# Global Information Assurance Certification Paper

# Man-In-the-Middle Attack

**-A Brief**

**Author:** **Bhavin Bharat Bhansali**
**Submitted on:** **February 16, 2001**

**Objective:**
The Objective of this document is to understand the Execution of "Man-In-the-Middle" attack.

**Overview:**
The "Man In The Middle" or "TCP Hijacking" attack is a well known attack where an attacker sniffs packets from network, modifies them and inserts them back into the network. There are few programs/source codes available for doing a TCP hijack. **Juggernaut**, **T-Sight** and **Hunt** are some these programs. In this paper we shall explore Hunt for understanding how TCP Hijacking is deployed on an Ethernet segment.
**Hunt** is designed by kra kra@gncz.cz. The **Hunt** source code is available at the following URL:

ftp://ftp.gncz.cz/pub/linux/hunt/hunt-1.5.tgz

**Relevance:**
TCP Hijacking is an exploit that targets the victims TCP based applications like Telnet, rlogin, ftp, mail application, web browser etc. An attacker can grab unenrcypted confidential information from a victim's network based TCP application. He can further tamper the Authenticity and Integrity of the data.

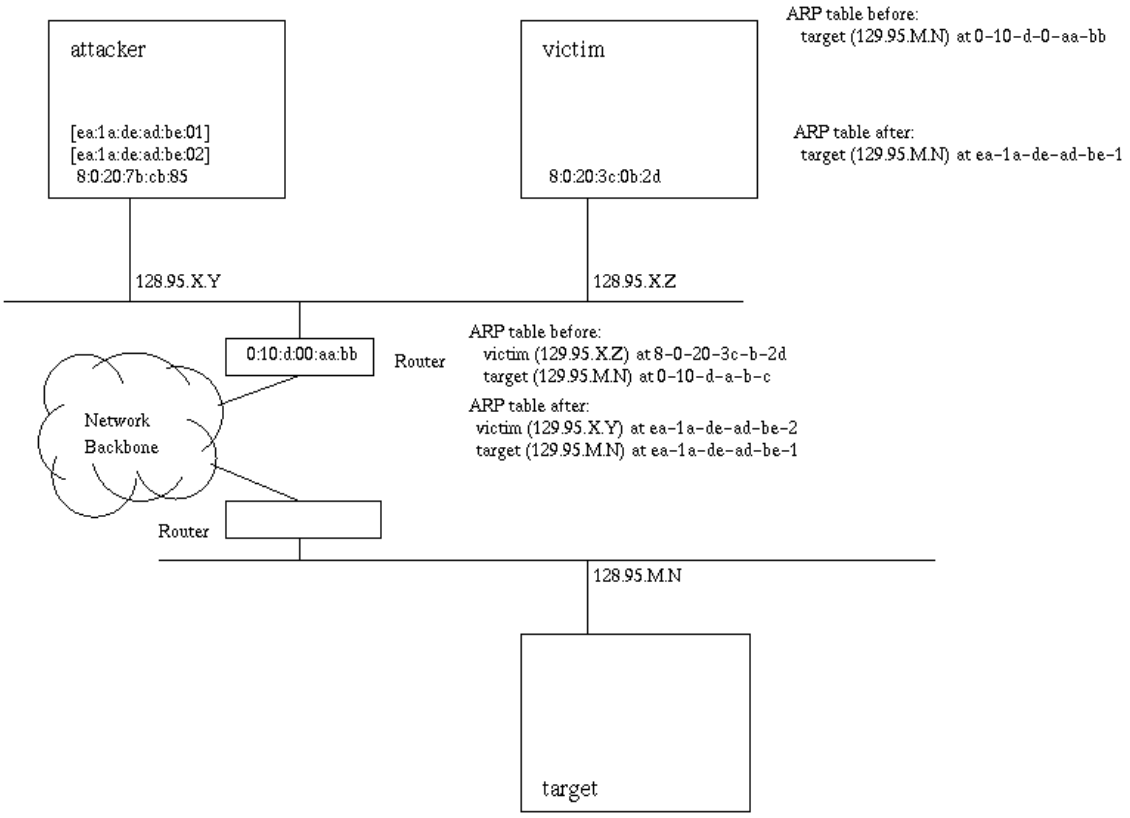
**Definition of Important Terms:**

- IP spoofing - IP spoofing involves forging one's source IP address. It is the act of using one machine to impersonate another. Many applications and tools in UNIX systems rely on source IP address authentication.

