David Fiala
CSC 705 – OS Security (Dr. Jiang)
Final Course Project
May, 2011

Firesheep for Android

Contents

Introduction

Launched in 2004, Facebook is a social network website that connects over 600 million users (1) in an online environment where each user may customize a unique profile that describes themselves and their interests. Additionally, users are encouraged embellish their profile by adding personal details which include but are not limited to their interests, activities, groups, present and past locations, education, work history, birthday, email, phone number, and more. Users are also able to upload pictures, videos, and other media that will be shown alongside their profile or in other locations on the site where they have been tagged in such media. While the list of potential content contained within a user's profile goes on, it is important to note the sheer amount personal information that is stored on any given profile along with its impact should it fall in the wrong hands.

Facebook content is by no means limited to the details displayed in an individual user's profile. Dynamic user-generated content such as wall posts wherein a user adds a short time-stamped message to their account public consumption are a very common occurrence. Other users will see these wall posts when they connect to Facebook and may themselves choose to publically respond to the posts. In essence, wall-posts are analogous to a public forum that is open to both public viewing and conversation. Such discussions may also be started by a user "Like"ing another website, group, picture, or other endorsement of any kind.

For individual communication, Facebook provides a private messaging service that allows two users to send messages to and from each other in a manner very similar to email. One major differentiating factor between email and Facebook messages is that Facebook messages are not routed across the internet between mail servers as email is. Instead a message will reside on Facebook's for its entire lifetime. Many users are turning to Facebook messages as an alternative to email for a variety of reasons such as reduced transmission time (email may have a delay as it is transmitted to its recipient's inbox) and the inherent convenience of not having to obtain a user's email address and launch your email if you are already logged in to Facebook.

With its large userbase and relatively simple ease of use, the millions of users that visit Facebook every month have poured in huge amounts of personal data and user-generated content as well as their implicit trust. With such a large personal investment in each person's account, there is actually a strong cause for worry about account privacy and safety. A potential account attacker has many reasons for attempting to access an account: View and modify profile details for profit or malicious purposes, access and send personal messages from the victim's account, create wall posts that endorse 3rd party products or ideas, stalking, and more. With so many reasons for account attacks coupled with an enormous user-base, Facebook has become a hacker's ideal target. In fact, over the past few years Facebook has been exposed for many security vulnerabilities (2) and privacy debacles (3) that range from unauthorized viewing of users accounts to complete account takeover (4). Throughout this time many security fixes and privacy "enhancements" have been proposed and implemented by Facebook in addition to a heavy cycle of press releases (5) that emphasize constant additions to account privacy and security controls.

While Facebook has addressed numerous critical vulnerabilities as they have been discovered, one key type of attack that involves network traffic tapping(sniffing) of unencrypted data in order to gain access

unauthorized account access has been left wide open. In fact, network traffic tapping, or sniffing, is a well known, longstanding security problem that is particularly problematic on open wireless networks that have become ubiquitous today in nearly all public places such as schools, restaurants, hotels, and places of work. Since using an open wireless network can allow any attacker in the vicinity to covertly take over any unencrypted Facebook account session, this problem becomes paramount to most other security vulnerabilities that Facebook has previously encountered. Worse, this problem has continued to remain unaddressed without a permanent solution for all Facebook users.

To help bring the much needed awareness to this problem, this research project's contributions are as follows:

- Design and implement an application that can covertly detect and hijack Facebook sessions that on an unencrypted wireless network
- Further, implement this project on a mobile handheld Android mobile device or smartphone to demonstrate how covert an account hijack may be
- Provide an intuitive, easy user interface that underscores exactly how simple it is for anyone to perform an account hijack

Background

Historically, many computer network protocols were not necessarily designed with security in mind. Generally, simplicity, portability, and speed were preferred over security. In fact, several very prominent network protocols are guilty of not adequately protecting data transmission such as FTP, Telnet, and rsh. Since their release, the desire to enhance the security of each of these tools has driven the development of modern equivalent such as SSH and SFTP albeit at a large cost in terms of encryption overhead on the CPU. Nonetheless, while some old network protocols without any transmission privacy still exist and are used to a lesser extent, one of the most prevalent insecure protocols, the hyper-text transfer language (HTTP), still exists and is extremely prominent.

