# Proactive Project: Detecting and Analyzing the Zeus Banking Trojan

## Preparation

Malware Installation on VM  
We set up a virtual machine (VM) running Windows 7 to isolate and execute the Zeus Banking Trojan. The malware, sourced from theZoo repository, was extracted using the password 'infected' to prepare it for simulation. This step ensures the malware is safely contained within an isolated environment, reducing any risk to external systems.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## Simulate Malware Execution

**Traffic Monitoring and PCAP File Capture**:  
Once the malware was executed, network traffic was monitored using Wireshark . The PCAP file generated during this process provided insights into Zeus's command-and-control (C2) communication patterns and other network activities.

A computer screen shot of a computer

Description automatically generated

**System Logs Post-Malware Execution:**  
We examined system logs to identify changes made by the Zeus Trojan to the operating system. This step was critical for linking network activities with system-level anomalies such as process creation and registry modifications.

A computer screen shot of a computer

Description automatically generated

.

## Memory Analysis

Memory Dump Using FTK Imager  
The memory of the infected VM was dumped using FTK Imager. This data was essential for conducting in-depth forensic analysis, allowing us to pinpoint active and injected processes associated with Zeus.

A screenshot of a computer

Description automatically generated

## Network Monitoring and Intrusion Detection

1. **Suricata Configuration and Alerts**  
   Suricata was configured to monitor network traffic on the VM. Default rules were used to detect common threats, and custom rules were written to identify Zeus-specific network patterns. Alerts generated by Suricata were forwarded to Splunk for centralized analysis.

A screen shot of a computer

Description automatically generated

**Command breakdown:**

* Runs the Suricata tool with superuser permissions
* Specifies the configuration file for Suricata. The suricata.yaml file contains the settings and rules used by Suricata to analyze traffic
* Tells Suricata to process the zoz.pcapng file, which is a packet capture file containing network traffic data. Suricata will analyze this file instead of live network traffic
* Specifies the directory (/home/kali/Desktop/suricata) where Suricata will save its log files and analysis results
* A screen shot of a computer screen

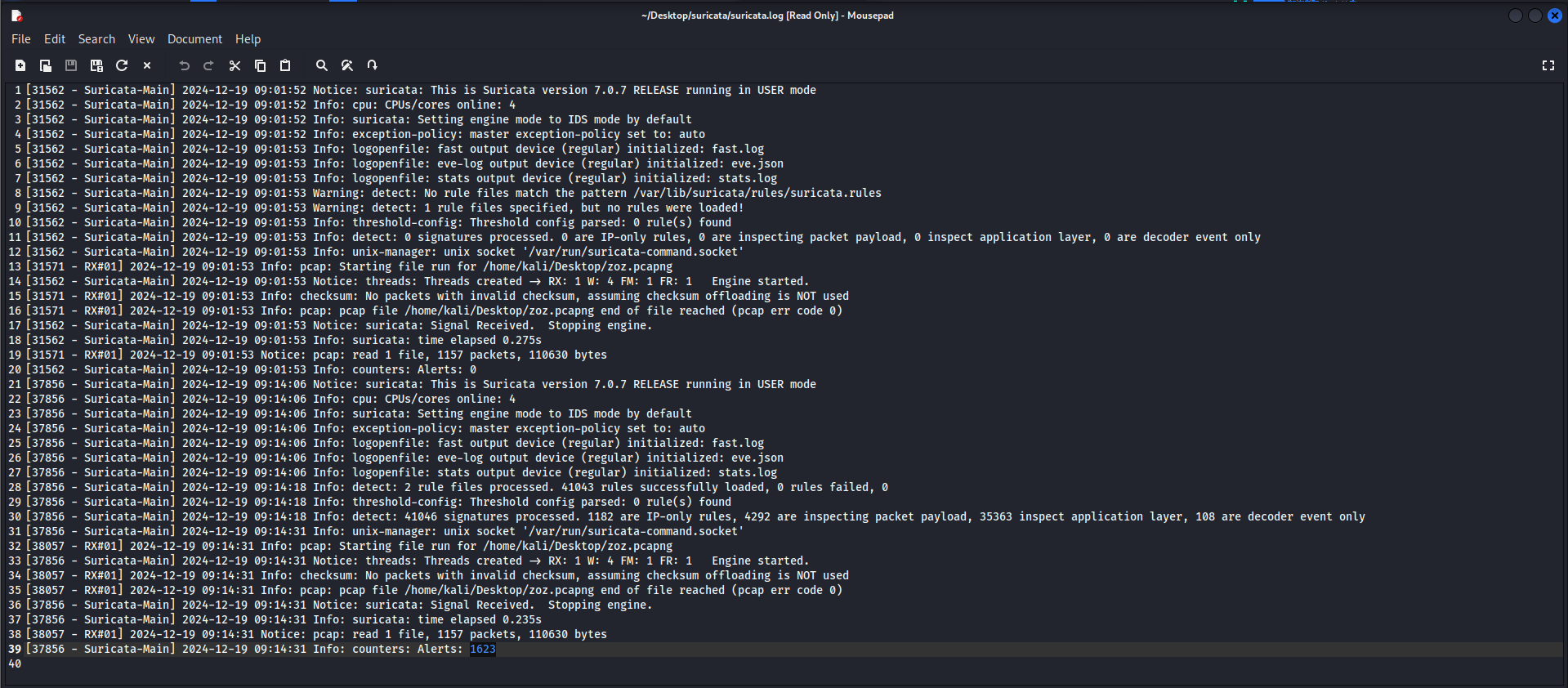
  Description automatically generatedSo now we see the suricata logs before adding the zeus rules
* A screen shot of a computer

