CS 378 Final Project Report:
Automating ARP/DNS Spoofing

William Hua, Matthew Xavier, Tarik Ozkaya, Maximus
Nov/Dec 2016

I. What's The Tool?

Our tool provides a simple way to hijack a website and ARP spoof a victim's machine in order to route traffic to our malicious site and steal credentials. Our proof of concept is able to clone any site and pass all inputs to a remote website which stores the sensitive information in a database. Usage for the tool is as follows:

python3 dns-spoof.py <website-name> <network interface>

Examples of <website-name> include "www.google.com" and "www.pcworld.com". Examples of <network-interface> include "eth0" and "wlan0". The network interface names can be looked up by running "ifconfig" on the Unix terminal. The tool does require some dependencies, including nginx, ettercap, locate, wget, and sslstrip.

Additionally, the tool requires that the firewall be set up correctly, with ports 80 and 6666 open. This information can be viewed by running "python3 dns-spoof.py -h" for help information. If any of the dependencies are missing, the program will exit with an informative error message describing which dependency is missing.

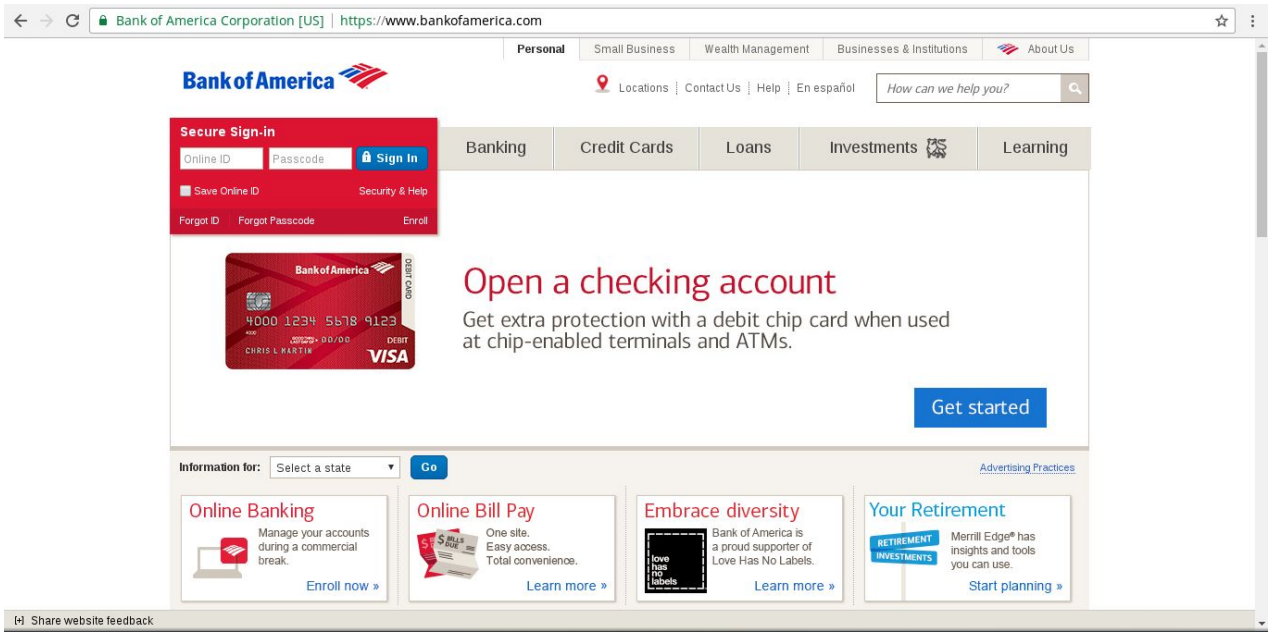
II. How It Works?

