NAME: JOAN NDOLWA MOYI

REG NO: ENE212-0064/2016

UNIT CODE: EEC2505

UNIT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

LAB 1: MAN IN THE MIDDLE AND OTHER NETWORK ATTACKS

OBJECTIVES

1. To understand the basics of network sniffing
2. To understand the basics of replay attacks
3. To understand the basics of insertion attacks
    a. including the development of filters for Ettercap
    b. including common programmatic defenses against replay attacks
4. To understand the general concepts of man-in-the-middle attacks
5. To execute sniffing, replay, and insertion attacks against cleartext data
6. To execute MITM attacks against strong cryptography

INTRODUCTION

Eavesdropping

Eavesdropping generically refers to intentionally listening to a private conversation. In the context of computer networks, this is often called *sniffing*, and involves receiving and decoding network packets on one's network segment that are not sent by or meant to be received by your host.

The usefulness of sniffing is immediately obvious when the data is being passed in plain text, because any "cleartext" data sniffed is data that can be immediately read. Usernames and passwords are often easy to extract with a little protocol knowledge or an ASCII decoding of the network data.

In the old days, almost no network traffic was encrypted because it required more memory and CPU power, which were both considered too costly when compared with the likelihood of meaningful losses caused by a sniffing attack. As a result, the TELNET remote shell program remained popular into the late 1990s even though all data and authentication information were sent in cleartext.

Nowadays however, sniffing is one of the most common and easiest attacks to execute due to the proliferation of computer networks, free user-friendly sniffing tools and especially wireless LANs where sniffing requires only proximity and a laptop. Most web traffic is unencrypted, together with many other protocols such as SMTP and IMAP (email). Most remote shell traffic is now encrypted due to the tremendous risk of sending system authentication information unencrypted, but SMTP, IMAP, or web forum passwords are often still unencrypted and they may be the same as more sensitive system passwords. (For example, John's unencrypted webmail password may be the same as his login password on the same system, even though the system uses SSH for logins.) Additionally, while TELNET is not often used for sensitive internet traffic, it is still used in many internal networks (like companies and banks) because of legacy systems that cannot handle encryption.

Tools such as tcpdump and Wireshark are both very powerful, freely available sniffing programs and are available for most computer platforms in use today.

Replay attacks.

Replay attacks can be more damaging and successful than mere eavesdropping because they can cause remote actions -- sometimes even against cryptography.

The basic idea of a replay attack is to capture packets sent on a wire between two hosts with the intent to later replay the payloads (or more rarely the same packets) in order to cause the same result. For example, most web-based CGIs are extremely susceptible to primitive replay attacks because they are stateless and respond to a single web request with the appropriate information in the request. This request is usually built up over several page requests, but can often be triggered by submitting a full request at once.

Imagine an online greeting card application that has several steps, all managed by the same CGI application, with all input saved in the background (traditionally done with hidden form fields):

1. The first page takes your login name and password: (user: *rms*, password: *gnu*)
2. The second page takes the address of the recipient: *esr@catb.org*
3. The third page selects an image: *h2obuffalo.jpg*
4. The fourth page provides an entry form for the message: *Happy Hacking!*
5. The fifth page previews, submits and sends the card.

This process would normally take 5 steps by a human, with the possibility of going backwards and forwards to edit form fields. However, the whole process can often be summed up in one HTTP request like this:

http://example.com/card.cgi?login=rms&pass=gnu&addr=esr@catb.org&img=h20buffalo.jpg&msg="Happy Hacking!"&submit=submit

Loading that single URL will skip the whole process and send the above card directly. This is a simple example of the "payload" type of replay attack, where the commands included in a network stream are reissued to a server.

While this URL could be sniffed and entered manually, one can imagine capturing the entire stream of packets (including the TCP handshake) that generated the single HTTP request that resulted in sending the card. After the capture, we might simply re-inject the packets into the network. While this seems like it should work, TCP and other protocols use sequence numbers that are chosen pseudo-randomly upon connection initiation. Without valid sequence numbers, a naive "replay" of packets into the network will be rejected by the server. This is why "payload" replay attacks are popular, and incidentally highlights one reason why unpredictable numbers are critically important in security.