  Description automatically generatedThen we modify the zeus rules
* Then we add the zeus rules to the rule files



* A screen shot of a computer screen

  Description automatically generatedThen we check the suricata logs after adding the zeus rules
* Then we check the suricata logs with both

A black background with white text

Description automatically generated

* So here as shown we see the alerts are increased

1. **Splunk Integration**  
   Logs from Suricata and the system were ingested into Splunk. Dashboards were created to visualize abnormal outbound traffic, correlate network anomalies with system activity, and track malicious operations in real time.

A screenshot of a computer

Description automatically generated

* Now we Delete all files and directories within the proactive folder. This ensures a clean slate for importing new logs. The prompt [y/n] asks for confirmation before proceeding
* Copies all Suricata log files (e.g., eve.json, fast.log, stats.log, suricata.log, systemlogs.json) from the /home/kali/Desktop/suricata directory to the proactive folder within Splunk
* **eve.json:** Contains detailed JSON-formatted event logs generated by Suricata.
* **fast.log:** A lightweight log format summarizing significant alerts.
* **stats.log:** Provides performance and traffic statistics of Suricata.
* **suricata.log:** The main log capturing Suricata's runtime activity.
* **systemlogs.json:** Likely system-specific logs formatted for analysis
* We change the inputs.conf



A screenshot of a computer program

Description automatically generated

* These configurations tell Splunk to monitor specific files or directories for ingestion and assign them specific attributes like an index and sourcetype

**Breakdown**:

[monitor:///home/kali/Desktop/suricata/eve.json]

index = proactive

sourcetype = \_json

* **Purpose**: Instructs Splunk to monitor the eve.json file generated by Suricata.
* **Index**: The data will be stored in the proactive index within Splunk.
* **Sourcetype**: The \_json sourcetype specifies that the data format is JSON. This helps Splunk parse and structure the logs effectively for searching and visualization

[monitor:///home/kali/Desktop/suricata/\*.log]

index = proactive

sourcetype = suricata\_logs

* **Purpose**: Directs Splunk to monitor all files with the .log extension in the /home/kali/Desktop/suricata directory.
* **Index**: Again, the data is stored in the proactive index.
* **Sourcetype**: The sourcetype suricata\_logs is used for these files. This is likely a custom sourcetype defined to categorize and parse Suricata logs effectively.

