Man-in-the-Middle TLS Protocol Downgrade Attack

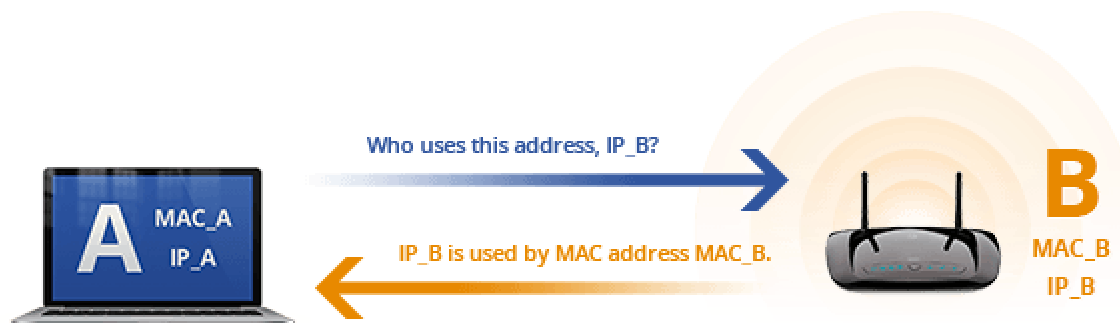
By Justin Copeland

Published: 2014-08-19 · Archived: 2026-04-06 03:09:38 UTC

A flaw was recently found in OpenSSL that allowed for an attacker to negotiate a lower version of TLS between the client and server (CVE-2014-3511). While this vulnerability was quickly patched, an attacker that has control of your traffic can still simulate this attack today. Let's explore how this is possible through looking at man-in-the-middle attacks and how browsers handle SSL/TLS connections. In addition, we will see the implications of the attack on cryptographic security.

In order to man-in-the-middle a connection between two devices on a local network, we need to convince the client and the local gateway, such as a router, to redirect traffic through the attacker. This is done by manipulating the Address Resolution Protocol (ARP) cache. ARP is used to pair physical addresses (MAC addresses) with IP addresses. When a device wishes to send data to a particular IP address, it first queries its ARP cache to find the MAC address of the receiving device. If the MAC address is not in the cache, it sends an ARP request to the network. The intended recipient device receives the ARP request and sends an ARP reply containing its own MAC address, after which both devices now have updated ARP caches. The communication can be simplified as follows:

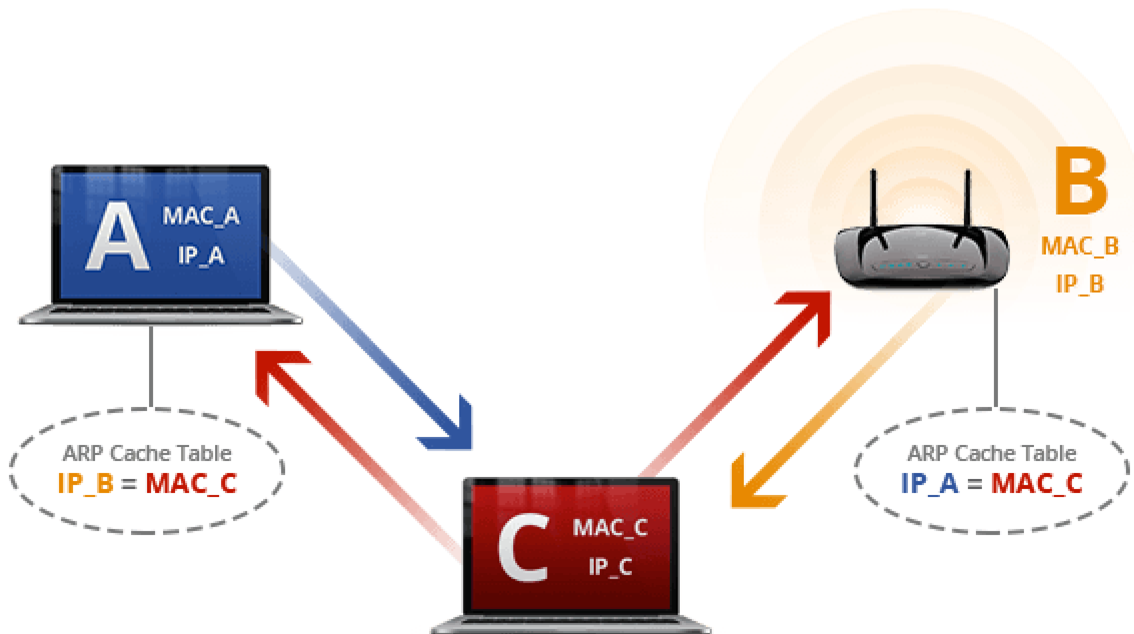
Updating Device ARP Cache Tables



1. **Device A** queries every device on the network, asking, "Who uses this address, `IP_B`?"
2. **Device B** replies back to **Device A**, saying, "`IP_B` is used by MAC address `MAC_B`."
3. **Device A** receives reply and updates its ARP cache to pair `IP_B` and `MAC_B`.
4. **Device A** now uses `MAC_B` to send data to Device B.

However, ARP cache updates are gullible. ARP caches will update given a reply even without a request. ARP also has no message authentication or integrity checking, meaning an attacker can spoof information in a given reply. In order to intercept communications, the attacker does the following with Devices A and B:

Manipulating Device ARP Cache Tables



1. **[Attacker C]** sends a reply to **Device A**, saying, " `IP_B` is used by MAC address `MAC_C` ." Whenever **Device A** wishes to communicate with **Device B**, it will send data through **[Attacker C]**.
2. **[Attacker C]** sends a reply to **Device B**, saying, " `IP_A` is used by MAC address `MAC_C` ." Whenever **Device B** wishes to communicate with **Device A**, it will send data through **[Attacker C]**.

Device C now receives all packets sent between Device A and B, such as the client and the router. Device C can enable IP forwarding to forward all traffic received onto the proper device. The attacker now can sniff, modify, or drop received packets. We can use this to abuse a feature of browsers to negotiate an older version of SSL/TLS.

When a client wants to send data to a server using SSL/TLS, a client must first go through a handshake to authenticate itself with the server. This handshake starts with the "ClientHello," where the client sends to the server a version of SSL or TLS that it supports, the supported ciphers, and other session data. In older versions of SSL (version 2), it was possible to intercept this handshake packet and modify the supported ciphers list to only contain weak ciphers. This is no longer possible since SSLv3 uses a hash in the final part of the handshake, where both the client and server hash and compare sent and received messages.

All modern browsers support SSLv3 up to TLSv1.2, but will use the highest version supported by a server. A middleman cannot directly modify any packets sent in the handshake, but a middleman can intercept and drop certain packets. By tricking the browser into thinking that the server does not support a given version of SSL/TLS, an attacker can downgrade the negotiated version. Let's see how this is done.

MITM TLS Protocol Downgrade Attack

Client sends " `ClientHello` " to server. The middleman intercepts and drops the packet.

```
            if re.search('x16x03x01.{2}x01',orig_load,flags=0): # ClientHellop.drop() # Drop cur
```

Middleman sends " `FIN` , `ACK` " over TCP to server. This terminates the current connection.

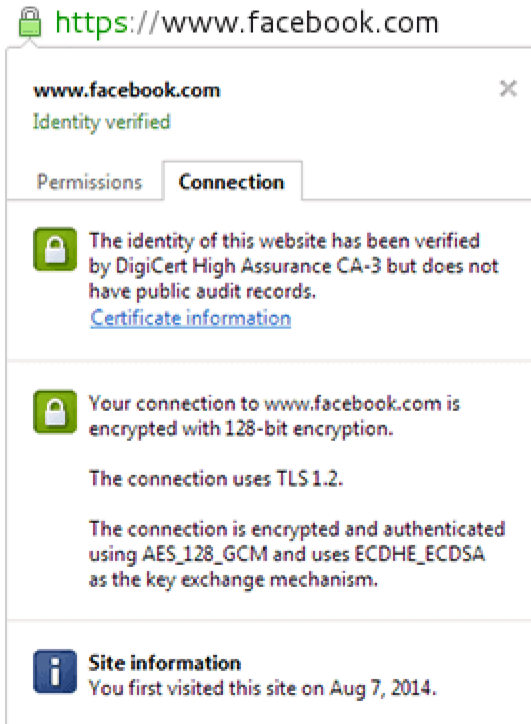
```
new_packet = IP(dst=pkt[IP].dst, src=pkt[IP].src)/TCP()new_packet[TCP].sport = pkt[T
```

Client re-attempts connection, sending “ `ClientHello` ” with lower version of SSL/TLS.