On the other hand, some protocols, such as UDP, do not necessarily use sequence numbers. In this case, a direct replay attack of a data stream could be successfully processed by a server. One can imagine replaying a network capture back onto the network, ignoring any response from the server or replaying the appropriate packets if necessary. If done properly, it would look as though those packets simply "appeared" in the network, and because their headers would already contain the information from the *original sender*, the traffic would appear as though it were coming *from* the original sender, not the host executing the replay attack. Regardless, the result is the same, using commands that are known to be valid to achieve the same result without authentication.

Regardless of whether you are using "direct" packet replay attacks or more complicated "payload" replay attacks, both can be used for many purposes. One can easily imagine a web or other network conversation giving a user a raise or increasing or decreasing a balance of some kind, from something mostly harmless (like World of Warcraft hit points) or something more significant, like an airplane ticket purchase. If that conversation can be replayed, the same effect can be obtained repeatedly.

"Direct" Replay attacks work when there is nothing *unique* about particular transactions other than the request parameters or data payload. The main defense against such an attack is unpredictable (but easily validated) session tokens in the communication that tie a sequence of packets to a unique request. It is not enough to use the current time or a hash of the packet data, because this can easily be guessed or recalculated in a replay attempt.

Insertion attacks.

Insertion Attacks are essentially replay attacks where an attacker changes existing or inserts new data in the network conversation. For example, in the e-card example, an insertion attack could have changed the message in the replayed card from "Happy Hacking" to "You have been hacked”. Furthermore, this is a less impressive kind of insertion attack because it relies on the first e-card being sent before the modified message could be replayed.

In fact, typically, an "insertion attack" means that the attacker has the ability to capture, modify, and resend valid packets *as though they were the original data stream*. At the very least, this involves capturing the packet, editing, updating any relevant header information (especially the data payload checksum), and resending the modified packet.

This is a "man in the middle" attack in the sense that it requires an entity on the network that has the capability to block transmission of the original packets and send the modified packets. (We usually imagine this entity directly between two communicating parties, but that need not physically be true.) Sophisticated insertion attacks can even modify the responses in order to make the original host think everything has gone according to plan.

Sometimes people use "man in the middle" loosely to refer to any kind of network attack. However, the canonical Man in the Middle attack refers to a specific kind of attack against cryptosystems.

One extreme way of doing this is to insert a computer in the middle of the network that acts as a gatekeeper and is capable of modifying traffic. In this sense, an application proxy performs beneficial "insertion attacks" on a network. For example, a web proxy can exchange all graphics for locally cached copies if they exist. In fact, this is one technique that many "web accelerators" use, they replace images and other content with compressed versions, or in the case of images, lower quality lossy-compressed versions.

A stealthier way of performing insertion attacks is using ARP Spoofing as described above to convince hosts to send their data to the attacker *first*, who will then typically forward the data on after modifying it. This doesn't require modifying the physical nature of the network in any way, and can often be performed undetected from anywhere (even across the Internet) as long as the attacker can directly control an interface on the LAN (e.g. after exploiting a software vulnerability to take over a host).

Encryption can obviously hinder the opportunities for successful insertion attacks, but poor use of encryption can still leave open the opportunity to decrypt the packet, change it, and re-encrypt the packet before resending. This is especially true if a cryptographic nonce or other kind of session token can be reverse engineered, if a correct nonce can be generated or recalculated (like a TCP checksum) the nonce has lost its usefulness.

Man in the middle attacks.

The man in the middle attack works like this:

Alice and Bob want to encrypt their chat session, but neither Alice nor Bob have each other's public keys. In order to communicate securely, Bob and Alice need to send their public keys (bob.pub and alice.pub) to one another. Once both hosts have each other's public keys, they can begin encrypting their packets, secure in the knowledge that only the intended recipient can decrypt the data. However, imagine there is a user in the middle of the network, known as Eve. Eve wants to listen in on the conversation between Alice and Bob. In order to do this, Eve runs an application that accepts the real public keys alice.pub and bob.pub from Alice and Bob, but sends them both different *bogus* public keys, *bob1.pub* and *alice1.pub*. Bob and Alice don't know any better and will use the bogus public keys instead of the real public keys to encrypt packets for each other. Both Alice and Bob believe that they are using the correct public keys, and expect to decrypt the messages with their own private keys. How can they use their original private keys to decrypt these bogus messages?

