details**:
  - HTTP GET request captured through SSL/TLS.
  - Decrypted content shows the actual text exchanged.
  - SSL encryption details (cipher suite and certificate information).

  Why It Matters:

  Analyzing SSL/TLS traffic helps verify that encryption is working correctly and detect potential vulnerabilities in secure communications.



Intrusion Detection with Snort

Objective:

- Explore network intrusion detection using **Snort** and analyze alerts generated by different types of network scans.

- **Key Activities**:

- **Testing Snort Rules**:
    - Deployed Snort on the **Security Onion IDS** machine.
    - Generated network traffic using **Nmap scans**:
        - **XMAS Scan** to trigger Snort alerts.
    - Viewed alerts in **Sguil Dashboard** and analyzed the details of flagged packets.

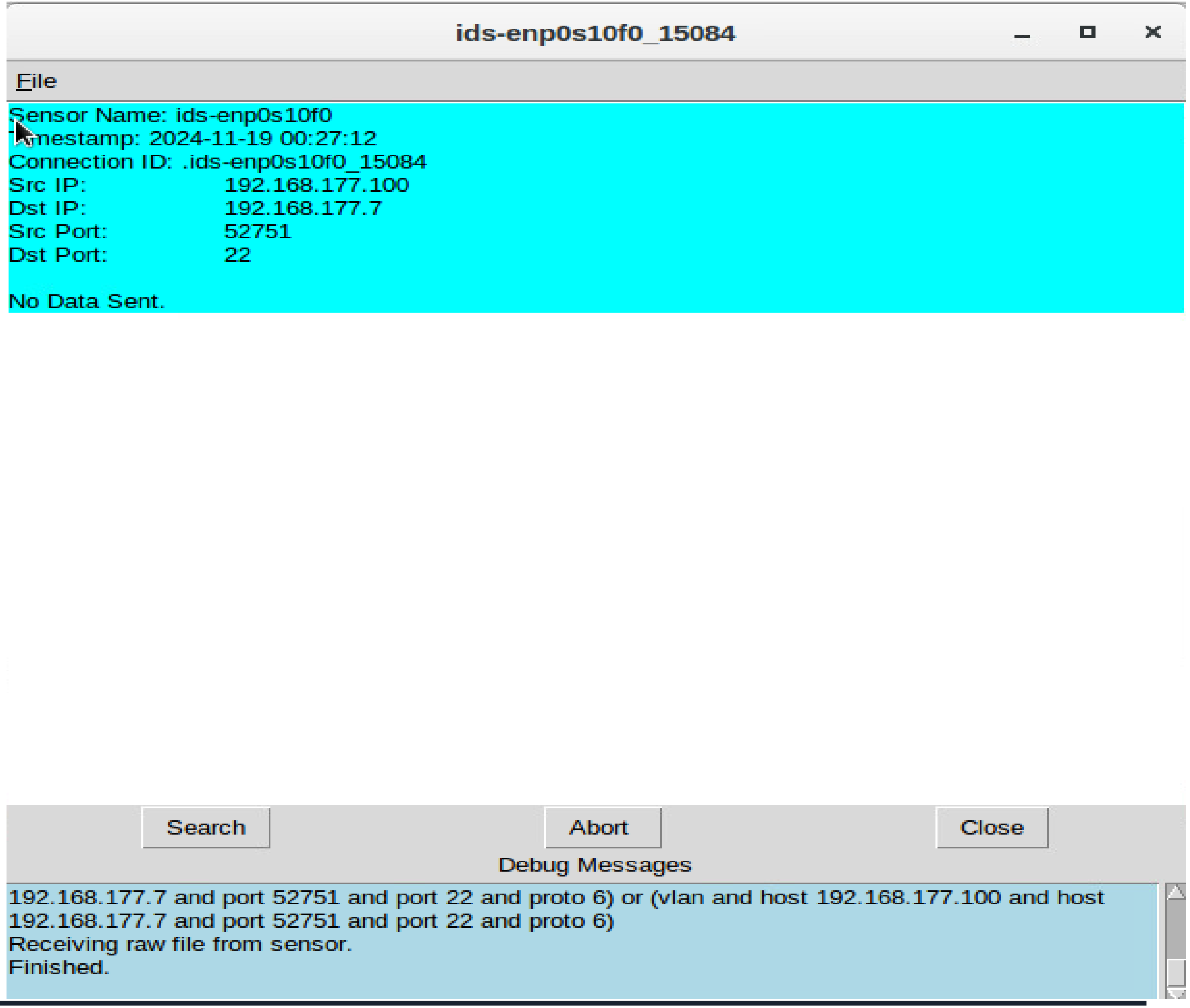
- **Creating Custom Snort Rules**:
    - Added a rule to detect **ICMP traffic** to the **local.rules** file.
    - Verified the rule by generating **ping traffic** to the **OWASP-BWA** machine.
    - Observed the alerts in the Sguil dashboard.

Testing Snort rules

This slide demonstrates the process of testing **Snort rules** for intrusion detection. The image shows a detected alert with detailed information.

- **Tool**: Snort
- **Details**:
  - **Source IP**: 192.168.177.100
  - **Destination IP**: 192.168.177.7
  - **Ports**: Source port 52751, Destination port 22 (SSH)
  - **Alert**: Snort detected suspicious activity (XMAS scan).

Why It Matters:
Testing Snort rules helps identify malicious activities, such as port scans, and ensures that the intrusion detection system (IDS) is configured properly.