a. Cloning

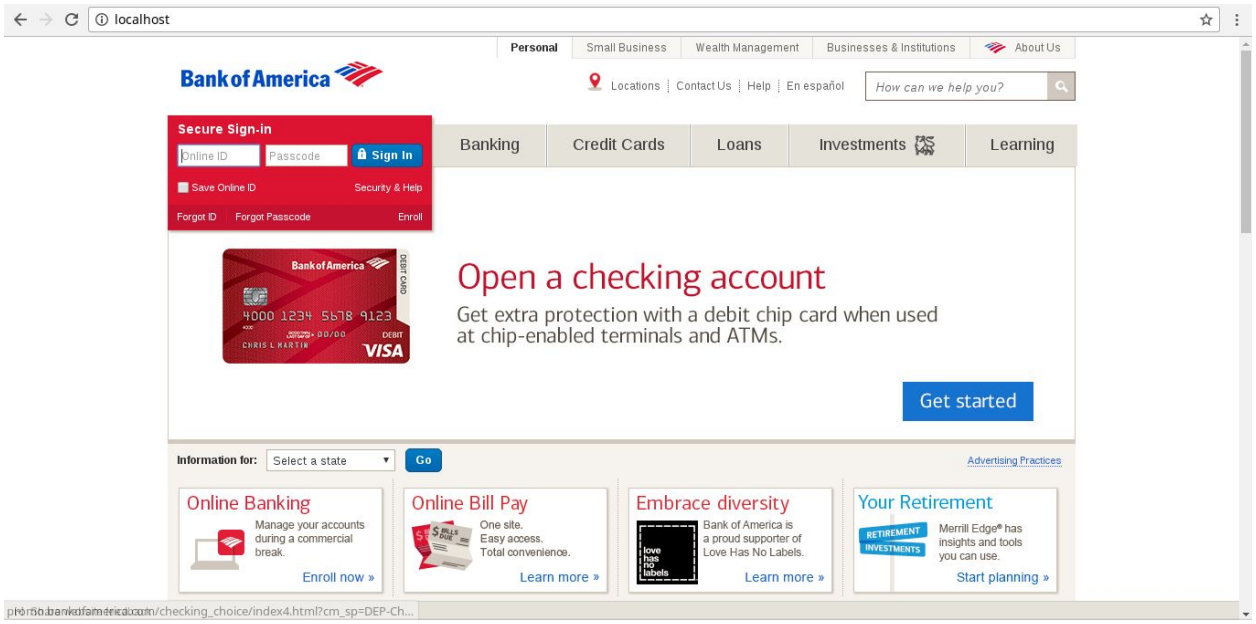
After the user passes the website we wish to clone, we use wget to clone the website. We use arguments on wget to configure for our use case, such as turning off robots.txt and adding the recursive flag to get directories within the website HTML structure. Initially, we were using httrack, but discovered that wget was able to get more sites.

We also modify the website downloaded for phishing purposes. We will elaborate more on this in the phishing section.

As an example below is a picture of the actual Bank of America website.



In contrast, below is a picture of the cloned Bank of America website that was produced by wget.



As shown, the website cloning is reliable in providing a look-alike website that can be hosted on the local server by the attacker.

b. Hosting

After cloning the website, we should have a directory that points to the website. We take the nginx configuration file and modify it so that the nginx server points to the newly cloned

website. This will essentially host a replica of the server on our local host. We need to restart nginx to ensure the updates appear.

First, the nginx configuration file is updated by the script so that the document root is pointing to the folder that will be created by wget, which contains the cloned website pages. Next, wget is run to generate a local copy of the website into a folder which is pointed to by the nginx.conf file. After the local website clone is created, the html files of the local copy are updated so that HTML forms that use a POST request send the request to the attacker's server. Finally, nginx is restarted to reflect the updates, and the ettercap and sslstrip tools are executed in order to conduct DNS/ARP spoofing.

c. ARP Spoofing

Next, the etter.dns ettercap file is updated so that ettercap will re-route requests for the website to the machine running the tool. Then, we will execute Ettercap and SSLStrip to conduct the ARP Spoofing. Ettercap is responsible for the ARP and DNS spoofing, and is run with the appropriate parameters. SSLStrip is used to downgrade a HTTPS connection to HTTP, so that ettercap can still manipulate the connection.

d. Phishing

After the user visits the phishing website, s/he enters the username and password. On the phishing website, we have modified all the form submissions to re-direct to our server and track all inputs. After submitting the form, the input is sent to our server. The script iterates through all the variables sent, and stores them in a database. Then, it redirects to a second page, which shows the previous stolen credentials.

