DENIAL OF CONVENIENCE ATTACK TO SMARTPHONES USING A FAKE WI-FI ACCESS POINT

by

ERICH DONDYK

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program in Mechanical Engineering
in the College of Engineering and Computer Science
and in The Burnett Honors College
at the University of Central Florida
Orlando, Florida

Spring Term 2012

Thesis Chair: Dr. Cliff Zou

ABSTRACT

In this paper, we consider a novel denial of service attack targeted at popular smartphone operating systems. This type of attack, which we call a Denial of Convenience (DoC) attack, prevents non-technical savvy victims from utilizing data services by exploiting the connectivity management protocol of smartphones' operating systems when encountered with a Wi-Fi access point. By setting up a fake Wi-Fi access point without Internet access (using simple devices such as a laptop), an adversary can prompt a smartphone with enabled Wi-Fi features to automatically terminate a valid mobile broadband connection and connect to this fake Wi-Fi access point. This, as a result, prevents the targeted smartphone from having any type of Internet connection unless the victim is capable of diagnosing the problem and disabling the Wi-Fi features manually. For the majority of smartphone users that have little networking knowledge, this can be a challenging task. We demonstrate that most current smartphones, including iPhone and Android phones, are vulnerable to this DoC attack. To address this attack, we propose implementing a novel Internet-access validation protocol to validate a Wi-Fi access point by taking advantage of the cellular network channel. It first uses the cellular network to send a secret to an Internet validation server, and tries to retrieve this secret via the newly established Wi-Fi channel to validate the connection of the Wi-Fi channel.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor Dr. Cliff Zou for all his support. I would also like to thank my committee members Dr. Mainak Chatterjee, Dr. Xun Gong, and Dr. Mark Llewellyn**.**

I would also like to extend my sincere gratitude to Dmytro Dondyk for his great dedication in the experiment implementations portion of this project as well as Raquel Cohen and Roberto Claretti for their constant support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE: INTRODUCTION

Just like cellular phones before them, smartphones are quickly transitioning from being a novelty to an essential everyday tool. Having a single device capable of serving as a web browser, GPS navigator, portable media player, camera, and mobile phone can be very attractive to consumers. With more than half of the smartphone market share, the Android operating system is currently the most popular among the general public (Yarow). Unfortunately, as users become more dependent upon these devices, so does the interest of people seeking to exploit them. In the Android platform, a specific form of a denial of service (DoS) attack, which we call a denial of convenience (DoC) attack, can be performed with little technical background on a large number of users simultaneously.

Launching a DoC attack on the Android operating system can be achieved with relative ease. First, the attacker configures a computer as a Wi-Fi access point. Once this is done, the attacker makes sure this fake access point is not allowing any data traffic. Finally, the attacker deploys the fake access point in a heavily transited place where users are likely to use their smartphones. If deployed in a location where Wi-Fi Internet access is expected, such as a café, hotel, or airport, the attacker must position the fake access point closer to the victim than the legitimate access point to provide a stronger signal. When inside the coverage area of the open access point, Android will automatically disconnect from the mobile broadband in order to connect to the hotspot. However, because this fake access point does not have any Internet connectivity, the victim will be deprived of any form of Internet service.

A smartphone being targeted by this attack would display an optimal network connection status. Only by manually disabling the Wi-Fi capabilities of the device, would the victim be able return to the mobile broadband and, therefore, regain Internet services. However, with more than one third of all US adults currently owning a smart phone (Smith), we cannot expect the average user to be able to diagnose this exploit and successfully navigate through its solution. Therefore, developing an automated solution to this type of attack is highly desired.

To handle similar types of connectivity issues, traditional operating systems have developed several network awareness mechanisms. Microsoft's Windows, for instance, uses the Network Connectivity Status Indicator (NCSI) feature to verify the validity of an Internet connection. NCSI achieves this by sending a validation challenge to a predetermined website and comparing its response against the expected result (Appendix). If the validation response matches the expected result, the system assumes a valid Internet connection has been established. In this paper, we develop an Android application that implements a similar network awareness mechanism. We then test its effectiveness by exposing it to a DoC attack under real conditions.

Although this type of solution is widely implemented by traditional operating systems, its effectiveness against more sophisticated DoC attacks is limited. Its weakness lies on the fact that the validation key, the value returned in the validation response, must remain constant. An attacker, therefore, can easily fool the mechanism by acquiring the static validation key and providing it to the victim when her/his system performs a network awareness test. As a result, developing a more robust network awareness feature capable of withstanding this type of attack would be favorable.

In this paper, we propose a novel solution capable of overcoming this sophisticated DoC attack. It achieves this by overcoming the main flaw of the mechanism previously considered: the validation key is not a static value. This, as a result, makes it impossible for an attacker without

Internet connectivity to provide the information expected by the device. Thus, prompting the smartphone to disconnect from the fake Wi-Fi access point and reconnect to the mobile broadband.