The answer is that Eve (as the "Man" in the Middle) captures and decrypts the packets with the private keys for the bogus bob1.pub and alice1.pub, inserts or merely eavesdrops on the connection, and re-encrypts the data with the *original* public keys that Alice and Bob sent (alice.pub and bob.pub) and forwards the packets to their original destination. Bob and Alice decrypt the packets with their own private keys, and so don't realize that their connection has been hijacked because from their perspective, the session behaved **perfectly**.

Modern implementations of public key cryptography for network services rely on several different secure key exchange methods. By default, SSH will pass the public key to the client the first time in cleartext. During this first key exchange, the connection is vulnerable to a MITM attack, and if this is a serious concern, the key should be installed by hand from trusted media or encrypted with preexisting trusted keys before being sent over an insecure network.

After the initial key exchange, the public key is saved and associated with the IP address in the ssh configuration file~/.ssh/known_hosts (or the Windows Registry if using PuTTY), so that if a different key is sent by a man in the middle (see the discussion of HTTPS below) or even if it is legitimately changed on the remote host it will be noticed and an alert will be generated. Compromise happens either by intercepting the initial exchange or by tricking a user into accepting an illegitimately changed key.

"Man-in-the-middle" is a term used in cryptography to describe scenarios where an attacker (the eponymous "man in the middle") between two remote parties can view or control data that would otherwise be secure. "Man-in-the-middle attack" usually refers to vulnerabilities in a key-exchange protocol whereby an attacker can subvert the encryption (typically by substituting secure keys with insecure keys) and gain access to the cleartext without the victims' knowledge. Man-in-the-middle attacks are possible due to characteristics of common networking protocols that make eavesdropping and other "insecure" activities possible.

PROCEDURE

1. I swapped into the lab.
2. I ssh into *users.isi.deterlab.net*
3. I verified that I could not log in to alice or bob by trying ssh alice or ssh bob
4. I logged into eve by ssh eve
5. Executed sudo su - to become root.
6. I used ifconfig to identify which interface is on the 10.x.x.x network.
7. Ettercap procedure:

   - Run ettercap using the command *ettercap -C*
   - Prepared the attack. In ettercap (using the menus), start "Unified Sniffing" under the "Sniff" menu.
   - Then, under the "Hosts" menu, "Scan for Hosts". Also, under "Hosts", choose "Hosts list". Highlight a host and press '1' to add it to a target group. Highlight the other host and also press '1'.
   - Under the "Mitm" menu, choose "Arp poisoning". Set the property to remote.
   - Under the "Start Menu" choose "Start Sniffing".

8. Eavesdropping:

   - I opened another ssh session and logged into eve.
   - On this session I captured data with tcpdump using the following command *tcpdump -i eth2 -s0 -w output.pcap.*
   - Unpacked the data with chaosreader using the command *chaosreader output.pcap*
   - I then downloaded all the html chaos reader output files onto my local pc using the command *scp -r 22 jkuatkaousers.isi.deterlab.net:/users/jkuatkao*.html c:\Users\user\Desktop\Chaos* on windows command prompt.

9. I used the downloaded html files for the eavesdropping and replay attacks.
10. For the replay attack I ssh tunnelled into bob using the command *ssh -L 8080:pc032:80 jkuatkao@users.deterlab.net*. That enabled me to access bob on my local pc's browser on port 8080.
11. For the insertion attack:

    First filter:

    - I created a file using the command *nano etc/ettercap/etter.filter* and wrote the filter code in the file.
    - From the Ettercap directory, I compiled the filter using *etterfilter etter.filter -o filter.ef*

    Second filter:

    - I created a file using the command *nano etc/ettercap/etter.filterone* and wrote the filter code in the file.
    - From the Ettercap directory, I compiled the filter using *etterfilter etter.filtero -o filtero.ef*

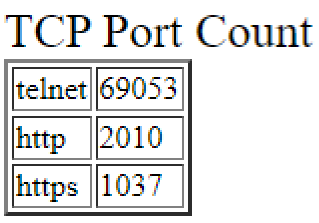
    - I started Ettercap with procedure 7 listed above.
    - Before starting sniffing, I had to add the filter. Under "Filters", choose "Load a filter". Load filter.ef' or filterone.ef.
    - Success messages in the filter code were displayed on the Ettercap console.
12. MITM vs Encryption.
    - I editted /usr/local/etc/etter.conf and uncomment the two iptables lines required for MITM against SSL.
    - I installed sslstrip.
    - I ran *sslstrip –a –l 8080* for some time then ctrl+C
    - I downloaded the sslstrip log file to my pc and analyzed its content.