[monitor:///home/kali/Desktop/suricata/systemlogs.json]

index = proactive

sourcetype = \_json

* **Purpose**: Specifies that the systemlogs.json file, which also uses a JSON format, should be monitored by Splunk.
* **Index**: The data is added to the proactive index.
* **Sourcetype**: Like eve.json, the \_json sourcetype is applied, enabling JSON parsing

**A screenshot of a computer

Description automatically generatedNow we Open splunk and start searching and reporting**

* Here we check our created index to see the output

A screenshot of a graph

Description automatically generated

* This shows the distribution of events by their sourcetype. The sourcetypes correspond to the log formats you configured in inputs.conf

A screenshot of a computer

Description automatically generated

* This query identifies server-bound traffic in the Proactive index with the json sourcetype, grouping connections by source IP, destination IP, and destination port. It filters for connections with more than 50 occurrences, sorts them by frequency, and renames the columns for clarity. The results highlight the most frequent and potentially significant network traffic patterns.

A screenshot of a computer

Description automatically generated

* This query is used to detect unusual outbound traffic by filtering network events where the destination port is not a standard port (e.g., HTTP, HTTPS, or DNS) and then

summarizing the resultsA screenshot of a computer

Description automatically generated

* **Correlating Alerts and System Events**:
* Combines alerts (from Suricata or similar tools) with system events like file changes and process starts to identify patterns of malicious activity.
* Helps investigate if alerts coincide with suspicious system behavior.
* **Incident Analysis**:
* Provides a timeline of related events for specific IPs, enabling deeper analysis of potential threats.

A screenshot of a computer

Description automatically generated

* **Identify High-Risk Sources**:
* Quickly highlights which source IPs generated the most alerts, helping to pinpoint potentially compromised or malicious systems.
* **Prioritize Investigation**:
* A black screen with white lines

  Description automatically generatedBy sorting the data by alert count, it prioritizes IPs that require further analysis based on their activity levels

visualization

A screenshot of a computer

Description automatically generated

* **Traffic Analysis by Destination Port**:
* Identifies trends in traffic volume directed to specific ports over time.
* Useful for spotting unusual spikes or patterns in port usage, which might indicate malicious activity or network scans.
* **Time-Based Insights**:
* Helps visualize how traffic patterns change throughout the day or during specific time frames

A black and white image of a person

Description automatically generated with medium confidence

Visualization

## Volatility Analysis

Using Volatility, we analyzed the memory dump to identify suspicious processes. Processes such as 'services.exe' and 'svchost.exe' were flagged for having unusual memory permissions and potentially injected code. Network activity linked to high/random ports was also scrutinized to uncover C2 communications

A screen shot of a computer screen

Description automatically generated

* The command processes the provided memory dump (memdump.mem) to extract and display a list of active processes at the time the memory snapshot was taken.
* It downloads the necessary debugging symbols (from Microsoft) to decode the memory structures and accurately retrieve process-related information.
* So here the output didn’t find anything specific so I used malfind to identify potential malicious code injections or memory anomalies in a memory dump

A black background with white lines

Description automatically generated

* + A screenshot of a computer

    Description automatically generatedSo here is the list of suspicious processes

**Now we use the pstree to :**

**Process Relationships**:

* Helps trace the lineage of processes, identifying how they were spawned and their relationships to one another.

**Suspicious Process Analysis**:

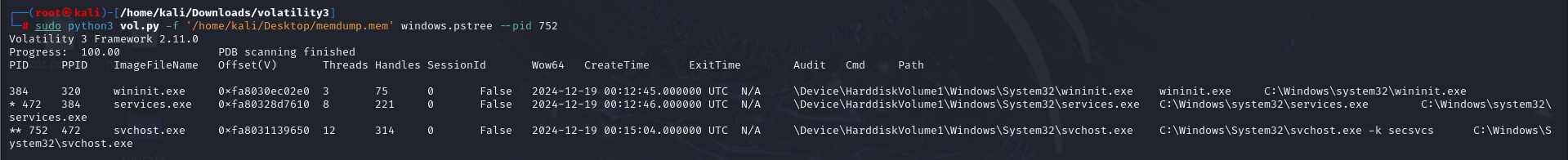
* If a parent process (e.g., explorer.exe) spawns unexpected child processes, it could indicate malicious behavior, such as process injection or privilege escalation