- ARP spoofing - ARP spoofing involves forging packet source hardware address (MAC address) to the address of the host you pretend to be.

- Simple Active Attack against TCP connections - An attack in which the attacker does not merely eavesdrop but takes action to change, delete, reroute, add, forge or divert data. Perhaps the best-known active attack is Man-In-the-Middle.

**The Attack:**




Attack Scenario involves three hosts: **Attacker**, **Victim**, and **Target**.

- **Attacker** is the system used by the attacker for the hijack.
- **Victim** is the system used by the victim for Telnet client connections to the target system.
- **Target** is the target system that the intruder wants to compromise. It is where the telnetd daemon is running.

A simple diagram of the network shows the **Attacker** and **Victim** hosts are on the same network (which *can be Ethernet switched and the attack will still work*), while the **target** system can be anywhere. (Actually, either **victim** or **target** can be on the same network as **attacker**: it doesn't matter.)

For the attack to succeed, the victim must use Telnet, rlogin, ftp, or any other non-encrypted TCP/IP utility. Use of SecurID card, or other token-based two-factor authentication is useless as protection against hijacking, as the attacker can simply wait until *after* the user authenticates, then hijack the session.
The attack scenario can be as simple as:

1. **Attacker**: Spends some time determining the IP addresses of **target** and **victim** systems. Determining trust relationships can be easily done with utilities like

SATAN, finger, systat, rwho or running who, ps, or last from previously stolen (or wide open "guest" style) accounts.

2. **Attacker**: Runs **hunt** as root on attacking host. Waits for **hunt** to indicate a session has been detected.
3. **Attacker**: Starts ARP relay daemon[1], prepares RST daemon[2] entry for use later, sets option to enable host name resolution (for convenience).
4. **Victim**: Logs in to target using Telnet. Runs pine to read/compose email.
5. **Attacker**: Sees new connection[3]; lists active connections to see if this one is potentially "interesting." If it is, attacker can either watch the session (packet sniffing) or hijack the session. Decides to hijack.
6. **Victim**: Sees strange new prompt. Tries pressing RETURN and doesn't know what to think. Tries web browser and notices that it still works fine (not a network problem). Not sure what to think.
7. **Attacker**: Finds this is a user session and decides to give it back (resynchronizes TCP/IP stream).
8. **Victim**: Sees prompt for keystroke, follows request, gets session back. Puzzled, decides to log in to root account to take a closer look.
9. **Attacker**: Turns on RST daemon to prevent new connections, waits to hijack root session.
10. **Victim**: Runs ssu to get SecurID protected root shell.
11. **Attacker**: Completes hijack after seeing root login.
12. **Victim**: Sees strange prompt. Tries pressing RETURN again. Same result as before. Tries web browser again. Same thing. Tries getting a new Telnet session. Fails. Tries ftp. Fails.
13. **Attacker**: Sets up backdoor, disables command history, resets session, turns off RST daemon.
14. **Victim**: Finally gets a new session. Original session is now gone. Assumes network outage or Windows TCP/IP stack corruption. Reboots system and everything is back to "normal".
15. **Attacker**: Waits for admin's sessions to all disappear (gone home for the night), then logs in using new backdoor. Installs rootkit (more backdoors, sniffer), cleans log files.

[1] Pls. refer to the "ARP Daemon" paragraph of the "Brief Overview of the Daemons / threads that are used by the exploit" topic

[2] Pls. refer to the "Reset Daemon" paragraph of the "Brief Overview of the Daemons / threads that are used by the exploit" topic

[3] Pls. refer to the "Sniff Daemon" paragraph of the "Brief Overview of the Daemons / threads that are used by the exploit" topic

**Design Overview:**
The development model is based on a packet engine (**hunt.c**) which runs in its own thread and captures packets from the network. The packet engine collects information of TCP connections/starting/termination, sequence numbers and MAC addresses. It collects the MAC addresses and sequence numbers from the server point of view and separate MAC addresses and sequence numbers from the client point of view. So it is prepared for hijacking. This information (seq. num., MAC, etc.) is available to modules so they don't have to analyze and collect it.