RESULTS AND DISCUSION.

EAVESDROPPING.

1. **What kind of data is being transmitted in cleartext?**

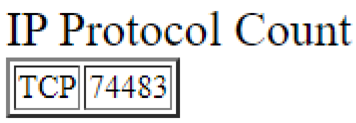
- **What ports, what protocols?**



TCP Port Count

telnet	69053
http	2010
https	1037

Port	**Port number**
https	443
http	80
telnet	23



IP Protocol Count

TCP	74483

- **Can you extract identify any meaningful information from the data? e.g., if a telnet session is active, what is happening in the session? If a file is being transferred, can you identify the data in the file?**

  The data being transmitted includes the usernames, passwords and scripts.

  The scripts being transmitted include:

  i. stock.cgi used for stock information modification.
  ii. access.cgi used for a conversation which appears to be the script of the movie The Matrix.

2. **Is any authentication information being sent over the wire? e.g., usernames and passwords. What usernames and passwords can you discover?**

   Yes, authentication information is being sent over the wire. It includes the usernames and passwords of following users:

USERNAME	**PASSWORD**
jumbo	donald78
jambo	minnie77
jimbo	goofy76

- **Note: the username and password decoding in ettercap is not perfect -- how else could you view plain text authentication?**

Ettercap displays usernames and passwords but the passwords appear as screen instead of the actual passwords.

I used tcpdump to capture the traffic in a file named output.pcap. Then used chaosreader to analyse the output.pcap .

3. **Is any communication encrypted? What ports?**

The communication using HTTPS protocol was encrypted on port 443.

REPLAY ATTACKS.

1. **Explain exactly how to execute the attack, including the specific RPCs you replayed.**

   The replay attack involved ssh tunnelling on my local web browser to connect to stocks.cgi interface using the Bob's account.

   The URLs listed below were used to make the attack.

   http://localhost:8080/cgi-bin/stock.cgi?symbol=ZBOR&new=99&hash=f82cbf7a6603e6119d1b0ee3b585ec0f

   http://localhost:8080/cgi-bin/stock.cgi?symbol=FZCO&new=41&hash=6fa11c1ac00566dfc38d8398ac87122f

   The first URL was used to set the ZBOR stock price to $99 and the second URL was used to set the FZCO stock price to $41. I used the above URLs because we were required to make FrobozzCo's stock look bad and Zembor Corp's stock look good.

2. **Explain how you determined that this strategy would work.**
   - While analysing the output.pcap file I found that Bob received data to update the stock price through the stock.cgi script.
   - I analysed the different updates and concluded that all updates had a unique value and hash.
   - I then found sessions that were specific to FZCO and ZBOR. I picked the URLs corresponding to the highest set value for ZBOR and the lowest set value for FZCO.
   - I copied the specific URL into the tunnelled browser and the stock data was updated.
   - I noted the hash value was not time dependent and you could use the same URL for consecutive updates. However, the hash value is specific to a share and a value.
   - I performed all the replay attack via the ssh tunnelled browser on port 8080.

3. **Execute your replay attack and show the results of your attack with a screen capture, text dump, etc. showing that you are controlling the prices on the stock ticker.**

   Change in ZBOR:

Got symbol: 'ZBOR'

Adding new figure: '99'

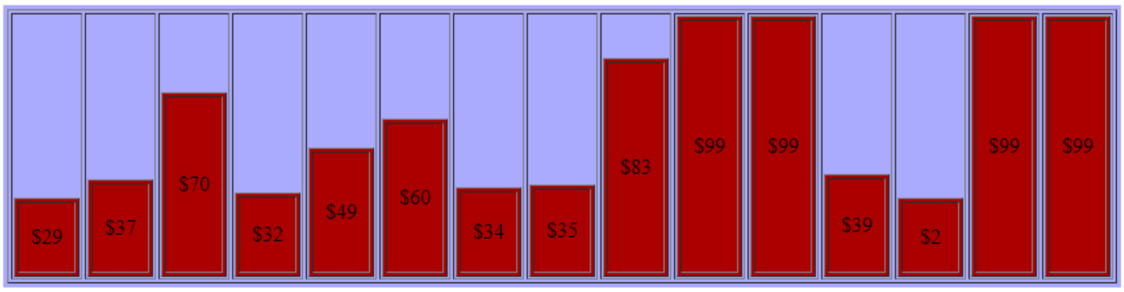
Data accepted.

