Cyber Security Coursework - Protocol Analysis

James Donohue - james.donohue@bbc.co.uk

Introduction

This is my report for the 'protocol analysis' part of the Cyber Security coursework assignment.

The third-party tools used in this report were:

- `file(1)` - command-line tool common on UNIX-based systems for identifying file types
- Wireshark v2.2.4 - graphical network protocol analyser

`capture1.pcap`

The `file` tool identifies this capture as follows:

`tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)`

From this we can learn that the capture was created on a little-endian processor (such as the Intel x86 family). The 'capture length' indicates that captured data was limited to 65535 bytes per packet (Lamping, Sharpe, and Warnicke 2014).

Opening the file in Wireshark shows it contains 44 packets. Using the *Statistics* > *Endpoints* report we can see the packets relate to 6 different Ethernet addresses and 5 IPv4 addresses.

Find the username & password from the HTTP Basic type of authentication

By using the `http` view filter in Wireshark to show only HTTP packets (see Figure 1) we can see two separate HTTP connections are made from host `192.168.0.3` to port 80 on host `192.168.0.1`. The first of these connections contains a single request (`GET /`) which results in a `401 Access Denied` response.

The second connection (coloured in purple on Figure 1) contains two requests made using HTTP Basic authentication. The content of these requests can be seen most easily using the *Follow > TCP Stream' feature (see Figure 2).

Basic HTTP authentication, originally specified in HTTP/1.0, allows for simple challenge-response authentication of web clients using the `Authorization` header. The value after the scheme name `Basic` is a Base64 encoding of the username and password, separated by a colon. As no encryption is performed on the password, the Basic scheme is explicitly non-secure (Berners-Lee, Fielding, and Frystyk 1996). In this case the client sends the following header:

`Authorization: Basic YWRtaW5pc3RyYXRvcjpwQHNzdzByZA==`

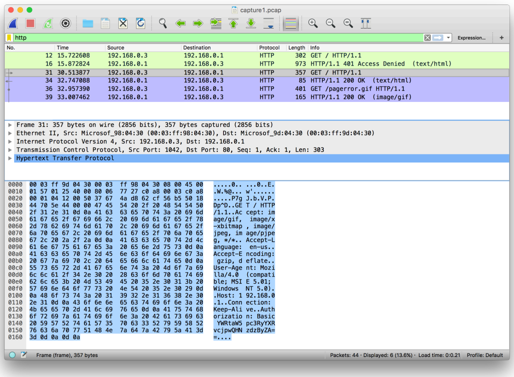

Figure 1: Wireshark filtering for only HTTP packets in capture1.pcap

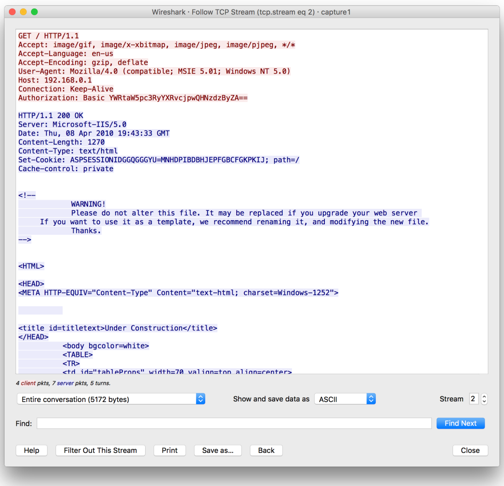

Figure 2: Wireshark Follow TCP Stream showing HTTP Basic authentication in capture1.pcap

The server responds with `HTTP/1.1 200 OK`, indicating that the authentication attempt was successful. The plaintext of the Base64 value can easily be determined using a trivial fragment of Python:

```
>>> import base64
>>> encoded = 'YWRtaW5pc3RyYXRvcjpwQHNzdzByZA=='
>>> base64.b64decode(encoded)
'administrator:p@ssw0rd'
```

Therefore the username sent was `administrator` and the password was `p@ssw0rd`.

Which hosts appear to be sending broadcast IP packets?

In IPv4, broadcast packets are those sent to a special address which causes them to be received by all hosts on a given subnet. The special address has all its unmasked bits set to 1 (Mogul 1984), which means ending in a sequence of one or more 255s when printed in typical 'dotted quad' form.

The Wireshark *Statistics > Conversations* view (shown in Figure 3) shows that four hosts are sending packets to `192.168.0.255`, which is the broadcast address for the `192.168.0.0/24` network.

