Man-in-the-middle attack to the HTTPS protocol

Article *in* IEEE Security and Privacy Magazine · March 2009

DOI: 10.1109/MSP.2009.12 · Source: IEEE Xplore

Franco Callegati
University of Bologna

Walter Cerroni
University of Bologna

Marco Ramilli
Yoroi

Man-in-the-Middle Attack to the HTTPS Protocol

Franco Callegati, Walter Cerroni, and Marco Ramilli
University of Bologna

Web-based applications rely on the HTTPS protocol[1] to guarantee privacy and security in transactions ranging from home banking, e-commerce, and e-procurement to those that deal with sensitive data such as career and identity information. Users trust this protocol to prevent unauthorized viewing of their personal, financial, and confidential information over the Web.

Netscape Communications introduced the Secure Socket Layer (SSL) for security-sensitive communication in 1994. The Internet Engineering Task Force (IETF) adopted it in 1999 as a standard known as Transport Layer Security (TLS)[2] to secure HTTP into HTTPS. An HTTPS URL indicates that the browser will download a Web page using HTTP but with a different default port (443) and an additional TLS encryption/authentication layer between HTTP and TCP. Consequently, most people consider HTTPS-based data exchanges safe, and the average user tends to trust the Web application as soon as the "lock" symbol appears. In this article, we show the danger of that assumption by demonstrating how attackers can successfully intercept the data transfer and corrupt the safety of the communication.

Attack Concepts

The *man-in-the-middle* (MITM) attack exploits the fact that the HTTPS server sends a certificate with its public key to the Web browser. If this certificate isn't trustworthy, the entire communication path is vulnerable. Such an attack replaces the original certificate authenticating the HTTPS server with a modified certificate. The attack is successful if the user neglects to double-check the certificate when the browser sends a warning notification. This occurs all too often—especially among users who frequently encounter self-signed certificates when accessing intranet sites.

This article assumes the scenario presented in Figure 1, in which a user on the *client host* (CH) wants to make a secure transaction on the *server host* (SH) using HTTPS. Given that CH and SH have to network exchange data, the *attacker host* (ATH) acts as a gateway for the traffic stream. The attacker (that is, the "man in the middle") intercepts traffic from the source and forwards it to the destination, thus gaining the ability to modify messages and insert new ones without either party realizing it.[3]

The attacker builds the attack in the following steps:

- Act as a gateway (MITM) between the CH and the LAN default router.
- Forward CH requests to connect to the SH to the default gateway without any interference.
- Intercept SH replies forwarded by the LAN default gateway.
- Create a false self-signed certificate to replace the original.
- Send the false certificate to the CH.
- When the CH accepts the certificate, build an encrypted channel between the CH and the attacker and another between the SH and the attacker.

At the end of these phases, the CH and the SH see an apparently secure communication channel between them. In reality, the attacker has the ability to decrypt the entire communication because he or she possesses the necessary keys.

Different variants of this attack exist, depending on the network configuration. The ATH can exist on the same network as either the CH or the SH, or generally reside anywhere on the Internet. This article assumes that the CH is connected to a switched Ethernet—the most common LAN technology in companies, small offices, and home networking (SOHO)—that's connected to the Internet via a router that acts as the default gateway and that the attacker is logged into a host (ATH) on the same LAN as the CH. This might happen either because the LAN is open and a malicious user can freely connect to it or because a host on the LAN has been broken and unauthorized users can log in.

The MITM attack is realized by maliciously modifying the Address Resolution Protocol (ARP) and the Domain Name System (DNS) protocol's normal behavior. In particular, *ARP poisoning* exploits techniques to sniff and retrieve traffic sent on a switched LAN using direct IP forwarding. Hosts rely on the MAC address to deliver IP datagrams, using the ARP protocol to find the matching IP and MAC addresses.[4] Given that ARP requests are broadcast, the attacker can read them and learn the MAC addresses of other LAN hosts. The attacker then implements ARP spoofing by impersonating MAC addresses of other hosts. By corrupting the matching between IP and MAC addresses, the attacker receives traffic from or to a given IP address.