Reload

Zembor Corp (ZBOR)

Don't Mess with ZEMBOR

Current price: $99



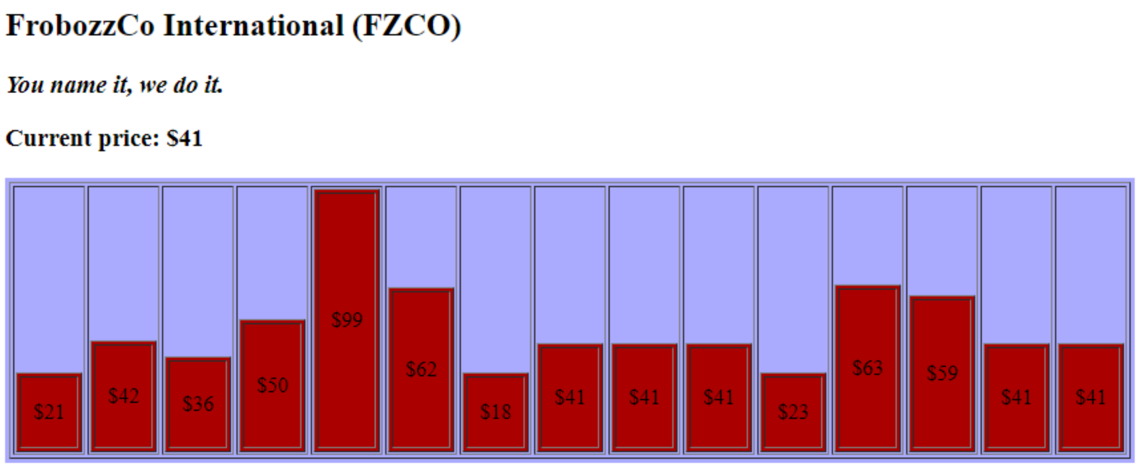
Change in FZCO:

Got symbol: 'FZCO'

Adding new figure: '41'

Data accepted.

Reload



FrobozzCo International (FZCO)

You name it, we do it.

Current price: $41

INSERTION ATTACK

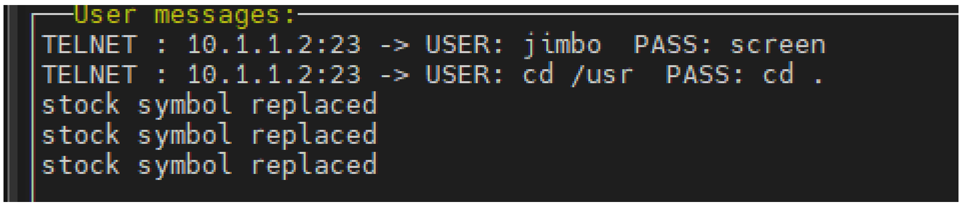
1. **You can change the symbols that a viewer of the ticker sees by intercepting the HTML bound for their browser. Write a filter to change the symbol FZCO to OWND.**

The code below was saved in a file called etter.filter.

```
#if the data is TCP or IP traffic destinated to HTTP 80

if (ip.proto == TCP ){

#search for FZCO to replace its data

 if (search(DATA.data, "data")) {

#perform find and replace string to OWND

 replace("FZCO", "OWND");

 msg("stock symbol replaced");

 }

}
```

The above filter was compiled to etterfilter etter.filter -o filter.ef

After including the filter in Ettercap, the output below was observed verifying that the code ran successfully.



```
User messages:
TELNET : 10.1.1.2:23 -> USER: jimbo  PASS: screen
TELNET : 10.1.1.2:23 -> USER: cd /usr  PASS: cd .
stock symbol replaced
stock symbol replaced
stock symbol replaced
```