III. What Problems Did We Encounter?

Our first idea did not meet all of the project requirements. As a result we had to brainstorm and think of a new tool to build. Additionally, the group ran into problems while setting up ARP spoofing to work on other machines on the same Local Area Network (LAN). One problem was initially using the python SimpleHTTPServer module to host the cloned website. Unfortunately, this was not sufficient to carry out the ARP spoofing attack, since the victim was never redirected to the python HTTP server. As a result, server technologies such as Apache and Nginx were looked at as viable alternatives. Ultimately, nginx was chosen as the server technology for the project since it was easier to configure and some group members had already worked with it.

Another problem associated with ARP spoofing was not realizing that the firewall was blocking connections to the machine hosting the cloned website. Fortunately, fixing the firewall issue was not as hard of a problem to fix. In regards to website cloning, the group found that some websites would not be cloned by httrack, the program used to save copies of websites to

the local machine. Even after disabling robots.txt adherence on httrack, the program still would not clone some sites such as www.facebook.com.

A major problem that limited the success of the program was the presence of DNS and browser caches. If the victim's machine had already visited a site that we were going to spoof, the victim would not be redirected to our site hosted on a local server. This is because the victim would use the already cached DNS information to route to the site, rather than actually making a DNS request. This was one the main problems that we had difficulty getting around, and ultimately our solution was to clear the browser and DNS caches of the victim's computer to allow the exploit to work.

IV. How Can We Improve It?

a. Ease of Use

Our current script is not incredibly easy to use. We have a few ideas for improvements:

1. Automated Installation:

Our tool requires 5 programs: wget, ettercap, sslstrip, nginx, and locate. We only display to the user if these modules are not installed, we do not actually install them. We could make it easier to install these modules; however, we have to make the instructions specific per OS.

2. Better Nginx Configuration:

We discovered that Nginx has different configuration based on the OS (or we assume since all our Nginx versions were the same). Specifically, Kali would have a different Nginx configuration than OS X and SUSE. We could make our configuration modifier script better so that it could correctly modify the configuration so that it works on every program.

3. Automated Port Opening:

Our program requires the ports 80 and 6666 to be open. We could automatically open up these ports for the program to run; however, these instructions are different per OS.

4. Choice of Network Interface/IP Addresses:

To create a better user experience, we could run the command ifconfig and present the results to the screen. We could then allow the user to pick from the listed interfaces. Additionally, we could list the IP addresses on the local area network and allow the attacker to choose one of the listed IP addresses to attack. This would make ettercap run more efficiently.

b. Speed Up Cloning

Using HTTrack and WGet will take too long to clone websites such as Google or Facebook because we attempt to clone the entire website. An improvement on this would be to limit the cloning to the first three levels of documents in the URL tree so we don't need to get every resource. One way to go about this is using directory depth as a function of '/' chars. In doing so we can say a URL of 'x/' is higher level than 'x/y/' which may be a depth two url. By limiting our cloning to these first three levels we can ensure faster replication times and also

foresee an extension in which we can re-link users back to the actual site, masking our credential theft.

c. Certificate Pinning

Many browsers have browser-side security measures in place to prevent spoofing websites. Some sites have a type of SSL Certificate that is synchronized to a browser. What this means is that if we spoof Google.com, and we go to Google.com (which re-directs to our local version), unless we have the correct SSL Certificate, the browser will warn the user that we are going to the wrong website. The browser is smart enough to reject the fake certificate and redirect to the actual, correct website. We saw this behavior with major sites such as Facebook.com. Finding a way to get around this problem, if possible, would allow our exploit to be very useful for getting credentials to sensitive sites, such as Facebook.com.

d. Sophisticated Attacks on Caching

As referenced above, one of our main problems was dealing with caches on the victim's computers. To combat this problem, our group would conduct additional research to see if we could get around caching. Based on some of the other groups' presentations, one possibility would be to add de-authentication to our program to kick users off the network. We could then test if this action would cause the victim's caches to be reset, and thus redirect them to our local server. Additionally, we could try to look into other methods of forcing victim computers to update their DNS cache with our poisoned DNS information.