Today, the one of the weakest links in internet privacy and security for end users is the lack secure communication channels and wide spread use of unencrypted HTTP traffic for nearly all web browsing. This situation is worsened when a user is browsing websites using HTTP only over an open wireless network. Since HTTP does not guarantee message authenticity of the client or server nor message privacy, it is possible for an eavesdropper to steal live traffic from an active wireless user. Using stolen internet traffic, it is possible for the eavesdropper to see exactly what other users are doing over the network or even use the stolen traffic out of context in order to impersonate the victim.

Network traffic packet sniffing is not a new concept and has been used extensively in packet sniffing applications such as tcpdump, Wireshark, and others. In fact, in late 2010 Eric Butler released a Firefox browser extension known as Firesheep (6) that incorporated packet sniffing to enable easy session hijacking of various websites including Facebook. With Firesheep it became possible for anyone with a laptop in range of a wireless Facebook user to both detect and hijack their active web session. This effectively allowed accounts to be taken over by any attacker within wireless range of the victim. Due to

the media attention and popular usage of Firesheep against unsuspecting users, Facebook launched an optional secure-HTTP (HTTPS) method in early 2011 for connecting to their site (7). In theory, the HTTPS option provided by Facebook was designed to prevent packet sniffing by transmitting all information of a secure channel that could not be read by any intermediate 3rd parties which would include attackers using Firesheep.

Motivation

As mentioned in the background, regular HTTP traffic is the leading (essentially monopolistic) protocol for web browser traffic. Given its inherent privacy limitations, one could essentially argue that using the HTTP protocol over an open network is the equivalent of inviting anyone within your wireless range to come and sit beside you on your laptop while they watch everything you view, say, and do on the internet. Unfortunately most users are oblivious to this risk and may entirely believe that what they do on their computer is kept private from those around them.

Exacerbating the situation are the press releases by many large social networking companies proclaiming enhanced privacy updates on a highly frequent basis. Thus, in turn this further leads average users to believe that their highly personal profile information is being safely guarded from potential attackers. In reality however, not only is profile information available to be theft via sniffing, but their entire account may be easily compromised including their friends profile information and even private messages. Resultantly, any privacy changes or additional controls that a social networking site provides to fine-tune exactly who can see your private information are entirely ineffective and are inconsequential when compared relative to the elephant-in-the-room security vulnerabilities residing with HTTP communication.

Unfortunately, even as social media networks such as Facebook add secure-HTTP access to their site, the default method that most users will continue to use is unsecure HTTP. However, by providing HTTPS some companies have mitigated both the technical users' complaints as well as media attention surrounding such tools as Firesheep. Nevertheless, most users are unaware of the continued threat of unencrypted packet sniffing and are still entirely vulnerable to Firesheep-like applications. Therefore this project's motivation is to return awareness to these issues in the hopes that all websites that transmit any type of sensitive data will require all users to use a secured communication channel at all times without exception. Additionally, as this project's result is an application designed to run on a rather inconspicuous mobile device, it is hoped that the covertness of using a small, portable mobile device will help underscore just how simple these attacks are and why they deserve immediate attention.

Design

For this project the focus will be on detecting and hijacking Facebook sessions on an open wireless network by sniffing unencrypted network traffic. The overall goal will be to design an android application that with simple controls will allow a user to both start and stop the sniffing as well as displaying a list of detected Facebook sessions in the wireless vicinity. After one or more sessions have been detected, they should be listed in a scrollable list along with the session user's Name, IP address,

and other pertinent information to identify each unique session. Finally, the application should provide its user with the ability to tap(click) on any one of the listed sessions and launch browser that will automatically open up to the victim's Facebook homepage already logged in as the victim. Each time the user clicks on any sniffed session they will have immediate control via hijacking of the victim's account on their mobile browser.