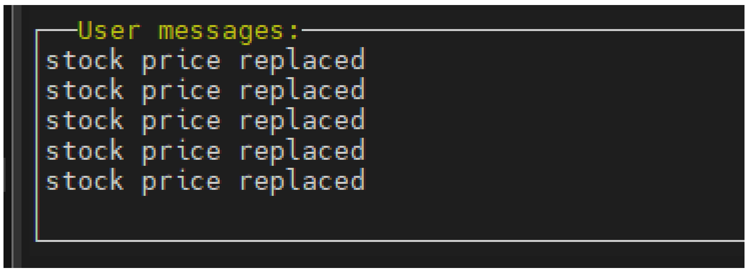
2. **Write a filter to affect the *prices* a user of the stock ticker sees.**

The code below was saved in a file called etter.filterone

```
#if the data is TCP or IP traffic destinated to HTTP 80

if (ip.proto == TCP) {

#search for $ to replace its data by $20

 if (search(DATA.data, "$")) {

#perform find and replace string to $20

 replace("$", "$20");

 msg("stock price replaced");

 }

}
```

The above filter was compiled to etterfilter etter.filterone -o filterone.ef

After including the filter in Ettercap, the output below was observed verifying that the code ran successfully.



```
User messages:
stock price replaced
stock price replaced
stock price replaced
stock price replaced
stock price replaced
```

3. **Given the power of etterfilter and the kinds of traffic on this network, one can actually make significant changes to a machine or machines that one is not even logged in to. How?**

By downgrading the SSH connection to a weaker version makes the connection vulnerable to insertion attacks.

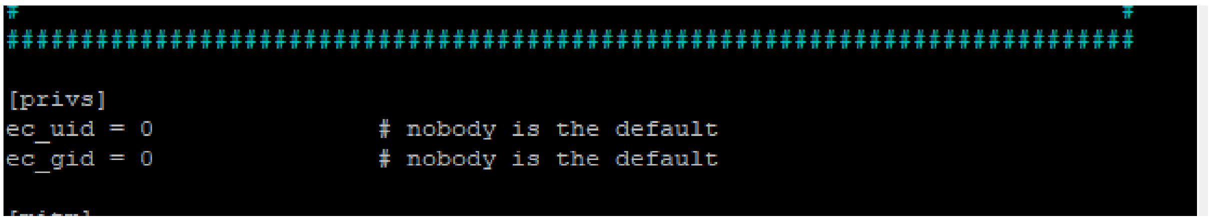
4. **Of the cleartext protocols in use, can you perform any other dirty tricks using insertion attacks? The more nasty and clever they are, the better.**

The communication between Bob and Alice involved Telnet and HTTP. We could drop the packets between these communications.

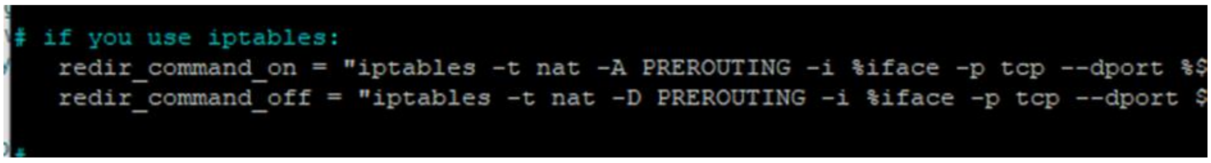
<u>MITM vs. ENCRYPTION</u>

1. **What configuration elements did you have to change?**

The following configuration elements were changed:



```
#                                                                              #
################################################################################

[privs]
ec_uid = 0                # nobody is the default
ec_gid = 0                # nobody is the default
```



```
# if you use iptables:
   redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %$
   redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport $
```

2. **Copy and paste some of this data into a text file and include it in your submission materials.**

The following data as in the image is copied and presented above.

3. **Why doesn't it work to use tcpdump to capture this "decrypted" data?**

. The ability to decrypt packets is only present if *tcpdump* was compiled with cryptography enabled

4. **For this exploit to work, it is necessary for users to blindly "click OK" without investigating the certificate issues. Why is this necessary?**

In HTTPs protocol the certificate needs to be accepted by the client who wants to establish the connection. Ettercap has a functionality to create an imitating certificate that is identical to server original certificate.

When a user clicks OK, the imitating certificate is accepted and a connection is established between the client and the man in the middle. This makes sniffing easier.

5. **What is the encrypted data they're hiding?**

sslstrip command was used instead of tcpdump and chaosreader.

Sslstrip provided an output log file which I analysed. The encrypted data was the stock.cgi scripts used for stock information modification. The various stocks and their prices.