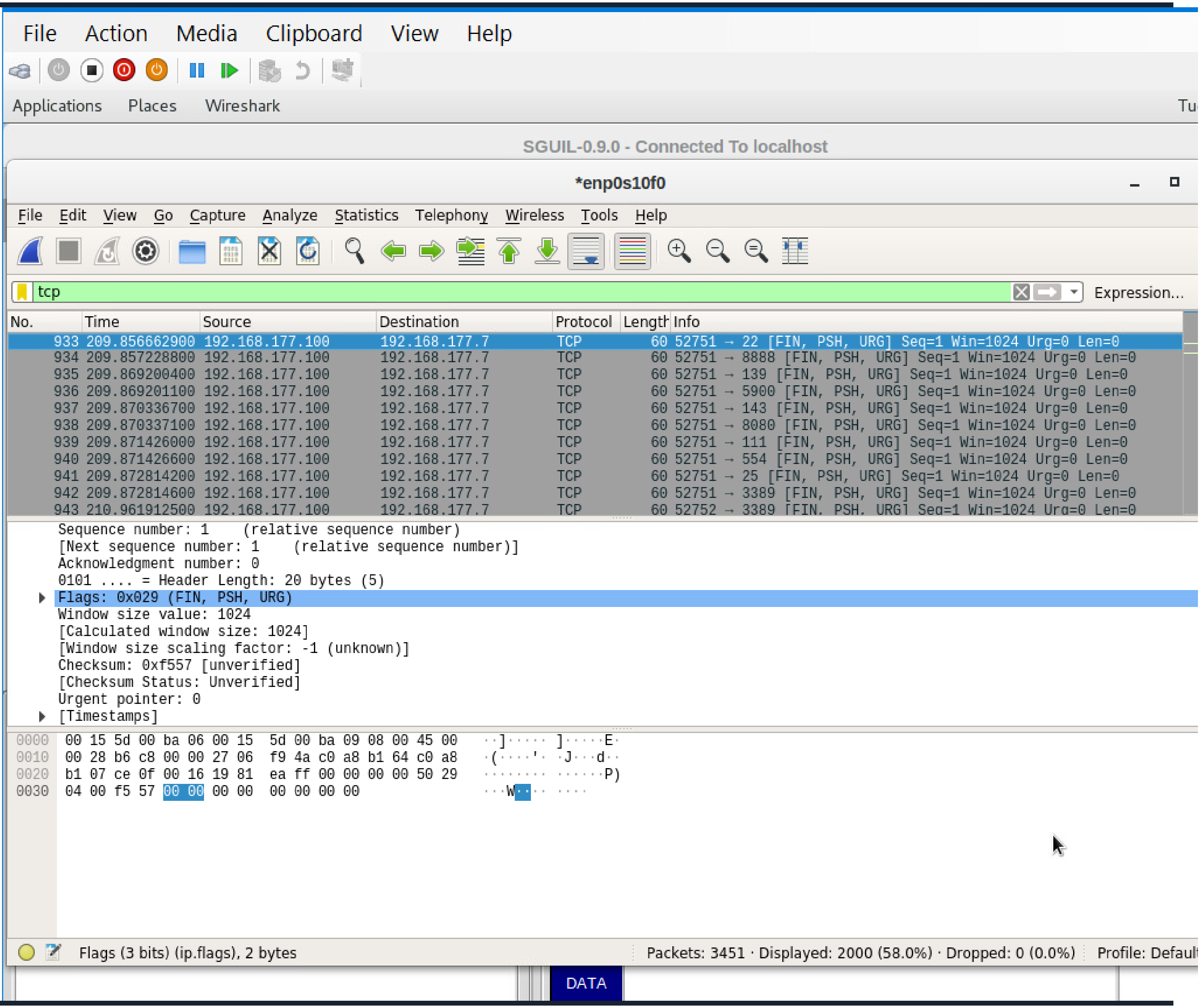
Testing Snort rules cont’d

This slide shows the detailed analysis of network traffic using **Wireshark** after Snort has detected an alert. The image displays the captured **TCP packets** with flags set, typical of an XMAS scan.

- **Tool**: Wireshark
- **Details**:
    - Captured **TCP traffic** with flags: **URG, PSH, FIN**.
    - **Source**: 192.168.177.100
    - **Destination**: 192.168.177.7
    - Snort rule triggered by the unusual combination of flags.

Why It Matters:

This analysis confirms Snort's detection capabilities and helps understand the nature of the suspicious traffic, aiding in securing the network from potential attacks.