There are three design major sections of this project:

Sniffing Engine/Native Backend

At the lowest level, the native backend is responsible for capturing live network packets as they arrive over the wireless network. The native backend will apply basic heuristics to attempt to read IP and TCP headers as well as limited parsing of the HTTP protocol of all packets that are delivered over the wireless interface. The purpose of separating this logic from the actual Java Android application is to increase performance. In the event that the number of packets arriving is large and frequent, processing them in a natively compiled binary will require significantly less processing power than the same work being done in the Java VM.

After receiving TCP packets that pass initial validation checks, the native backend will investigate if the packet contains an HTTP header and attempts to extract any HTTP cookies if they exist in the header. Once HTTP cookies have been extracted, they will be communicated to the next stage (the Java backend) for further processing along with the source IP, HTTP user-agent, and HTTP request each sniffed packet.

Post-processing/Filtering Java Backend

At the Java backend level, we assume that input data from the native backend is at least in a valid format although perhaps not authentic or redundant. The Java backend will perform tests to ensure that each set of cookies obtained from the native backend has not already been seen (ignore redundant data) and that they are valid. For example, if a user floods invalid HTTP requests on to the network then it is desirable for the Java backend to both detect and ignore invalid sessions as well as potentially blacklist the source IP so that further resources are not wasted.

Upon confirming that a group of cookies is authentic, the Java backend will copy the cookies as well as some limited account information that relate to those cookies to the user interface.

User Interface

The user interface will be responsible for starting and stopping the sniffing backends as well as reporting to the user a list of sessions that have been successfully detected. Upon receiving a new group of session cookies from the Java backend, the UI will populate its list with the new session for the attacker to view. The finally responsibility of the UI is to launch a browser with the victim's cookies pre-injected whenever the attacker clicks on a detected session.

Implementation

This section aims to highlight only some of the notable implementation details of this project.

Wireless Promiscuity

Before the operating system is able to receive packets not destined for its specific MAC address the wireless hardware's firmware must be specifically instructed to enter a mode called 'Promiscuous mode'. Once activated, the operating system and its processes will be able to sniff packets which were meant for other devices on the network. Without promiscuous mode it is impossible to sniff other's network traffic and thus impossible to hijack their web sessions.

Packet Sniffing

To facilitate the capture of raw network packets, the native backend required the use of a packet sniffing library. Natively no such libraries exist in a pre-compiled form for the Android platform. To overcome this problem the source code for libpcap was cross-compiled for the ARM architecture and then statically linked to the native backend.

An additional challenge with packet sniffing was to filter only relevant traffic from all of the other network noise. This is an important problem as it is imperative that the mobile processor does not become overloaded by processing unrelated packets. Part of the implemented solution was to instruct libpcap to only listen for TCP traffic destined to port 80 which is HTTP traffic. Second, the first few bytes of each packet are inspected to make a guess as to whether the packet is actually a packet containing a HTTP packet header instead of in the middle of a stream (a download). The combined efforts of these two techniques significantly reduce the potential search-space of all wireless packets down to only those immediately relevant to hijacking HTTP cookies.

Communication between native and Java backend

To facilitate communication between the native backend and the Java backend basic IO pipes were used. Since hijacked session information is in a plaintext format and usually limited in size, the Java backend simply reads standard-output as a pipe from the native backend. To achieve this the Java backend is responsible for launching and creating a pipe to its child process. A standardized protocol was developed that allowed the Java backend to read the data being sent by the native backend. The format simply separated each relevant data item by a newline character. When a variable number of data items were to be transmitted, then a number indicating the number was pre-pended so that the recipient knew how many lines to read. Each group of data included the source IP, User-Agent, requested URL, and all cookies.