A black background with white text

Description automatically generated

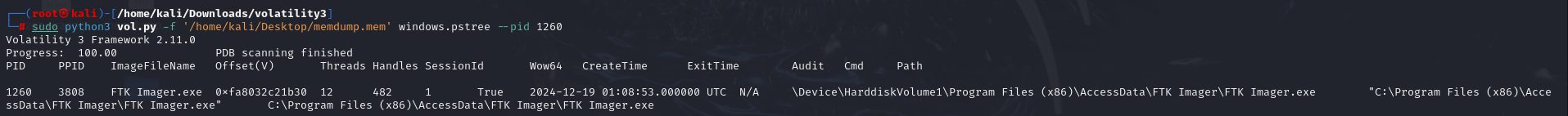
**Analysis of the Output**

* **Parent Process (PID 1328: explorer.exe)**:
  + This process is likely a parent to child processes, such as Wireshark.exe and dumpcap.exe.
  + The path and command-line arguments for explorer.exe appear legitimate, as they point to the Windows system volume.
* **Child Processes**:
  + **Wireshark.exe** (PID 2240):
    - Launched from the expected directory (C:\Program Files\Wireshark).
    - Indicates it was likely used for network analysis during this session.
  + **dumpcap.exe** (PID 3572):
    - Associated with Wireshark and used for packet capture, confirming legitimate usage.
  + **mmc.exe**:
    - A legitimate Microsoft Management Console executable, possibly for system monitoring or diagnostics.



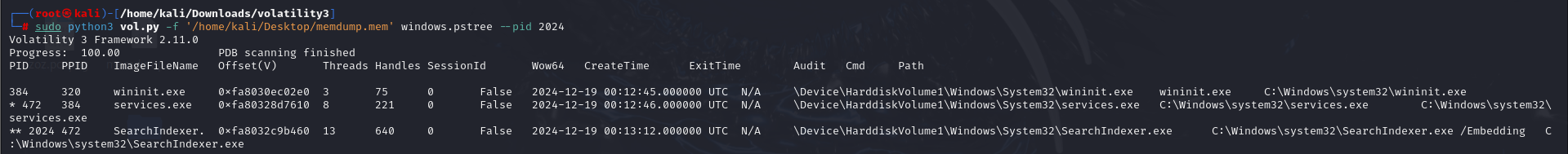
**Analysis of the Output**

1. **Parent Process**:
   * **wininit.exe (PID 384)**:
     + This process is responsible for initializing system services during the boot process.
     + It appears to be a legitimate parent process for services.exe.
2. **Child Process**:
   * **services.exe (PID 472)**:
     + Handles the management of system services and spawns svchost.exe processes.
     + Path and command-line arguments indicate normal activity.
3. **Focus Process (svchost.exe - PID 752)**:
   * **Command Line**: svchost.exe -k secsvcs
     + Indicates this instance of svchost.exe is managing the **Security Center Services** (secsvcs), which is legitimate behavior.



**Analysis of the Output**

* **Parent Process**:
  + PID 1080 appears to be the parent process that launched FTK Imager.exe.
  + This parent process could be investigated further if needed.
* **FTK Imager (PID 1260)**:
  + FTK Imager.exe is a legitimate digital forensic tool used to capture and analyze disk images.
  + The process details (e.g., path, command-line arguments, Wow64 environment) confirm that this is the standard executable running in 32-bit mode from its expected installation directory.
* **Legitimacy**:
  + No suspicious behavior or anomalies are evident in this output. The process appears legitimate and consistent with normal forensic use.



**Analysis of the Output**

1. **Parent Process**:
   * **services.exe (PID 472)**:
     + A legitimate system process responsible for managing services on Windows.
     + The fact that it launched SearchIndexer.exe aligns with typical Windows behavior.
2. **Focus Process (SearchIndexer.exe)**:
   * **What It Does**:
     + SearchIndexer.exe is a legitimate Windows process responsible for indexing files to improve search functionality.
     + The /Embedding argument is standard and indicates it is operating in embedded mode.
   * **Indicators of Legitimacy**:
     + The file path (C:\Windows\System32\SearchIndexer.exe) is the default location for this executable.
     + Command-line arguments and the parent process are consistent with normal Windows operations.