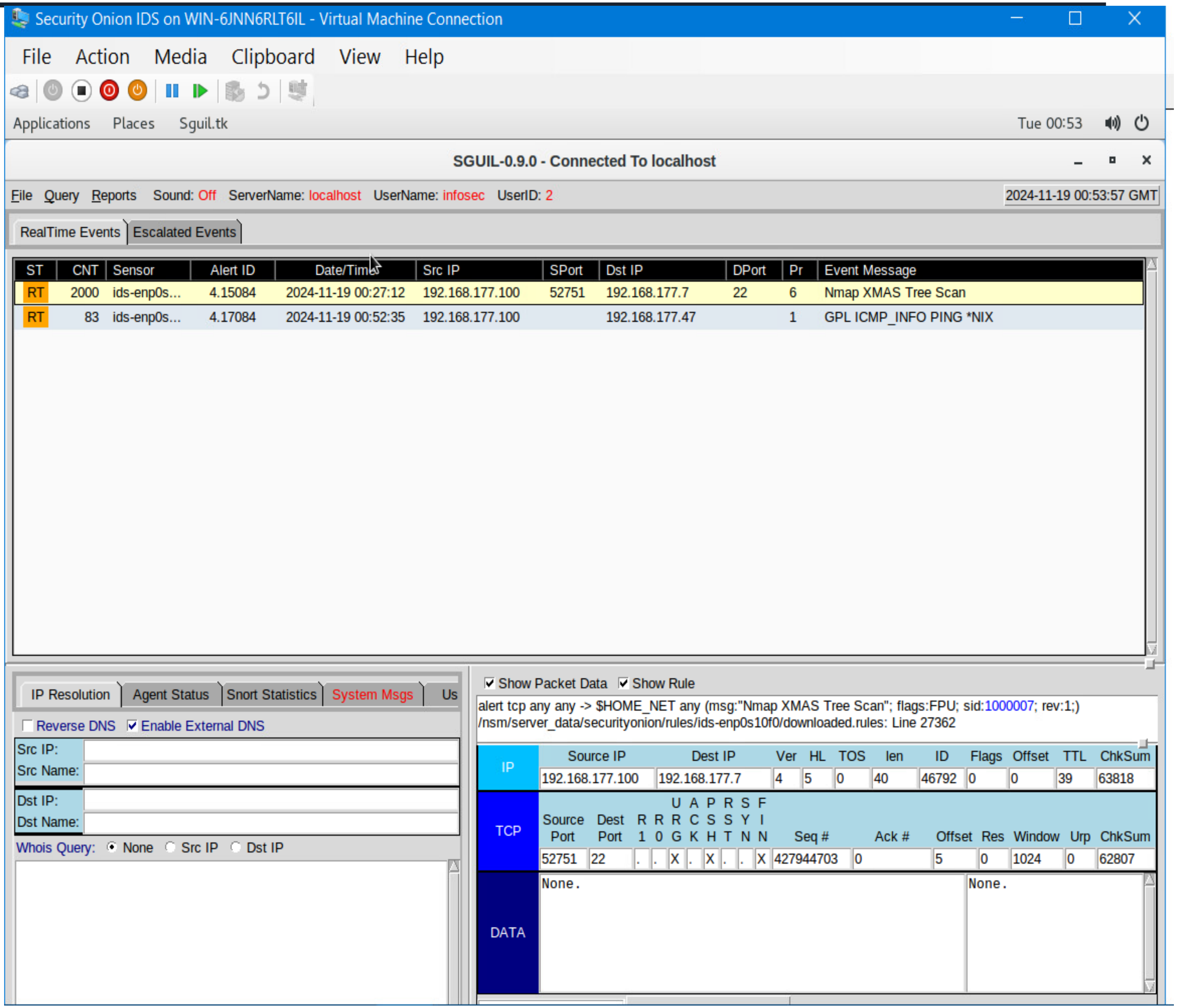
Creating Snort rules

This slide illustrates the process of creating custom **Snort rules** to detect specific network activities. The image shows the **Sguil dashboard** displaying alerts triggered by Snort.

- **Tool**: Snort and Sguil
- **Details**:
  - **Alert**: ICMP traffic detected (ping scan).
  - **Source IP**: 192.168.177.100
  - **Destination IP**: 192.168.177.7
  - Rule configured to identify suspicious network behavior.

  Why It Matters:

  Creating custom Snort rules allows for tailored detection of specific threats, enhancing the effectiveness of intrusion detection systems.



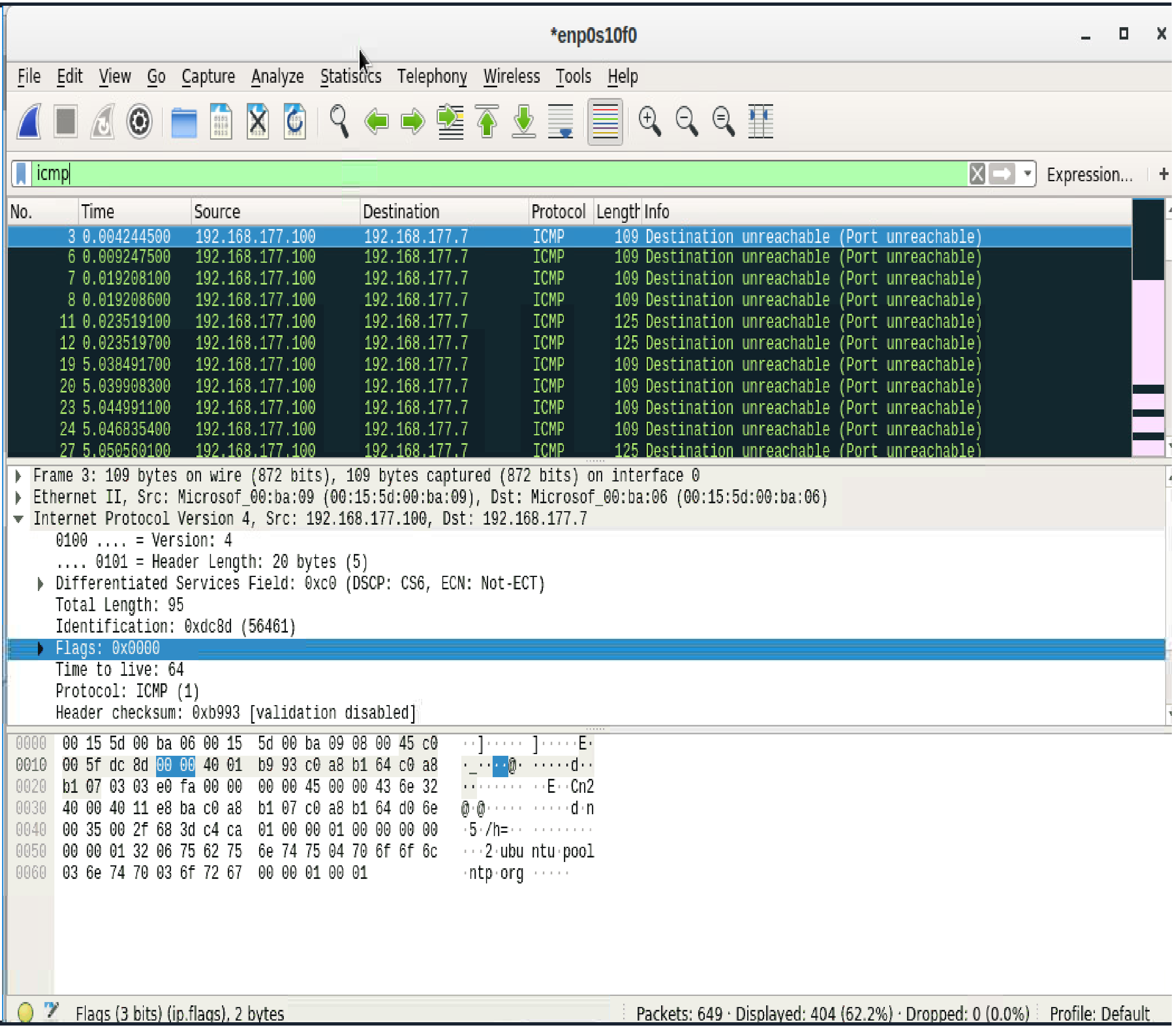
Creating Snort rules cont’d

This slide shows the detailed packet analysis in **Wireshark** after applying custom Snort rules. The image highlights **ICMP traffic** that has been captured and flagged as an alert.