Modules can register functions with the packet engine, which are then invoked when new packets are received. A module function determines if the module is interested in a packet or not and can place the packet in a module specific list of packets. A module function can also send some packet to the network if it is desirable to do it very fast. The module (usually in some other thread so it needs to be scheduled to be run) then gets packets from the list and analyzes them. In this way, you can easily develop modules, which perform various activities. Refer to the appendix section for learning the features offered by hunt.

**Brief Overview of the Daemons / threads that are used by the exploit:**

- Reset daemon
  Reset daemon is used to perform automatic resets of ongoing connections that **hunt** can see. You can describe which connections should be terminated by giving src/dst host/mask and src/dst ports.

- ARP daemon
  ARP daemon is used to do ARP spoofing of hosts. You enter src and dst addresses and desired src MAC. The dst is then forced to think that src has srcMAC. You can use some fake MAC or better MAC of host that is currently down.

- Sniff daemon
  Sniff daemon can log specified packets. The sniff daemon can also search for a simple pattern (string) in the data stream (see the bugs section). You can specify which connection you are interested in, where to search (src, dst, both), what do you want to search, how many bytes you want to log, from what direction (src, dst, both) and to what file should the daemon write.

- MAC discovery daemon
  MAC discovery daemon is used to collect MAC addresses corresponding to the specified IP range.

**References:**


- Krauz's, Pavel. "HUNT Project." 1.5 - bug fix release. 30th May 2000.
  URL: http://lin.fsid.cvut.cz/~kra/index.html#HUNT(9th February, 2001)

- Dave, Dittrich. "Session hijack script". 9th Dec 1999.
  URL: http://staff.washington.edu/dittrich/talks/agora/script.html

- Dave, Paras. "TCP Connection Hijacking".
  URL: http://cs.baylor.edu/~donahoo/NIUNet/hijack.html (12th February, 2001)

- Microsoft technical support. "Microsoft Security Program: Microsoft Security Bulletin (MS99-046) "December 23, 1999.
  URL:http://www.microsoft.com/TechNet/security/bulletin/ms99-046.asp(12th February 2001)

- Stevens, Richard. "TCP Connection Establishment and Termination". October 1993.
  Book Name: TCP/IP illustrated Volume I - The Protocols (7th February 2001).

- Kurtz, George. "Session Hijacking". July 28th, 1999.
  Book Name: HACKING EXPOSED - Network Security Secrets and Solutions (8th February, 2001)

## Appendix

**Features of the hunt exploit:**

- Connection Reset - With a single properly constructed packet you can reset the connection (RST flag in TCP header). You can reset server, client, or both. When you reset only one end the other end is reset when it tries to send data to the first host which will response with RST because of the connection reset on it.

- Connection sniffing/watching - You can watch hunt output for any connection which you choose from the list that hunt displays on the console.

- ARP-relay - You can insert packets to the network (rerouting) of all data it receives from ARP spoofed hosts.

- Connection Synchronization - This is one of the main features of hunt. If you put some data to the TCP stream (through simple active attack or ARP spoofing), you desynchronize the stream from the server/original client point of view. After some work done on that connection you can just reset it or you can try to synchronize both original ends again. The main goal behind this is to synchronize the sequence numbers on both client and server again.

- Switch/Segment traffic rerouting - With ARP spoofing you can force the Switch to send you the traffic for hosts on another segment/switched port. This may not work if the Switch has some security policy and MACs have been explicitly set up on a per port basis but in reality this configuration is hardly done on an "ordinary" network.

- ACK Storm - The ACK storm is caused by majority of TCP stacks Let's imagine that you send some data to an ongoing connection to the server (as if sent by the client - with expected seq. numbers, etc.). The server responds with the acknowledgment of the data you sent but the original client receives this acknowledgment too. But from the original client point of view, the server has acknowledged data that doesn't exist on the client. So something strange occurred and the original client sends the "right" sequence number with ACK to the server. But the TCP rules say that it is required to generate an immediate acknowledgment when an out-of-order segment is received. This ACK should not be delayed. So the server sends the acknowledgment of non-existent data to the client again and the client responds.

- **Bhavin Bharat Bhansali**
  - February 16, 2001