Arsenic Mobile - A zero configuration ARP poisoning and credential extraction tool for iOS

Kernel Sanders

Abstract

Trust and malicious activity on local networks were not major considerations when the address resolution protocol (ARP) was defined in 1982, and were still unsatisfied in 1991 when HTTP was defined. As security issues came to light, secure sockets layer (SSL, defined in 1995) was developed as an attempt to address these issues. However, because of the specifics of both protocols coupled with unscrupulous users, there remains an opportunity for exploitation.

Tools have been developed that exploit the vulnerabilities inherent in the protocols, and this paper will focus on original research implementing one such tool (arsenic) in a user friendly manner on mobile devices. What used to take multiple tools and configurations on a full OS can now be accomplished with a single command on a cell phone. Arsenic-mobile can be used to demonstrate and educate users on the potential dangers of these protocols commonly thought of as secure.

1. Introduction

When the protocols that are now used as the backbone for networking were first developed, the notion of trust and security were not major concerns of their creators. The predominately government and research university makeup of this new network did not require any extra precautions, as all members of the network could be trusted. As this small network grew into the internet, security concerns grew more serious as the network began to host sensitive information and moderate transactions. To counter these threats, Netscape developed the secure sockets layer (SSL) encryption protocol in 1995 [1]. This protocol uses asymmetric (public key) cryptography to exchange symmetric encryption keys to encrypt session traffic. SSL was completely rewritten in 1996, and eventually replaced by transport layer security (TLS) [2]. TLS was based on SSL and uses the same cryptographic principles. Today TLS 1.0 is the most prevalent form of encryption used on the internet, despite a published attack [3] [4].

2. Possible Attack Vectors

The SSL/TLS protocol presents a limited number of areas where an attacker could potentially circumvent the security measures and perform a man in the middle attack. First, an attacker could determine the private key associated with a signed certificate. This would mean reversing a strong public key algorithm such as RSA, which has been shown is very computationally expensive. It has been shown that to factor a 768-bit number it would take a standard computer of today 15,000 years [5]. This attack method is clearly not practical for a man in the middle attack.

Second, an attacker could attempt to sign their own certificate to say that they are the site they are impersonating. However, this requires the same computational work as determining a private key and therefore is also impractical.

Finally, an attacker could break into a legitimate certificate authority and sign themselves certificates for popular websites. This would also be extremely difficult, but it has been done before. On the 15th of March, 2011 an Iranian hacker managed to sign himself nine certificates for seven popular domains [6]. However, most internet traffic is not encrypted with SSL/TLS, and therefore most start these secure sessions from regular HTTP sessions. These secure sessions are initiated through HTTP 302 redirects, or when the user clicks on a link to an HTTPS site. Because these requests are sent unencrypted, they could be seen and denied.

3. sslstrip and Arsenic

At Blackhat 2009, Moxie Marlinspike released a tool that monitors HTTP connections and when a user attempts to elevate to an HTTPS connection (by clicking on a link or by being redirected) this tool maintains the HTTP connection to the user, while establishing an HTTPS connection out to the server [7].

This attack can be used to effectively capture a users credentials without causing any warnings (such as an invalid certificate warning) to appear in the browser. In modern browsers the user experience between the same website on HTTP and HTTPS is minimal.

Leveraging this fact, tools have been created that incorporate the ARP cache poisoning and network scanning with *sslstrip* to provide a simple, compact way to implement this attack. One such tool is Arsenic [8]. These tools operate well on laptops, but their effectiveness and concealability could be improved if they are able to be run on mobile devices, which is the goal of this research.

4. Porting to iOS

Porting Arsenic to iOS presented some difficult and unique challenges. The first and most important issue to overcome was Apple's code signing requirement. However, due to the active and prevalent nature of the "jailbreak" community, this proved to be a trivial step (given a device on the correct software version, i.e. $<$ 6.1.3). The jailbreak community has also ported a few dependancies of Arsenic to iOS, including *Python 2.7.3*, and *arpspoof*. However, iOS is does not have *iptalbes* or *ipfw*, the two firewalls that Arsenic is able to use. These are important because *sslstrip* is essentially a shadow proxy and packets must be forwarded to it. The lack of a firewall on iOS has been noted by others, with a "Global Moderator" and "Hero Member" of *iNinjas.com* saying, "As of now [November 5, 2012] sslstrip does not work on the idevice... there are no iptables and there is no port forwarding because of kernel limitations." [9] However, the iOS 4.0 jailbreak was based on a vulnerability in the implementation of the BSD pf firewall [10]. Although the pf firewall is not fully implemented in iOS (probably a relic of the BSD basis of Darwin) it is able to forward packets according to rules, which is enough to get *sslstrip* to work. In order for *sslstrip* to install properly, its dependancies must be installed on iOS first. This presents a challenge because these dependancies, *Twisted-Web*, *zope-interface*, and *pyOpenSsl* have C source code that must be compiled on the device. Luckily, the jailbreak community has ported *gcc* to iOS. With a modification to the python Makefile, and the exclusion of the portmap feature of *Twisted-Web*, installation of the dependancies and *sslstip* went smoothly. The final piece of Arsenic that was missing from iOS was the standard Unix command *tail*. Instead of compiling and installing tail, *arsenic-mobile* includes a python implementation that functions just as well.