- **Tool**: Wireshark
- **Details**:
  - **Source IP**: 192.168.177.100
  - **Destination IP**: 192.168.177.7
  - **Protocol**: ICMP
  - Detailed breakdown of the packet, including headers and flags.

Why It Matters:

This analysis helps validate that Snort rules are functioning correctly and provides deeper insight into the network traffic being monitored for potential threats.



Live Memory Analysis

Objective:

•Perform live memory analysis on Linux and Windows systems using tools like **Volatility**, **Process Hacker**, and **Process Monitor**.

•**Key Activities**:

•**Linux Processes Analysis**:

- •Identified running processes with commands like ps and `lsof`.
- •Examined **open network connections** and **kernel modules**.

•**Process Hacker**:

- •Used **Process Hacker** on the **Malware VM** to analyze running processes.
- •Identified **parent-child relationships** to detect potential malicious activity.

•**Process Monitor**:

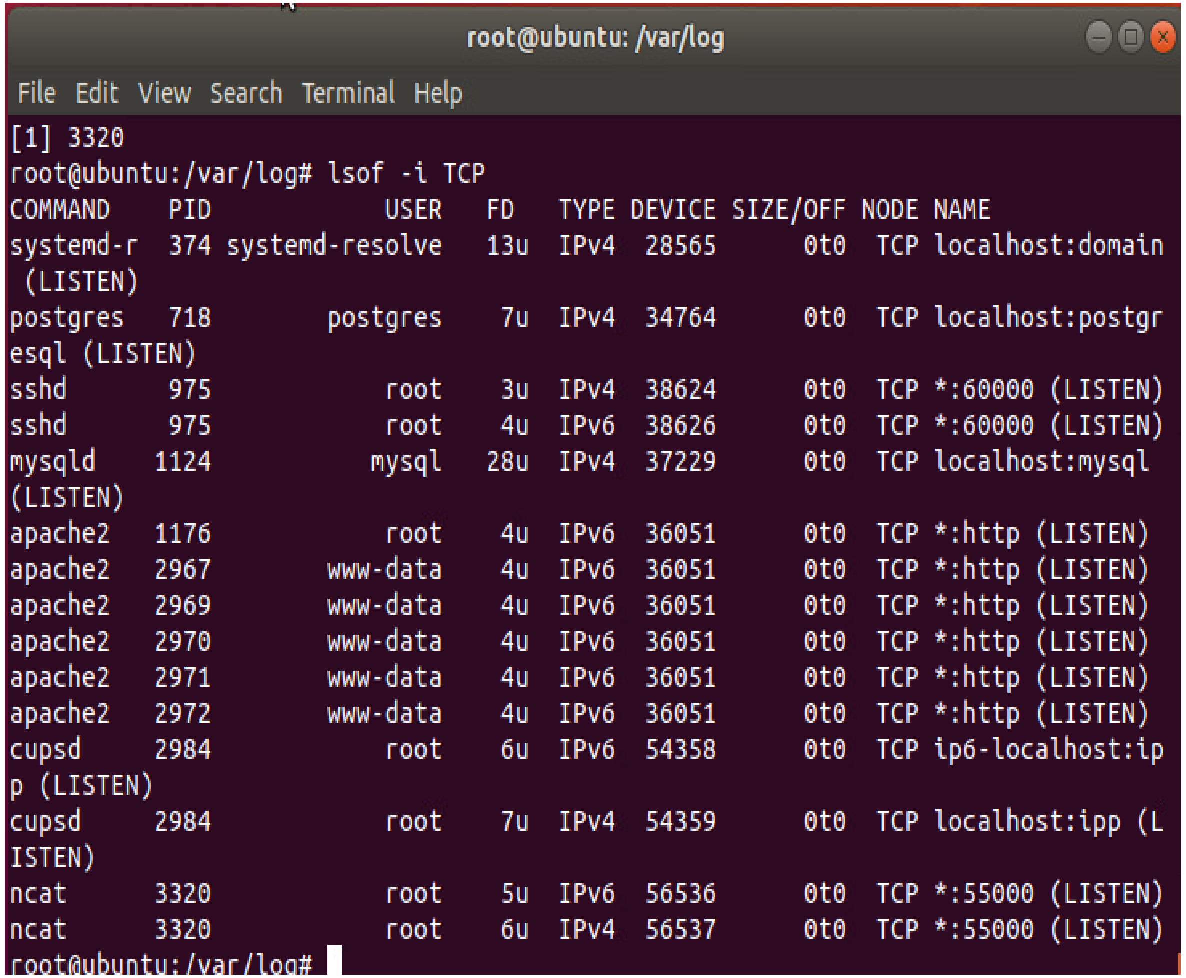
- •Captured and filtered registry changes made by **Notepad**.
- •Identified where font and font size settings are stored in the Windows Registry.

Linux Processes

This slide shows how to analyze **Linux processes** using the lsof command. The image displays processes actively using **TCP connections**. Key information includes:

- **COMMAND**: Process name (e.g., sshd, apache2).
- **PID**: Process ID.
- **USER**: User running the process.
- **FD**: File descriptor.
- **TYPE**: Connection type (IPv4/IPv6).
- **NODE NAME**: Port or service (e.g., http, tcp, 55000).

Why It Matters:
This helps identify active services, detect unauthorized connections, and secure the system by closing unnecessary ports.



