CSE 5473: Network Security

Final Project Report

Derek Coventry
Dikai Xiong
Jonathan Seaman
Michelle Vavra

Introduction

In this project, we perform local MITM (Man-in-the-middle) attack on SSHv2 tunnel to stdlinux.cse.ohio-state.edu.

Systems:

- Victim: Ubuntu 16.04
    - Victim IP: 10.0.2.4
    - Gateway IP: 10.0.2.2
- Attacker: Kali Linux

Programming Lanugage:

- Python 2.7.14

Libraries:

- scapy
- pycryptodome
- sshpubkey

The SSHv2 tunnel to stdlinux.cse.ohio-state.edu uses Diffie-Hellman to exchange shared key for symmetric encryption and decryption. “diffie-hellman-group-exchange-sha256” is used as the exchange method. The negotiated algorithms are ssh-rsa, aes128-ctr, and umac-64@openssh.com, by default.

Our MITM can successfully establish a shared key with the server. However, the client will run into **ssh_dispatch_run_fatal: error in libcryto**, after the user answers “yes” during RSA key verification. The decryption of packets using shared keys is yet to be implemented.

Source: https://github.com/keddybear/cse5473-redteam-final-project

Instructions

Step 1: Setup

Use VirtualBox to run both Victim (Ubuntu) and Attacker (Kali Linux). Configure their IP addresses and make sure they are connected to the Internet and that they can PING each other.

Step 2: ARP Poisoning

In Kali Linux, run mitm.py

This file will trick the client (10.0.2.4) into believing that we are the gateway (10.0.2.2), and trick the gateway (10.0.2.2) into believing that we are the client (10.0.2.2), by sending them ARP packets where we replace their respective mac addresses with ours.

Step 3: Capture, Modify, and Forward

In Kali Linux, run sniffer.py

This file will capture packets sent from the client and server, modify them, and forward them to their intended destinations.

pretender.py

sniffer.py utilizes a pretender class that processes captured packets and has a fake_client and a fake_server object to store needed key exchange information.

Capture

Since packets are not encrypted before the key exchange is complete, the following information is store in pretender class:

Protocol Exchange

V_C	Client's SSH version string (CR and NL excluded)
V_S	Server's SSH version string (CR and NL excluded)

Key Exchange Init

I_C	The payload of the client's SSH_MSG_KEXINIT
I_S	The payload of the server's SSH_MSG_KEXINIT

DH Group Exchange Request

min	Minimal size in bits of an acceptable group
n	Preferred size in bits of the group the server will send
max	Maximal size in bits of an acceptable group

DH Group Exchange Group

p	Safe prime (modulus p)
g	Generator for subgroup (base g)

DH Group Exchange Init

e	Exchange value sent by the client

DH Group Exchange Reply

f	Exchange value sent by the server

Modify

After *DH Group Exchange Group*, pretender will replace exchanges values from the client and server with its own e and f.

In the final step, *DH Group Exchange Reply*, pretender will also replace the host key from the server and compute its own hash.

Forward

Lastly, sniffer.py will ensure all modified packets retain the same length as the original ones, so that acknowledge numbers as well as sequence numbers are not out of order.

Step 4: Connect

In Ubuntu console, enter:

```
ssh {username}.{number}@stdlinux.cse.ohio-state.edu
```

This command will terminate connection automatically when host key verification fails.

However, the following command will ask the user to accept the unknown host key by typing “yes” or “no”:

```
ssh -o "UserKnownHostsFile=/dev/null" {username}.{number}@stdlinux.cse.ohio-
state.edu
```

Obstacles

1. Modifying the length of packets will cause acknowledgment numbers to be out of sequence. Even manually fixing them will still cause spurious retransmission at some point. Cause is unknown.

Solution:

This obstacle is circumvented by adding useless data to packets so that their length remains the same.

2. Server will disconnect if the public DH value sent by client is “bad”. The definition of “bad” is unknown.

Solution:

Different values are tried. Eventually, one is found, and it does not take MITM too long to compute a fake public DH client value.

Fatal Error

ssh_dispatch_run_fatal: error in libcrypto

Cause: unknown