5. Installation Process

5.1. Jailbreaking

The first step required to install *arsenic-mobile* on a device running iOS is to jailbreak the device. This will allow unsigned code to run on the device. Depending on the version of iOS the device is running this may or may not be possible. A cursory internet search will determine if it is possible and how to accomplish jailbreaking the device.

5.2. Setup

In order to install the necessary components and transfer files to the device running iOS, OpenSSH must be installed. From within the Cydia application search for OpenSSH and install it. The IP address of the device can be found under the Settings app in the Wi-Fi section by touching the blue arrow to the right of the network you are connected to.

5.3. Installation

Download *arsenic-mobile* from https://github.com/kernel-sanders/arsenic-mobile. Transfer the *arsenic-mobile* folder to the device by running

```
scp -r arsenic-mobile root@[IP Address]:
```

The default password is *alpine*. Then SSH into the device. To SSH into the device, from a computer on the same network run the following command (Linux or OSX in terminal, Windows will require an external program like *PuTTY*)

```
ssh root@[IP Address of the device]
```

Next, install python and run the setup script.

```
cd arsenic-mobile
cd Python
dpkg -i berkeleydb_4.6.21-4p_iphoneos-arm.deb
dpkg -i libffi_1%3a3.0.10-5_iphoneos-arm.deb
dpkg -i sqlite3-lib_3.5.9-2_iphoneos-arm.deb
dpkg -i sqlite3_3.5.9-12_iphoneos-arm.deb
dpkg -i python_2.7.3-3_iphoneos-arm.deb
cd ..
python setup.py
```

The setup script will install the rest of the dependancies and custom files necessary for *arsenic-mobile* to function correctly.

6. Functionality

arsenic-mobile has 3 modes of operation. First, it can preform a simple ping scan of the network (the gateway and the device's IPs will not be shown). Second, it can ARP spoof and man-in-the-middle attack a single victim. Finally, it can run in "Coffee shop mode" where it will scan the network every 5 seconds looking for new hosts. Any new hosts will be ARP spoofed and added to the man-in-the-middle attack. This attack mode would be useful in a high traffic public area with unsecured Wi-Fi. All captured credentials are printed to the screen along with the domain they are associated with, as well as written to a log file called "credentials.txt" in the current working directory.

7. Future Work

Arsenic-mobile has been tested in closed laboratory environments and has demonstrated the feasibility of an easy to use man-in-the-middle attack that defeats SSL/TLS. However, as websites have realized this vulnerability, many now keep their entires sessions in SSL (as opposed to only logon events or account management). When this is the case, *sslstrip* is unable to expire the session cookies to force the user to re-logon (allowing their credentials to be captured). There may yet be a way to force a user in a secure session to have to re-logon, and that would be an interesting area of further research. Beyond the technical aspects of the tool, having it packaged as a .deb file for ease of installation is another area to look into. Finally, a graphical user interface that would appear as an app tile on the home screen of the device and present a polished interface to the user could also extend this project.

8. Conclusion

Although great strides have been made to secure communications on the internet, the trusting nature of its founding protocols present exploitable vulnerabilities. Tools have been developed to demonstrate these vulnerabilities, and *arsenic-mobile* is the first of its these on Apple's iOS mobile operating system. The portability and ease of use that *arsenic-mobile* provides should only serve to emphasize the need for users to take extra care when browsing on open or untrusted networks.

References

[1] Netscape Corporation, "The ssl protocol," 1995.

[2] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (ssl) protocol version 3.0. rfc 6101," 1996.

[3] "Ssl pulse: Survey of the ssl implementation of the most popular web sites," 2013.

[4] D. Goodin, "Hackers break ssl encryption used by millions of sites," 2011.

[5] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thom, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit rsa modulus," Cryptology ePrint Archive, Report 2010/006, 2010, http://eprint.iacr.org/.

[6] Comodo Group, "Report of incident on 15-mar-2011," 2001.

[7] M. Marlinspike, "Defeating ssl in practice," 2009.

[8] KernelSanders, "Arsenic - a zero configuration arp poisoning and credential extraction tool," 2012, https://github.com/kernel-sanders/Arsenic.

[9] Apetrick, "Re: Sslstrip," 2012, http://ininjas.com/forum/index.php?topic=4537.0.

[10] Jean, "Cve-2010-3830 - ios <4.2.1 packet filter local kernel vulnerability," 2010, http://esec-lab.sogeti.com/post/2010/12/09/CVE-2010-3830-iOS-4.2.1-packet-filter-local-kernel-vulnerability.