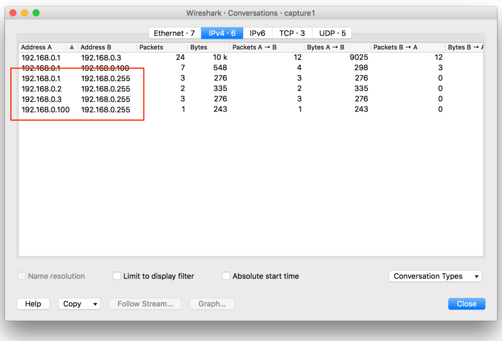

Figure 3: Wireshark 'Conversations' view for capture1.pcap with broadcast traffic highlighted

The hosts that appear to be sending broadcast IP packets (via UDP) are therefore:

```
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.100
```

The packets are UDP datagrams on ports 137 and 138 (NetBIOS), meaning we can infer that these hosts are likely to be running Windows.

`capture2.pcap`

The Address Resolution Protocol (ARP) (Plummer 1982) is used to determine the hardware (MAC) address of a target host based based on its IP address. In normal operation, the host attempting to resolve a MAC address sends a broadcast 'request' message containing the IP address that is being queried. This will be received by all network interface cards (NICs) on the same subnet. The target host then sends a ‘reply’ message containing its own MAC address to the host that made the request, which caches it for later reuse.

Because ARP has no built-in support for authenticating such messages, it is possible for a malicious user to use it to disrupt network activity. In ‘ARP poisoning’, an attacker sends fake ARP requests or replies with the intention of either intercepting network traffic between two hosts (man-in-the-middle attack) or producing a denial-of-service (DoS) (Mishra 2016).

Method of detection

ARP poisoning can be suspected when a host begins sending ARP packets containing a ‘sender’ hardware (MAC) address that is different to a MAC address already associated with the given protocol address (IP) in earlier traffic.

Therefore, a simple implementation of an ARP poisoning sensor is to record the sender MAC address associated with the IP address in each ARP packet. When a new ARP packet is received, if a different sender MAC is given for the IP in address in question than one previously recorded, an alert can be issued and any other required action taken.

In order to demonstrate this approach using the supplied packet capture file, a Scapy script (`poisonarp.py`) was written. This script implements the above approach by iterating over each packet in the capture and storing a mapping of IP address to MAC address in a dictionary object.

The script can be executed as follows:

```
$ python poisonarp.py cwk_pcaps/capture2.pcap
```

Example output

```
Read 328 packets from cwk_pcaps/capture2.pcap
WARNING: suspected ARP poisoning attack at packet 298
  Attacker: 00:03:ff:98:98:02 (last seen using 192.168.0.2)
  Victim: 00:00:00:00:00:00 (192.168.0.1)
  Hijacked resource: 00:03:ff:98:98:03 (192.168.0.3)
```

Limitations

To identify the victim, this script uses the ‘target’ hardware and protocol addresses given in the ARP packet, which is sufficient for the packet capture given. However, ARP poisioning can also be performed by broadcasting malicious ARP replies (so-called Gratuitous ARP or GARP). In this scenario the intended victim may not be immediately apparent.

Also, there are some scenarios in which the MAC address for an IP address may change legitimately (for example due to replacement of a part). These events will produced ‘false positives’ in a sensor that uses this approach.

Alternatives: Wireshark

Wireshark also identifies this type of attack using the message 'Duplicate IP address detected' (see yellow text in Figure 4).

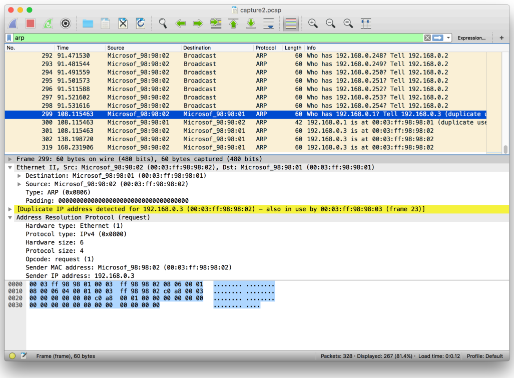

Figure 4: Wireshark showing duplicate IP address usage in capture2.pcap

`capture3.pcap`

The `file` tool identifies this capture as follows:

```
capture3.pcap: tcpdump capture file (little-endian) - version 2.4 (802.11, capture length 65535)
```

This reveals that unlike the other capture files (which were captured on an Ethernet network) this capture was created using a wireless (802.11) network.

Wireshark reports that the capture file contains 25895 packets. Using the display filter `wlan.fc.protected == 1` reveals that 59.5% of the packets (100% of data frames) are flagged as `protected` (see Figure 5). This bit indicates that the packet data is encrypted using an 802.11 encryption mechanism such as WEP (Wired Equivalent Privacy) or TKIP (Orebaugh, Ramirez, and Beale 2006).