Process Hacker

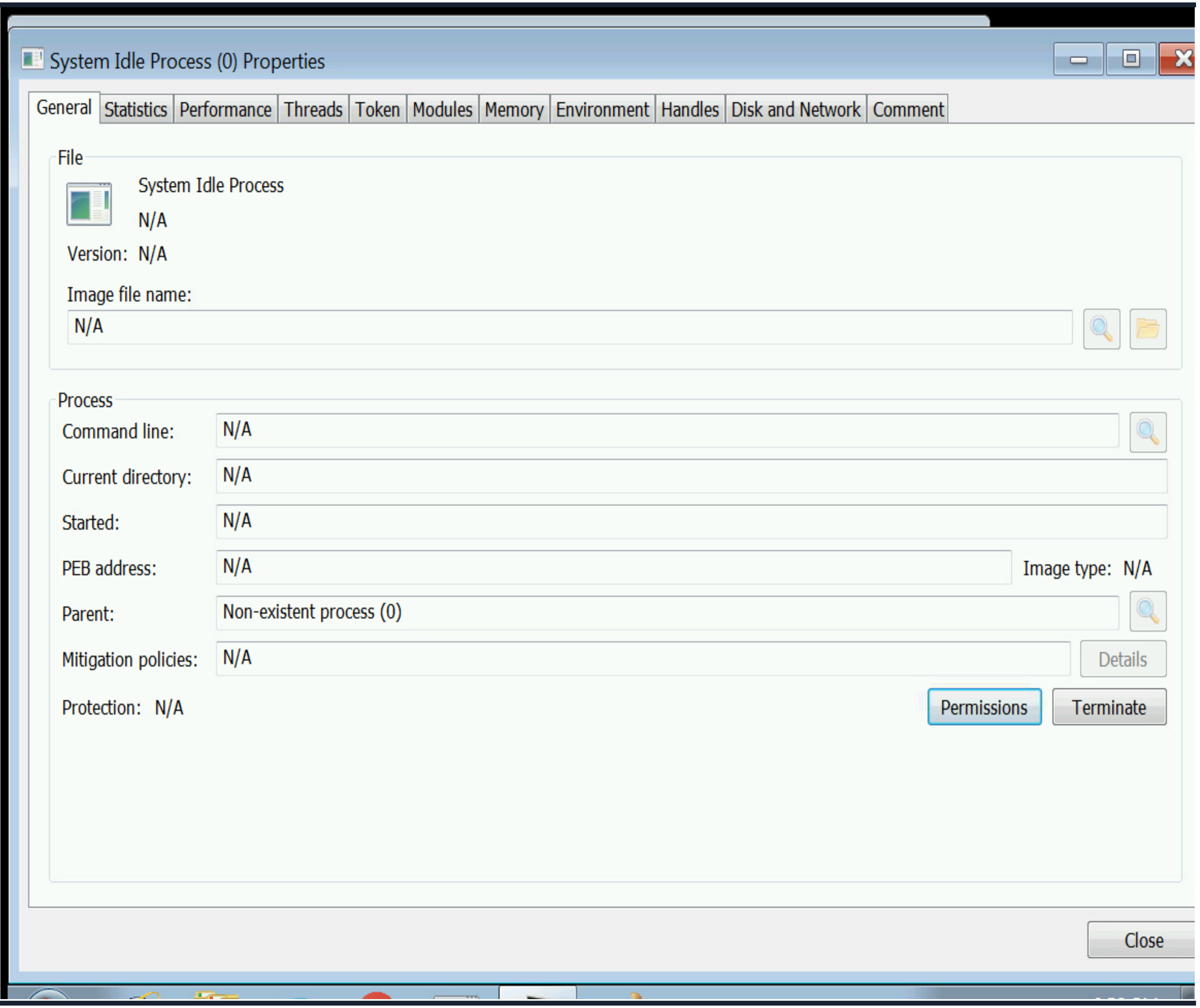
This slide shows the **Process Hacker** tool, a powerful utility for analyzing system processes in detail. The window displays properties of the **System Idle Process**.

•**Tool**: Process Hacker

- **Details**:
- **Process**: System Idle Process
- **Information**: Command line, current directory, parent process, protection settings
- **Tabs**: Statistics, Performance, Threads, Memory, Handles

Why It Matters:

Process Hacker is essential for **monitoring and managing processes** on a system. It helps identify **suspicious activity, malware**, or anomalies by examining process details and their relationships.