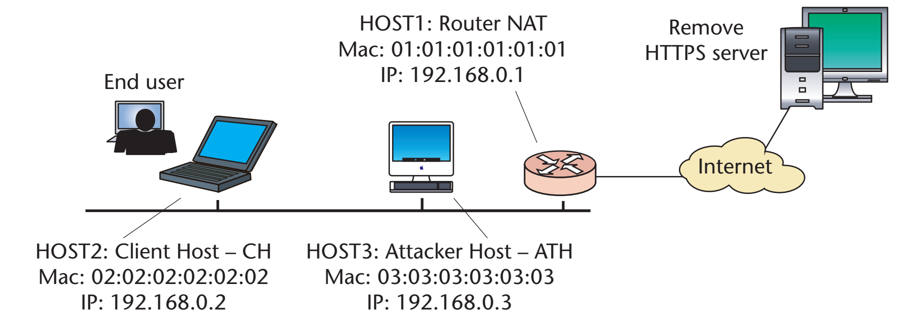

Figure 1. Typical network configuration. The user on the client host (CH) wants to make a secure transaction but is vulnerable to the man-in-the-middle attack. An attacker host (ATH) is connected to the same LAN as the CH either because a malicious user can freely connect to it or because a host on the LAN has been broken and unauthorized users can log in. The ATH will intercept and manipulate HTTPS traffic between the CH and server host.

```
[user@HOST1] $ arp -a
host2.victim.org (192.168.0.2) at 02:02:02:02:02:02 [ether]
   on eth0

[user@HOST2] $ arp -a
host1.victim.org (192.168.0.1) at 01:01:01:01:01:01 [ether]
   on eth0
```

(a)

```
[user@HOST1] $ arp -a
host2.victim.org (192.168.0.2) at 03:03:03:03:03:03 [ether]
   on eth0

[user@HOST2] $ arp -a
host1.victim.org (192.168.0.1) at 03:03:03:03:03:03 [ether]
   on eth0
```

(b)

Figure 2. ARP poisoning. (a) Before the attack, the MAC addresses associated with the IP addresses are the real ones. (b) After the attack, the router believes the MAC address of the CH is that of the ATH, and the CH believes the MAC address of the router is that of the ATH.

A Practical Example

Suppose a student at the University of Bologna wants to retrieve data from a secure Web server using one of the university labs' hosts. We call the imaginary server in this example https://example.unibo.it. The user has to log into the HTTPS server in this scenario to view his university career records.

Going back to Figure 1, let's assume that the hosts connected to the LAN use private IP addresses—for instance, from the network 192.168.0.0/24—and the router implements network address translation (NAT). This represents a rather typical configuration today. Figure 1 shows the IP and MAC addresses of the hosts involved.

The attacker starts by ARP poisoning the LAN. He then discovers the HOST1 and HOST2 IP and ARP addresses from ARP requests during normal LAN operation. During the attack, he periodically sends ARP replies to HOST1 and HOST2 impersonating HOST2 and HOST1, respectively, by claiming that the IP addresses 192.168.0.1 and 192.168.0.2 are at the MAC address 03:03:03:03:03:03. Before the attack, the ARP cache of the attacked hosts will appear similar to Figure 2a; afterward, it will resemble Figure 2b. This action aims to redirect all the traffic between the CH (192.168.0.2) and the router (192.168.0.1) to the ATH (192.168.0.3).

Now the attacker must execute a DNS spoof[5] to redirect all DNS requests to the attacker's machine. The goal is to have all traffic from the CH to the Web server pass through the ATH. The attacker uses a tool called *dnsspoof* to make replies to DNS requests issued by the CH point to other IP addresses than they're supposed to point to. In our example when the CH sends a DNS request asking for the IP address of example.unibo.it the request goes to the ATH because of the ARP poisoning. The ATH implements DNS poisoning and sends a DNS reply to the CH that example.unibo.it, translates into its own IP address—that is, 192.168.0.3. At this point, all traffic between HOST2 and HOST1 as well as DNS requests by HOST2 can be intercepted and modified by the ATH.