This means that it will not be possible to simply use Wireshark's protocol dissection functionality to extract passwords from the capture, as the relevant data is encrypted. However, the display filter `wlan.wep.iv`, which identifies frames that include the WEP Initialization Vector (IV), shows the same list of packets, meaning that all of these packets are encrypted using WEP.

The stream cipher used by WEP has been shown to be insecure and vulnerable to passive attack by an eavesdropper (Fluhrer, Mantin, and Shamir 2001), given access to sufficient IVs. The tool aircrack-ng can be used to crack WEP encryption using a packet capture file as follows:

```
$ aircrack-ng cwk_pcaps/capture3.pcap
```

This successfully extracts the WEP key `6B:69:6E:67:73` ("kings") from the capture file as shown in figure 6.

If we input this key in hexadecimal form into Wireshark (using *Preferences > Protocols > IEEE 802.11 > Decryption Keys*) it will automatically decrypt the WEP-encrypted packets. Using the `http.request` display filter we can see that the HTTP requests being sent include the header (Figure 7):

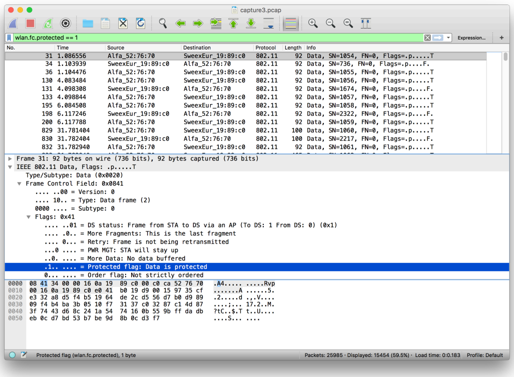

Figure 5: Wireshark showing encrypted Wifi packets in capture3.pcap

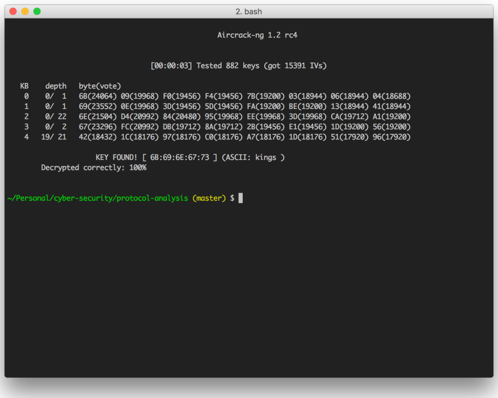

Figure 6: Output from aircrack-ng after cracking WEP on capture3.pcap

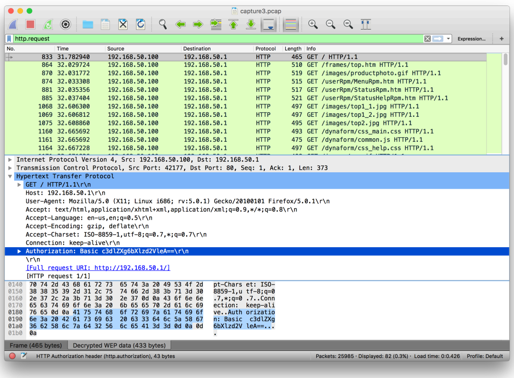

Figure 7: Wireshark showing decrypted HTTP traffic from capture3.pcap

```
Authorization: Basic c3dlZXg6bXlzd2VleA==
```

Using the same Python fragment used for `capture1.pcap` above we can therefore discover that the HTTP Basic username was 'sweex' and the password was 'mysweex'.

References

Berners-Lee, T., R. Fielding, and H. Frystyk. 1996. "RFC1945: Hypertext Transfer Protocol – HTTP/1.0." https://tools.ietf.org/search/rfc1945.

Fluhrer, Scott, Itsik Mantin, and Adi Shamir. 2001. "Weaknesses in the Key Scheduling Algorithm of RC4." *Selected Areas of Cryptography* 2259. Springer: 1–24.

Lamping, Ulf, Richard Sharpe, and Ed Warnicke. 2014. "Wireshark User's Guide: Capturing with Tcpdump for Viewing with Wireshark." https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html.

Mishra, Charit. 2016. *Mastering Wireshark*. Birmingham, UK: Packt Publishing.

Mogul, Jeffrey. 1984. "RFC919: Broadcasting Internet Datagrams." https://tools.ietf.org/html/rfc919.

Orebaugh, Angela, Gilbert Ramirez, and Jay Beale. 2006. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Rockland, MA: Syngress.

Plummer, David C. 1982. "RFC829: An Ethernet Address Resolution Protocol." https://tools.ietf.org/html/rfc826.