Process Monitor

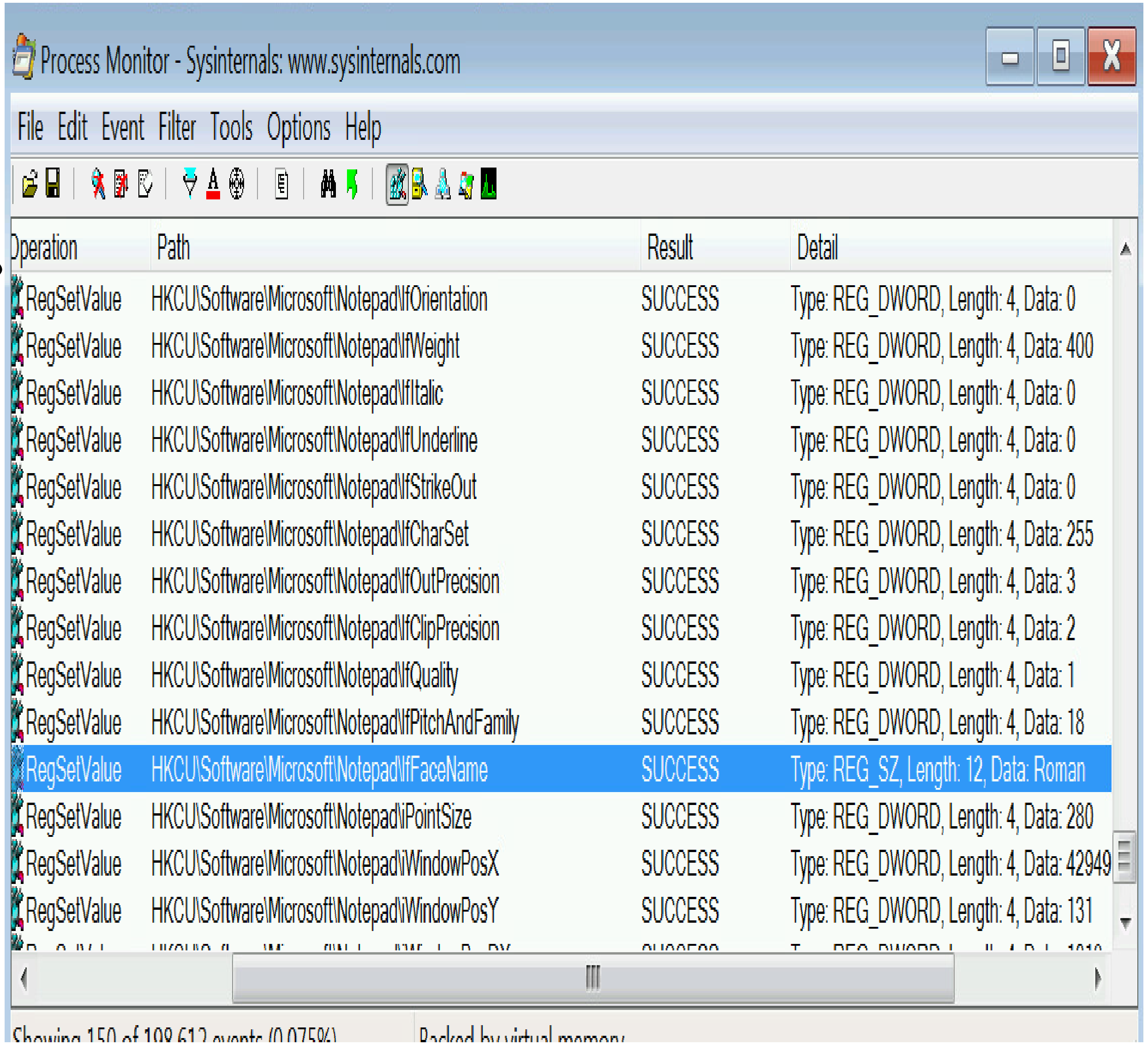
This slide shows **Process Monitor**, a real-time monitoring tool used for tracking system activity, particularly **registry, file system, and process/thread operations**.

•**Tool**: Process Monitor

- **Details**:
- **Operation**: Registry modifications (RegSetValue)
- **Path**: Locations in the Windows Registry related to Notepad settings
- **Result**: SUCCESS indicates the operation was completed successfully
- **Detail**: Shows the data written or modified in the registry

Why It Matters:

Process Monitor helps analyze **system changes, diagnose issues**, and identify malicious behavior by tracking **registry edits and file system activities** in real-time. It is valuable for **troubleshooting** and **forensic investigations**.



Firewall Configuration with iptables

Objective:

•Configure and analyze firewall rules using **iptables** to secure network traffic.

•**Key Activities**:

•**Time-Based Access**:

- •Configured firewall rules to allow **SSH access** to the **DMZ machine** only during specific hours (8 AM to 4 PM CST, Monday to Friday).
- •Verified access control by adjusting the system clock.

•**Blackhole Routing**:

- •Created a blackhole route to drop packets to a specific IP address to block unwanted traffic.

•**Bogon Filtering**:

- •Blocked packets from **invalid IP address ranges** (bogons) to prevent malicious activity.

•**Stateful Firewall**:

- •Implemented a stateful firewall to allow only **established and new connections**.
- •Dropped **invalid packets** to enhance security.

Time-based Access

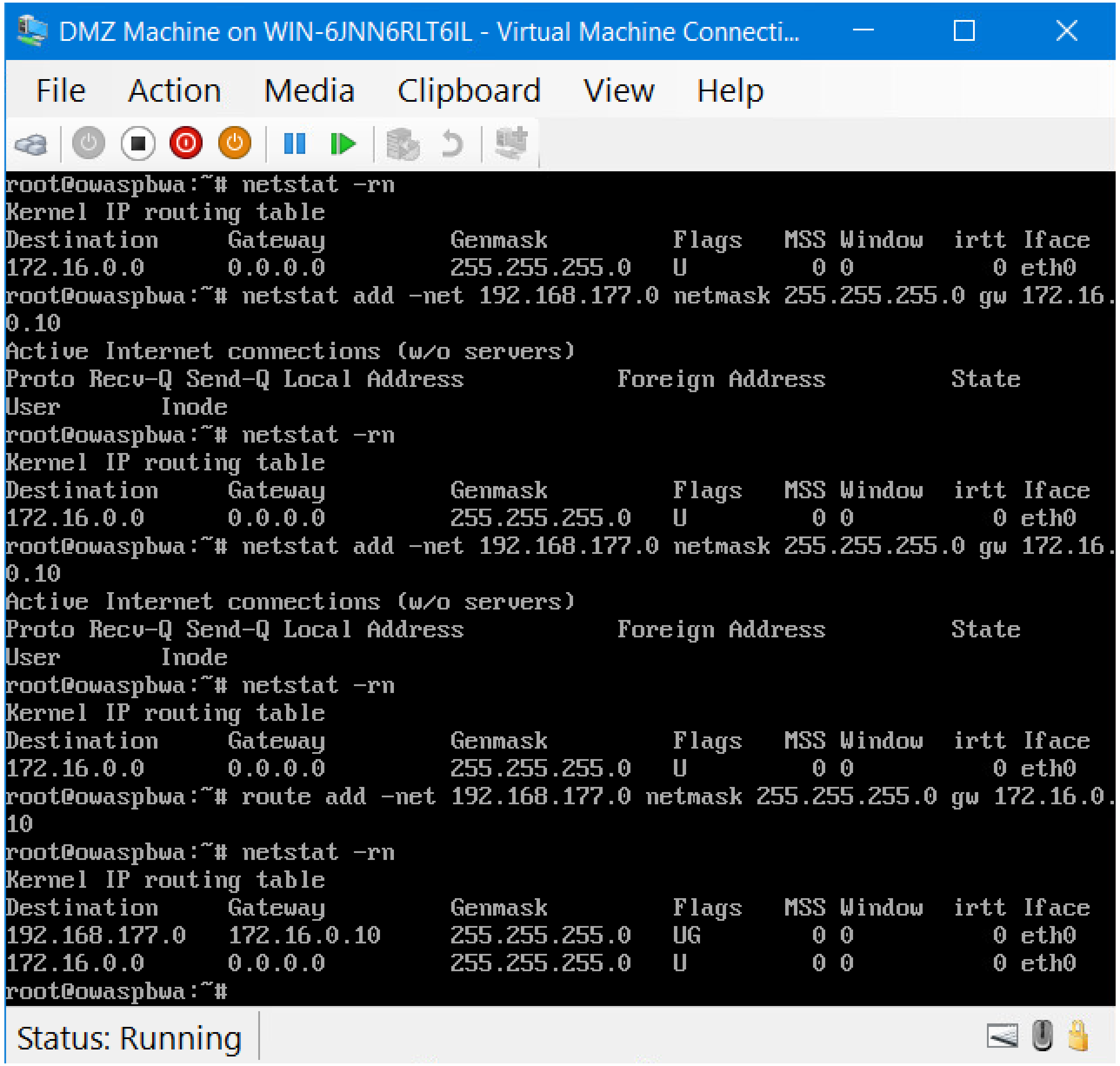
This slide demonstrates configuring **time-based access rules** for controlling network traffic based on specific time windows.