Protocol downgrade attacks rely on the assumption that an error or termination of the connection means the connection failed due to a SSL/TLS failure. Additionally, in order to be compatible with previous versions of SSL/TLS, a client may [attempt multiple connections] until a successful connection is made. Therefore, by repeating the protocol downgrade, a middleman can convince the client to negotiate SSLv3 with the server.

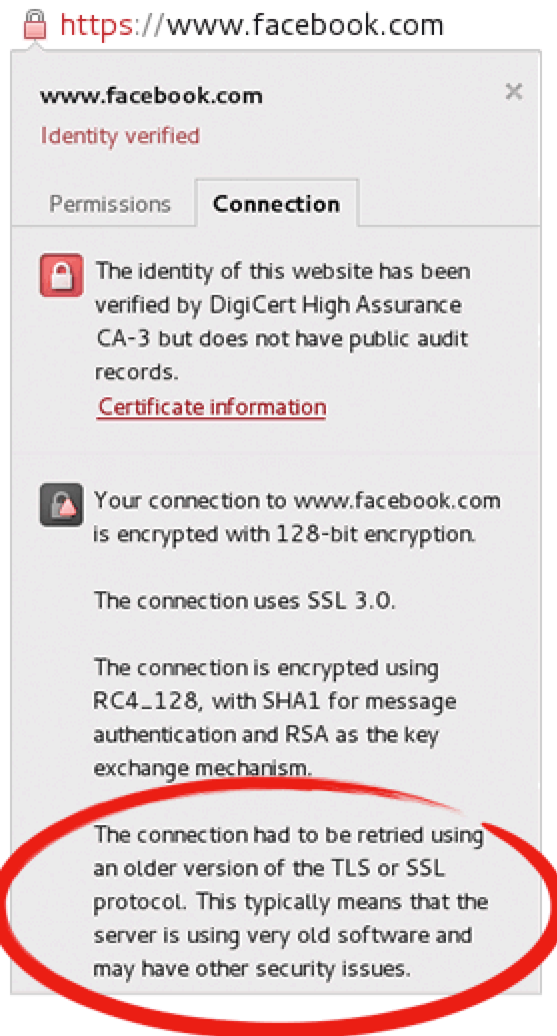
This protocol downgrade attack was tested while connecting to Facebook using the latest versions of Firefox, Chrome and Opera. Prior to the attack, the browsers connected to Facebook’s server using TLSv1.2. In the presence of an active downgrade attack, Firefox and Chrome both negotiated lower SSLv3 connections. Opera was not susceptible to the active attack.

TLSv1.2 vs SSLv3 in Browsers



Standard connection to Facebook.com using

TLS 1.2



Attack downgrades connection to use

SSL v3

Why Move Away From SSLv3 Now?

While SSLv3 included special mitigations to prevent protocol downgrade attacks, it is not necessarily the ideal protocol to use. SSLv3 has significant cryptographic differences, which could result in weaknesses that further demonstrate why TLSv1.2 should be the current standard. The agreed-upon encryption and authentication ciphers, as well as key exchange mechanisms differed significantly in our protocol downgrade tests. In the above example, TLSv1.2 uses elliptic curve cryptography (ECC) along with counter mode for AES, while SSLv3 uses the older RC4 cipher and RSA.

Some may ask why this is necessary. In his 2013 Black Hat talk, Alex Stamos discussed the current state and future of cryptography. He argued that one of the dangers lies in the potential to break older ciphers or key exchange mechanisms at some point in the future. In the case of RSA, cryptographers and mathematicians have made significant progress in the problem of factorization. Diffie-Hellman (DH) relies on the discrete logarithm problem for cryptographic security, and while no efficient algorithm used to compute discrete logs exists, the runtime of discrete logarithm algorithms has significantly decreased in the past year. As Stamos discussed, once RSA or DH fails, code-signing will break, and attacks on SSL/TLS will become very prevalent.

In summary, an active attack on a connection can result in lowered cryptographic security. Clients and servers can prevent this from happening by supporting only newer versions of TLS. Additionally, clients should respond properly to failed handshakes. Currently, many browsers opt for interoperability over security, which makes protocol downgrade attacks feasible. These changes will require significant time and effort. Browsers would need to reimplement aspects of how they handle handshakes. Backwards compatibility may break in some instances. However, eventually we will need to require use of newer versions of TLS that support ECC. Why not make the push now, and prevent future attacks?

Source: https://www.praetorian.com/blog/man-in-the-middle-tls-ssl-protocol-downgrade-attack/