A black background with white dots

Description automatically generated with medium confidence

**Analysis of the Output**

1. **Parent Process**:
   * **wininit.exe (PID 384)**:
     + This is the first process in the chain, responsible for initializing system services during Windows startup.
     + It spawns services.exe (PID 472), which is responsible for managing system services.
2. **Child Processes of svchost.exe (PID 472)**:
   * Multiple svchost.exe processes (PIDs 1732, 856, etc.) are spawned under services.exe.
   * These processes manage different Windows services, such as:
     + **LocalServiceNetworkRestricted**.
     + **RPCSS** (Remote Procedure Call).
     + **seccs** (Security Services).
     + **Windows Management Instrumentation**.
3. **Other Processes**:
   * **taskhost.exe (PID 1052)**:
     + A legitimate Windows process that is responsible for running various host processes.
   * **SearchIndexer.exe** (PID 856):
     + A legitimate process responsible for indexing files for search functionality.
   * **SearchIndexer.exe** is launched by svchost.exe and operates under LocalSystem privileges.
4. **Legitimate System Processes**:
   * The svchost.exe and its child processes listed in the output, such as taskhost.exe and SearchIndexer.exe, appear to be running legitimate system functions. These processes are integral to system operations, handling tasks like service management, file indexing, and search optimization.

* **Now we use netscan to :**

**Purpose of This Query**

1. Identify active or recent network connections associated with processes.
2. Detect unusual or suspicious external connections to potentially malicious IPs.
3. Audit open and listening ports to identify possible vulnerabilities.
4. A screen shot of a computer

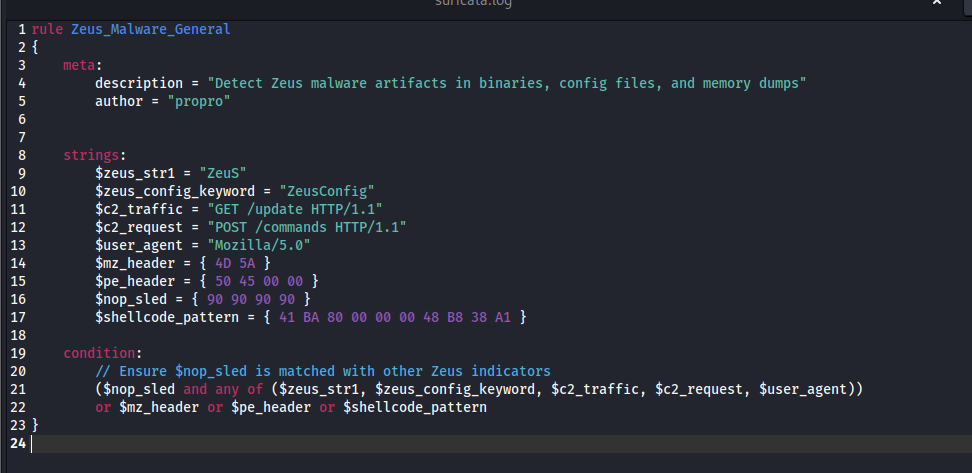
   Description automatically generatedProvide forensic insights into the state of network activity at the time of the memory dump.

**Output Observations**

1. **Listening Services**:
   * Several ports are in the LISTENING state for processes like services.exe, svchost.exe, and lsass.exe, indicating open services ready to accept connections.
2. **Suspicious Connections**:
   * **Foreign Address**: 56.171.191.50 (PID 532, svchost.exe):
     + A CLOSED connection to this external IP might warrant investigation to determine whether it was legitimate or part of malicious activity.
   * **Foreign Address**: 24.118.141.50 (PID 3800, firefox.exe):
     + A CLOSED connection associated with firefox.exe. If unexpected, this might indicate suspicious browsing or activity.
3. **Unusual Ports**:
   * High random ports (e.g., 49156, 16470) are commonly used for temporary or dynamic connections. However, their presence in processes like services.exe may require further investigation for potential abuse.
4. **Internal Connections**:
   * Local communication, such as 127.0.0.1:5238, suggests inter-process communication or local testing.
5. **Listening Ports on Well-Known Services**:
   * Ports like 135, 445, and 1900 are associated with Windows services such as RPC, SMB, and UPnP. While commonly used, these ports are often targeted by attackers