•**Tool**: iptables on the Firewall Machine

- **Details**:
- Shows the **routing table** and **network configuration** on the **DMZ Machine**.
- **Time-based access** allows SSH connections to the DMZ Machine only during specific times (e.g., 8:00 AM to 4:00 PM CST).
- Ensures traffic is forwarded according to defined **time constraints** for security.

Why It Matters:

Time-based access limits exposure to **potential attacks** by only allowing connections during **authorized time windows**, reducing the risk of unauthorized access during off-hours.



Time-based Access

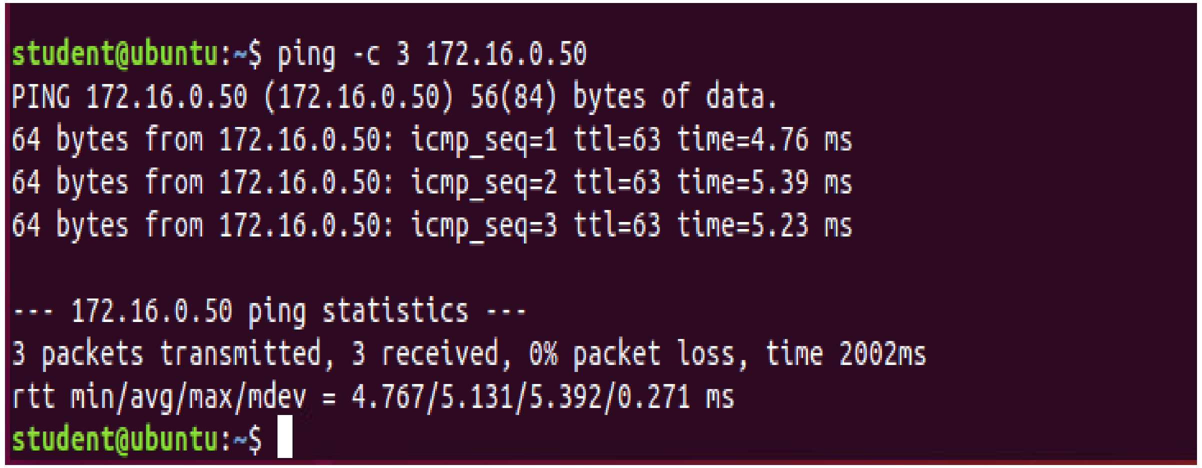
This slide shows a successful **ping test** to verify time-based access rules.

•**Tool**: ping command on the Ubuntu Web Machine

- **Details**:
- The **ping** command successfully sends and receives packets from **172.16.0.50**.
- Indicates that the time-based access rules allow network traffic during the permitted window.
- **3 packets** were transmitted and received with **0% packet loss**, confirming connectivity.

Why It Matters:

Verifying connectivity helps ensure that the **time-based access rules** are functioning correctly. It demonstrates that access is permitted within the specified time frame, enhancing **network security** and **controlled access**.