To make the CH believe ev-

```
root@5[knoppix]# webmitm -d
Generating RSA private key, 1024 bit long modules
. . . . . . . . . . + + + + + + +
. . . . . . . . . . . . . . + + + + + + +
E is 65537 (0x10001)
You are about to be asked to enter information that will be
incorporated
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]: Italy
Locality Name (eg, city) []: Cesena
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
University of Bologna
Organizational Unit Name (eg, YOUR name) []: Marco Ramilli
Email Address []:marco.ramilli@unibo.it
Please enter the following 'extra' attributes
To be sent with your certificate request
A challenge password []: IEEESecurityPrivacy
An optional company name []:
Signature ok
Subject=/C=IT/ST=Italy/L=Cesena/0=University of Bologna/
OU=Security/CN=Marco
Ramilli/emailAddress=marco.ramilli@unibo.it
Getting Private key
Webmitm: certificate generated
```

Figure 3. Creation of a fake HTTPS session. Webmitm generates a new certificate at the attacker host (ATH) that will be sent to the client host (CH) as if it was issued by the Web server (SH). If the CH accepts the fake certificate, the ATH will be able to intercept and decrypt all traffic because it (not the SH) holds the private key matching the public key in the certificate.

```
root@6[knoppix]# ssldump -r temp -k webmitm.crt -d > out
root@6[knoppix]# ls
Desktop out temp tmp webmitm.crt
```

Figure 4. Use of ssldump. The captured traffic is in file `temp` and the decrypted text will be stored in file `out`.

erything is normal, the ATH must forward the traffic between the CH and SH by enabling IP forwarding at the ATH via a simple modification to the IP routing tables. For instance, the attacker can enable IP forwarding using *fragrouter* (http://packages.qa.debian.org/f/fragrouter.html).

Once the attacker has intercepted the traffic between the CH and the SH, the ATH can start a fake HTTPS session in which the ATH can sniff and decrypt its traffic. This requires generating a fake certificate in the name of the SH at the ATH with *webmitm*. Running webmitm for the first time requires the attacker to answer several questions (see Figure 3). In practice, the attacker should provide answers related to the SH because the browser on the CH might display it to the user receiving the certificate.

If the CH accepts the fake certificate when it connects to https://example.unibo.it, the ATH will intercept and decrypt all traffic because it (not the SH) holds the private key matching the public key in the certificate. To sniff network traffic, we used *Wireshark* (www.wireshark.org), a popular software protocol analyzer available on most platforms.

When the CH connects to the HTTPS site, it has to accept the new certification. Most browsers alert the user when they receive a certificate not signed by a trusted authority. Issuing "self-signed" HTTPS certificates is common; the end user is usually familiar with this alert and quickly accepts the certificate. The attack works by exploiting this bad habit—indeed, an experienced user who reads the certificate in detail would have a good chance of detecting something unusual. Nonetheless, we believe that a detailed check for certificates lacking official signatures isn't a common practice. Therefore, the ATH has a good chance of seeing the HTTPS session initiated under its control. In this case, the ATH captures the traffic flowing on the secure channel. When it decides to terminate the capture phase, the ATH saves the captured network traffic and decrypts the file using *ssldump* (www.rtfm.com/ssldump), as shown in Figure 4.

In the cleartext file, the attacker will likely find confidential information such as the username and password of someone connecting from HOST2. However, this is typically a long succession of characters, in which it's difficult to identify the bits of text containing the sought-after information. The attacker can solve this problem by checking the original Web page for the input syntax (see Figure 5).

Finally, a search in the decrypted file of the intercepted traffic using the homepage input syntax provides the sought username and password, as shown in Figure 6.

