

Topher's Spring Research Semi-Script

[Page 1]

{click}

[Page 2]

How often do you use a login form like this one, and give it your username and password? What I'll be talking about today is a simple attack that I have used in my lab to steal usernames and passwords that my co-workers have entered into this form, on their own computers. Fortunately, I told them what I was doing first, but if not for that, they would never have suspected that their machines were being attacked.

This attack is not novel, yet the vulnerability that it exploits still exists. My proof-of-concept program was developed in order to better understand how vulnerable systems are to the attack I use, and to later explore the possibilities of developing a remedy.

So, first, some background...

{click}

[Page 3]

Typically, when a user browses the web, data between the client and server is transmitted in the clear, and can be recorded, read, and even altered by any node in the path between the two machines.

{click}

[Page 4]

In order to protect sensitive data, SSL is used in to secure the connection. SSL provides server authentication, data confidentiality, and data integrity. Now passwords, web-based email, financial records, etc. can be transmitted in a secure channel across an insecure network.

So, how can you tell if you are securely connected to a web server? How can you tell if your data is being transmitted securely?

{click}

[Page 5]

Look at the address bar. In both Firefox and Internet Explorer, you will see that the scheme name has changed from "http" to "https". You will also notice a lock on the right-hand side of the address bar. Always look for the lock!

{click}

[Page 6]

But this login form is supposed to be secure, right? Why doesn't the login page have a lock on it? Is it transmitting my username and password in the clear?

{click}

[Page 7]

No, if we look at the HTML source for this page, we see that the contents of the form are sent to a page on a secure web server. When the form is submitted, first, a secure connection is made to the server, and then the username and password are sent over that secure channel.

So, everything works perfectly, right? Well, usually it does, but let's look at this slide again.

{click}

[Page 8]

When the client's browser requests the login page from the web server, the web server has to send it across this path made up of other machines. What if one of them was controlled by a malicious person? Anybody controlling one of these machines would just need to alter the login form as it travels to the client, and instruct the client's machine to send the username and password back to the attacker in plaintext. The user would never know that they had received an altered copy unless they read the HTML source of the form, and knew what the page originally looked like.

{click}

[Page 9]

Vulnerable login pages are actually quite common. As you can see, this list contains financial sites, and some very popular websites. {click} and BYU's home page.

{click}

[Page 10]

Fortunately, not many malicious individuals have control over critical machines like these, but that doesn't have to stop them. All somebody has to do is use a technique known as ARP-spoofing,

{click}

[Page 11]

and now he's part of the path, he's a man-in-the-middle. Now, you've probably heard of a "man-in-the-middle" attack before. This is any type of attack where the attacker has control of network traffic between 2 machines. And with ARP-spoofing, we can do that.

So, now that we see that many web login forms are vulnerable, how hard are they to exploit? Not hard at all. Well, let's talk about the method that is used.

{click}

[Page 12]

When a client wants to send data across the Internet, {click} it usually starts with the hostname of the server, but to send data across an IP network, the IP address of the server is needed. {click} DNS is used to resolve that hostname to an IP address. {click} but in order for data to get from the client, out to the Internet, it often has to travel across an Ethernet network.

{click}

[Page 13]

To travel across the local network, the client needs to know the MAC address of the next node in its

path. {click} Your computer's routing tables are examined {click} to determine which machine is the next node {click} in the path towards the server. But we still need to know the {click} MAC address of that machine in order to communicate with it. {click} ARP is used to resolve the MAC address of that machine. {click}

Now for a more detailed example of ARP in action

{click}

[Page 14]

To simplify this example, we'll use single digit numbers for IP addresses, and letters for MAC addresses. ARP uses two kinds of operations, requests and replies. If Host 1 needs to send something to Host 2, {click} it broadcasts an ARP request across the network saying, "Everybody: Host 1 (with MAC A) needs to know who Host 2 is." {click} Host 2 receives the request and then sends a reply directly to Host 1, {click} "MAC A: Host 2 has MAC B." Host 1 then updates its ARP table, mapping IP address 2 to MAC B. Now it can send data to Host 2. That's it. There's no means of authentication, just blind trust.

Now, knowing that ARP is a very trusting protocol, it's not too difficult to see how this trust can be abused. Machines will process ARP replies that they never requested. In other words, even though Host 1 already received a reply from Host 2, I could later use Host 3 to send an unsolicited ARP reply {click} to Host 1 {click} saying, "MAC A: Host 2 has MAC C", and Host 1 would {click} update its table accordingly, and send any traffic intended for Host 2, over to Host 3.

This is known as ARP-spoofing. Sending ARP replies in order to misinform another host, and cause it to enter an incorrect mapping into its ARP table.

{click}

[Page 17]

Now, let's talk about the actual attack that I used. First I used ARP-spoofing {click} to make the client think that my machine was the router {click}, then I did the same thing to the router {click}, telling the router that I was the client {click} that I was attacking. This set my machine up as a man-in-the-middle. Controlling all of the traffic between the client and the outside world.

{click}

[Page 18]

My program started to examine each packet that arrived for the target, or was sent from the target, only two special packets were being looked for, everything else was forwarded. {click} For instance, this request to Google would be examined {click}, then simply forwarded on, {click} and any traffic coming back from Google {click} would not {click} be interrupted in any way. The first packet that was being watched for was an HTTP request {click} for the BYU home page {click}. When this was detected {click}, the request was dropped. The program then sent an {click} altered copy of the home page to the target {click}. This copy {click} directed the client to send the user's {click} name and password over an {click} insecure connection to a non-existent page. When this packet was detected, it was read and saved {click}, and the client was then {click} redirected {click} to a legitimate login page.