Time-based Access

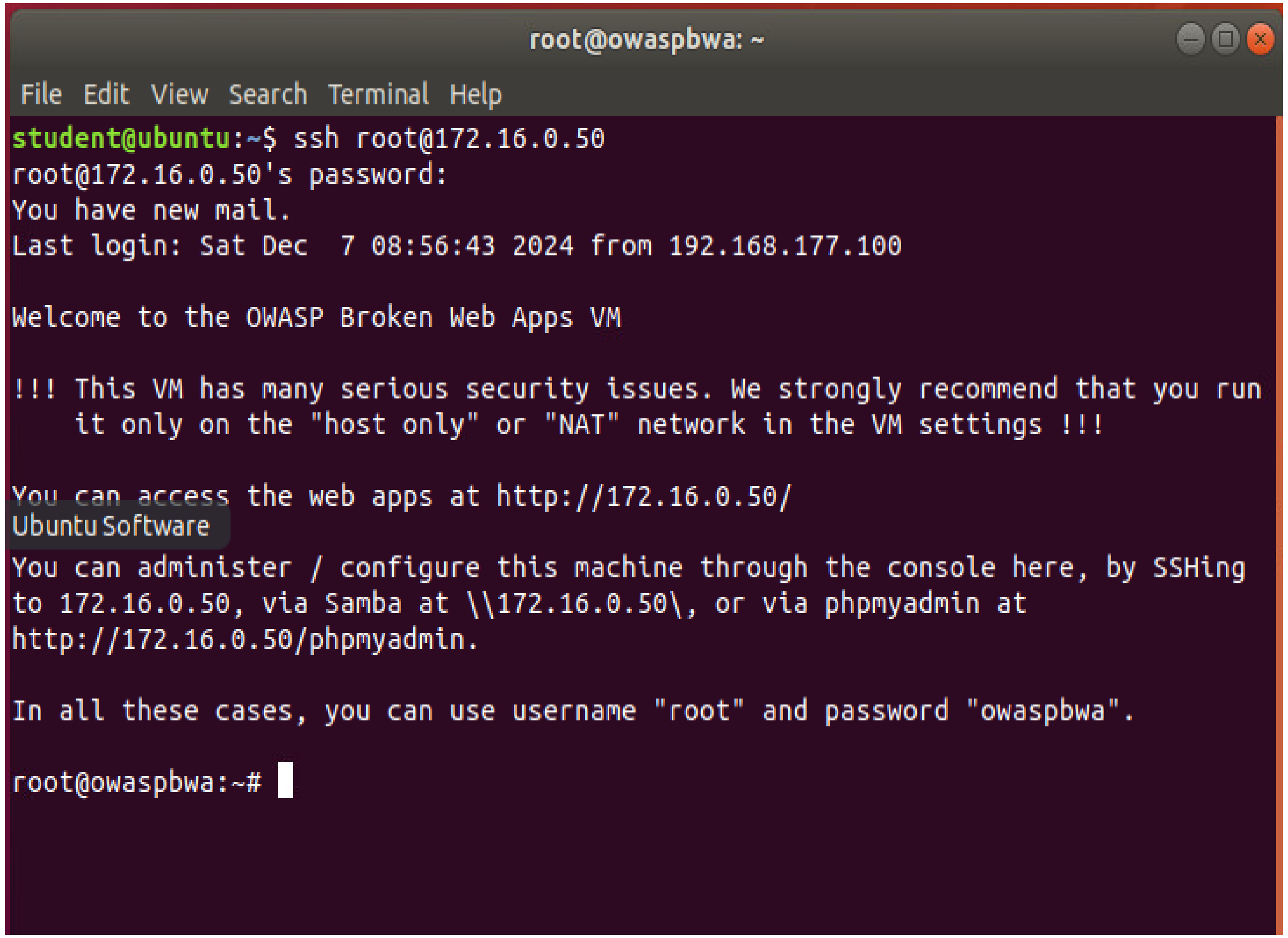
This slide demonstrates successful **SSH access** during the allowed time window.

•**Tool**: SSH connection from the **Ubuntu Web Machine** to the **DMZ Machine** at **172.16.0.50**.

- **Details**:
- The SSH session connects successfully, prompting for a password.
- The login is authenticated, and the terminal displays the **OWASP Broken Web Apps VM** console.
- Shows administrative access to manage or configure the DMZ machine.

Why It Matters:

This confirms that the **time-based access rule** allows SSH connections only during the specified time frame. It helps ensure **secure remote access** to critical machines while limiting exposure to unauthorized users outside the allowed hours.



Conclusion and Key Takeaways

Throughout this course, I gained valuable hands-on experience with essential cybersecurity tools and techniques. In **Module 2**, I learned to capture and analyze network traffic using **Wireshark**, which allowed me to identify different types of attacks. **Module 3** focused on **SSL/TLS encryption** using **OpenSSL**, where I created and tested encryption keys and certificates and decrypted traffic in Wireshark. In **Module 4**, I explored **Snort** for intrusion detection, analyzed alerts generated by different scans, and created custom rules to detect specific traffic. **Module 5** provided insights into **live memory analysis** using tools like **Volatility**, **Process Hacker**, and **Process Monitor** to identify running processes, memory dumps, and registry changes. Finally, in **Module 6**, I configured firewall rules with **iptables** to implement time-based access, blackhole routing, bogon filtering, and stateful firewall configurations. These modules have equipped me with practical skills for network defense, intrusion detection, and system analysis, laying a solid foundation for real-world cybersecurity roles and further certification.

Challenges Faced

Completing this project presented several challenges that tested my problem-solving and technical skills. One major challenge was navigating the different virtual machines and ensuring they were properly configured for each module. Switching between environments like the Security Onion IDS, Ubuntu Web, and Malware VM required careful attention to detail and time management. Additionally, understanding and executing complex commands in tools like **Wireshark**, **OpenSSL**, **Snort**, and **iptables** required thorough research and practice. Troubleshooting errors, such as failed SSL connections or misconfigured firewall rules, was time-consuming and required patience and persistence. Another challenge was managing the volume of information and remembering the correct steps and commands for each task. Despite these obstacles, overcoming these challenges provided me with valuable hands-on experience and a deeper understanding of cybersecurity tools and techniques, preparing me for real-world applications.

Career Skills Obtained

Through this course, I have gained essential skills that are valuable for a successful career in cybersecurity. These include:

- **Network Traffic Analysis**: Proficiency in using **Wireshark** to capture, analyze, and interpret network traffic for threat detection and performance monitoring.
- **Intrusion Detection and Prevention**: Experience in deploying and configuring **Snort** to identify and respond to network-based attacks.
- **Cryptographic Analysis**: Understanding of **SSL/TLS protocols** and the ability to create, test, and decrypt encrypted communications using **OpenSSL**.
- **Live Memory Forensics**: Skills in analyzing system memory using tools like **Volatility**, **Process Hacker**, and **Process Monitor** to identify malicious processes and anomalies.
- **Firewall Configuration**: Competence in securing networks through **iptables** by implementing access controls, blackhole routing, and stateful inspection.
- **Problem-Solving and Troubleshooting**: Ability to troubleshoot complex configurations, identify errors, and resolve issues effectively across multiple environments.

These skills are foundational for roles such as **Security Analyst**, **Network Administrator**, **Incident Responder**, and **Forensic Investigator**, providing the technical expertise needed to protect and defend digital assets.

References

- DeVry University SEC290 Course Materials
- Wireshark Documentation
- OpenSSL User Guide
- Snort User Manual
- Process Hacker Documentation
- Process Monitor Guide
- Volatility Framework Documentation
- iptables Manual and Linux Kernel Documentation