Checking the validity of a sniffed session

To detect if a sniffed session was valid or not the Java backend would probe the Facebook servers to test the cookies it received. If the sniffed cookies were in fact valid, then Facebook would return back to the application the user's profile without any error messages. Consequentially this also allowed the backend to determine the profile name of the hijacked cookies. On the other hand, if Facebook returned an error then it is assumed that the sniffed cookies were invalid. Upon receiving an invalid response the

application may optionally decide to blacklist any further cookies from the source IP to reduce any further processor and network waste.

Injecting cookies and hijacking the session

To provide the attacker with a usable hijacked session, the Android application made use of the Android 'WebView' object. Additionally, a cookie manager object is available that allows a developer to programmatically inject cookies in to the WebView framework. Besides WebView, it is not possible to show and control a website inside of an existing Android Application. The native Android browser does not provide the level of control needed to instrument the needed changes by our application. By default the WebView object allows an application to display an HTML page with little other functionality other than scrolling. Clearly an attack desires the ability to interact with a hijacked session so special hooks and overrides were put in place to allow Javascript to run, clicking of links, loading of images, overriding the User-Agent, and cookie modification.

Results

Development and testing was performed on a Nexus One Android device running Android version 2.2. This device was chosen as its wireless network device is capable of enabling promiscuous mode while several other test units were not due to firmware restrictions.

Experiments were run on NCSU's open wireless network with several Facebook accounts that had HTTPS disabled. Although it was challenging to get both test users' laptops on the same wireless channel as the test hijacking device, we were able to successfully detect and hijack all tested accounts whenever their session was sent over the network.

Interestingly, at the time of experimentation we had the test accounts turn on the HTTPS security option in Facebook. The expected results was to not be able to detect any Facebook sessions at all as the traffic should in theory be entirely encrypted. Quite to our surprise we detected that many Facebook plugins that 3rd party websites use to bring customized content from Facebook to their site were in fact requesting HTTP URLs instead of HTTPS. Due to this the victim's browsers were revealing their cookies unencrypted over the network without the user's knowledge. Session hijacking was available in these cases despite the user's attempt to ensure HTTPS traffic only.

Conclusion

This project successfully brought HTTP session hijacking capabilities similar to Firesheep to the Android platform. It has demonstrated that session hijacking remains a major concern to web security and is trivial to implement. Further, this project demonstrates that even with the limited capabilities of a handheld mobile device, hijacking is still viable. This indicates the potential for very covert session hijacking as a mobile device has become very mainstream in today's culture and not cause for suspect in public.

Due to the nature of the HTTP protocol, the most obvious solution to prevent attacks such as the one demonstrated in this paper is to ensure that all network payload are properly encrypted. The recommendation to take from this work is that it is imperative that all traffic be encrypted, and that there must be absolutely no exceptions to this rule. Any exception must be treated as an exploitable weak-link in the chain of security.

Finally, although this project was specifically designed to target Facebook users, it is important to remember that all websites not requiring encryption are just as guilty of not protecting their users. Today some major websites do provide HTTPS, but almost none of them absolute require it without exception. Until this step is taken we must not consider the HTTP security problems a closed issue.

References

1. [Online] http://www.businessinsider.com/facebook-has-more-than-600-million-users-goldman-tells-clients-2011-1.

2. [Online] http://www.theregister.co.uk/2008/03/25/facebook_exposes_private_pics/.

3. [Online] http://www.securitymanagement.com/news/facebook-security-hole-remains-unplugged-two-weeks-hackers-say-005777.

4. [Online] http://www.cbsnews.com/8301-501465_162-20029639-501465.html.

5. [Online] http://www.seattlepi.com/default/article/Will-Facebook-privacy-move-appease-its-foes-892990.php.

6. [Online] http://codebutler.com/firesheep.

7. [Online] http://www.itworld.com/security/134897/facebook-offers-protection-against-wireless-firesheep-attack.