## Signature-Based Detection

YARA Rule Application  
Custom YARA rules were developed to detect Zeus-related patterns in binaries and memory dumps. These rules were applied to scan the infected system, enabling precise identification of Zeus artifacts.



**Rule Breakdown**

meta:

description = "Detect Zeus malware artifacts in binaries, config files, and memory dumps"

author = "propro"

* **description**: Provides a description of what the rule is for. In this case, it detects Zeus malware artifacts.
* **author**: Indicates the creator of the rule, which is "propro" in this case.

**Strings Section**

This section defines several strings and patterns that the rule will search for in the analyzed data.

1. **$zeus\_str1 = "ZeuS"**:
   * A string match for the word "ZeuS". This is a direct match for a known string found in Zeus malware.
2. **$zeus\_config\_keyword = "ZeusConfig"**:
   * Looks for the string "ZeusConfig". This could be part of configuration files related to the malware.
3. **$c2\_traffic = "GET /update HTTP/1.1"**:
   * Matches HTTP GET requests used for communication with the malware's command-and-control (C2) server. In this case, it's looking for a request to /update, which is characteristic of Zeus malware.
4. **$c2\_request = "POST /commands HTTP/1.1"**:
   * Looks for POST requests to /commands, another potential sign of C2 communication.
5. **$user\_agent = "Mozilla/5.0"**:
   * Matches the "Mozilla/5.0" user-agent string. Zeus may use this in its HTTP requests, so detecting this string in the traffic could indicate Zeus activity.
6. **$mz\_header = { 4D 5A }**:
   * This is a byte pattern (4D 5A), which represents the "MZ" header found at the beginning of Windows executable files (PE files). This is used to match the header of executable files, which could be part of Zeus malware.
7. **$pe\_header = { 50 45 00 00 }**:
   * This is another byte pattern (50 45 00 00), which represents the "PE" header found in Windows Portable Executables (PE files). It's used to identify Windows executables, which may contain Zeus malware.
8. **$nop\_sled = { 90 90 90 90 }**:
   * This matches a sequence of NOP (No Operation) instructions (90 in hexadecimal). A "NOP sled" is commonly used in shellcode to align the payload in memory. This may indicate the presence of shellcode or an exploit attempt related to Zeus.
9. **$shellcode\_pattern = { 41 BA 80 00 00 00 48 B8 38 A1 }**:
   * This is a byte pattern (41 BA 80 00 00 00 48 B8 38 A1) that could represent part of the shellcode used by Zeus malware. It is specific to the Zeus payload and is used to detect the presence of shellcode in memory or a binary

condition:

// Ensure $nop\_sled is matched with other Zeus indicators

($nop\_sled and any of ($zeus\_str1, $zeus\_config\_keyword, $c2\_traffic, $c2\_request, $user\_agent))

or $mz\_header or $pe\_header or $shellcode\_pattern

* The condition defines the logic for when the rule should trigger.

1. **($nop\_sled and any of ($zeus\_str1, $zeus\_config\_keyword, $c2\_traffic, $c2\_request, $user\_agent))**:
   * This part ensures that if a **NOP sled** ($nop\_sled) is found, one of the other Zeus-related indicators must also be present. This is useful for detecting shellcode used by Zeus.
2. **or $mz\_header or $pe\_header or $shellcode\_pattern**:
   * If any of the following are found:
     + The **MZ header** ($mz\_header), indicating an executable file.
     + The **PE header** ($pe\_header), indicating a Windows executable file.
     + The **shellcode pattern** ($shellcode\_pattern), indicating possible shellcode from Zeus.

The rule will trigger if any of these conditions are met