We've shown that it's possible to attack Web-based connections secured via HTTPS by exploiting some properties of common LANs as well as typical behaviors of inexperienced users. The attack's implementation isn't trivial, but it's certainly not difficult for an experienced user. Furthermore, the ease with which an attacker can spoof a certificate underlines the fact that sites should recognize the potential hazards of self-signed certificates. Users quickly become accustomed to browsers' warnings; however, they ignore these at their peril! Although strong encryption is a powerful tool in providing data protection, the security it supplies is only as good as the weakest link in the chain. □

```
&ndsp;<input type="text" name="userid" maxlength="64" size="14"
value="">
</span>
<td align="left" class="unique">
<span class="uniqueError">
&nbsp;
</span>
</td>
</tr>
<tr class="unique">
<td align="right" class="unique">
<span class="uniqueLabel">
&nbsp;Password
</span>
</td>
<td align="left" class="unique">
<span class="uniqueinput">
&nbsp;<input type="password" name="password" maxlength="10"
size="14" value="">
</span>
.. .. ..
```

Figure 5. Input syntax check. Searching for the string "input type" in the HTTPS server's homepage provides information about the name of the fields containing usernames and passwords.

```
Connection: Keep-Alive
Cache-Contrlo: no-cache
Cookie: JSESSIONID=62ED8B17175165BE92D43F2E61DB0A10
Family=studente&
userid=marco.ramilli&password=IEEESP----------------------------
20 20 19.5022 (12.4627) C>S application_data
21 21 19.5022 (0.00000) C>S application_data
22 22 21.4693 (1.9671 ) S>C application_data
19 17 21.4973 (1.9794 ) S>C application_data
----------------------------------------------------------------
```

Figure 6. Username and password retrieval. Searching the `out` file for the string "password" provides the sought `userid=marco.ramilli` and `password=IEEESP`.

Acknowledgments

We thank Matthew Bishop at the University of California, Davis, for his suggestions on the draft of this manuscript.

References

1. E. Rescorla, *HTTP Over TLS*, IETF RFC 2818, 2000; www.ietf.org/rfc/rfc2818.txt.
2. T. Dierks and C. Allen, *The TLS Protocol*, IETF RFC 2246, 1999; www.ietf.org/rfc/rfc2246.txt.
3. H. Xia and J.C. Brustoloni, "Hardening Web Browsers against Man-in-the-Middle and Eavesdropping Attacks," *Proc. 14th Int'l Conf. World Wide Web* (IW3C2), ACM Press, 2005, pp. 489–498.
4. D.C. Plummer, *An Ethernet Address Resolution Protocol*, IETF RFC 826, 1982; www.ietf.org/rfc/rfc826.txt.
5. US Federal Bureau of Investigation Nat'l Press Office, "Web 'Spoofing' Scams Are a Growing Problem," press release, 21 July 2003; www.fbi.gov/pressrel/pressrel03/spoofing072103.htm.

***Franco Callegati** is an associate professor of communication networks at the University of Bologna, Italy. His research interests include teletraffic modeling and performance evaluation of telecommunication networks, optical packet switching, and optical networking. Callegati has a PhD in electronic and computer engineering from the University of Bologna. He is a member of the IEEE. Contact him at franco.callegati@unibo.it.*

***Walter Cerroni** is an assistant professor of communication networks at the University of Bologna, Italy. His research interests include performance evaluation of optical packet/burst-switched networks, focusing in particular on contention resolution mechanisms, and signaling protocols for application-aware services over dynamic optical networks. Cerroni has a PhD in electronic and computer engineering from the University of Bologna. He is a member of the IEEE. Contact him at walter.cerroni@unibo.it.*

***Marco Ramilli** is a PhD student of electronic, computer, and telecommunications engineering at the University of Bologna, Italy. His research interests include security and penetration testing of distributed systems, and electronic voting systems security. Ramilli has an MS in computer engineering from the University of Bologna. He is a member of the IEEE. Contact him at marco.ramilli@unibo.it.*