In short, our paper makes the following contributions: (i) Exposes a specific type of DoS attack that Android’s Wi-Fi protocol is unable to prevent. In addition, demonstrate how it can be easily mounted on a large number of victims simultaneously. (ii) Applies a network awareness mechanism commonly used by traditional operating systems to prevent this type of exploit. We implement this solution in the form of a lightweight application and test it thoroughly under real conditions. (iii) Uncovers how this mechanism can be fooled by a more sophisticated version of the attack. Furthermore, demonstrate this by implementing an attack that fools the network awareness solution. (iv) Proposes a novel solution capable of overcoming the limitations of the previously implemented network awareness mechanism. Our solution requires no user intervention. Thus, making it especially attractive because of the large number of smartphone users that cannot be expected to diagnose and solve this type of exploit.

CHAPTER TWO: BACKGROUND

By design, Android smartphones connect to the Internet using only one channel at a time. For users with a data plan, this channel is normally their provider's mobile broadband. However, this channel may transition automatically to Wi-Fi if the user enters the coverage area of an access point and if the following two conditions are met: (i) the Wi-Fi capabilities of the device are enabled and (ii) the access point is open or has been previously accessed. Throughout this process, Android's connectivity manager does not check if the Wi-Fi access point has a functioning Internet connection. An attacker, therefore, can prompt a device to disconnect from the functioning mobile broadband by establishing a Wi-Fi access point. Then, deprive the victim of Internet services by blocking all data traffic through their access point.

In order to determine the status of an Internet connection, traditional operating systems have developed a variety of network awareness features. Windows, for instance, uses the Network Connectivity Status Indicator (NCSI) to achieve this. NCSI verifies that the targeted access point has Internet connectivity by performing two simple tests. In the first test, it attempts to load a webpage via HTTP through the newly established connection. This page, used solely by this Windows feature, only contains the text line "Microsoft NCSI". If this content is retrieved successfully, it assumes the access point is connected to the Internet. In the second test, NCSI verifies that the page dns.msftncsi.com resolves to the expected IP address 131.107.255.255**.** If either one of these tests is successful, it is assumed the access point is connected to the Internet (Appendix).

CHAPTER THREE: RELATED WORK

Previous works on smartphone security can be organized into two subjects. The first of these subjects is smartphone malware. Many works on this area of study focus on Android because its large share of the smartphone market has made it the most attractive mobile platform for malware development. In Backes' and Sastry's works, the malware is introduced into the system by escalating the permissions of a seemingly harmless application (Backes) (Sastry). Vidas considers different exploits that can be mounted on Android phones if an application has an unnecessary amount of permissions (Vidas). Schmidt demonstrates how the Android permission system can be bypassed by introducing malware that uses the native Linux applications of the device (Schmidt). Porter proposes a tool that assists developers or users in determining whether an app is overprivileged (Porter). Nauman, on the other hand, proposes a framework that enhances the default Android permission systems (Nauman). Obviously, these works differ from ours in that they do not consider network exploits.

The second subject of smartphone security research considers the implementation of traditional computer exploits and defenses on mobile devices. Because smartphones share many characteristics with regular computers, they also share their susceptibility to some widely used forms of attacks. Kumar, for example, demonstrates how several port scanning techniques can be used on Android to gather information about the device (Kumar). Then, use this information to mount different eavesdropping, man-in-the-middle, and denial of service attacks. Portokalidis, on the other hand, considers how resource limitations make it problematic or impractical for smartphones to use traditional security measures such as file scanners (Portokalidis). To overcome this, it proposes a model that delegates all security checks to a virtual replica of the phone hosted on a remote server.

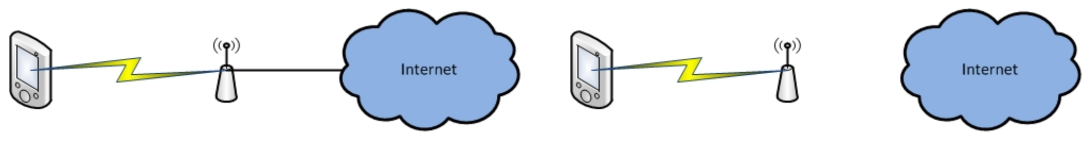
Currently, there are many works available on rogue access point detection. Several enterprise Wi-Fi security systems, for example, rely on lists of authorized access points to detect when a rogue access point is introduced into an area (Fluke) (Aruba) (WiMetrics). ETSniffer, on the other hand, provides the rogue access point detection capabilities to the end user. By utilizing network metrics to detect latencies characteristics of this type of exploits, ETSniffer is able of identifying evil-twin access points with a high level of accuracy (Song).

Our work is different from all previous works because it considers a form of attack unique to smartphones. Also, it is able to successfully implement on Android a defense mechanism currently employed by tradition operating systems. Finally it proposes a novel network awareness feature that relies on the mobile broadband connection of the device to provide an attack resistant authentication scheme.

CHAPTER FOUR: ATTACKS

4.1 Attack 1: Passive Access Point

The first method considered in this paper for executing a DoC attack is through a Wi-Fi access point without an Internet connection. When an Android smartphone enters the coverage area of a wireless router, it is automatically assigned an identifier and loaded into the Wi-Fi stack of the device. If the phone's Wi-Fi connectivity options are enabled and the access point is open or has been previously accessed, it will automatically connect to it. It will then terminate any ongoing mobile broadband connection that might have been established prior to the Wi-Fi connection. However, the device does not verify the access point has a functioning Internet connection at any time during or after the connectivity process. Therefore, by setting up a Wi-Fi access point that is not connected to the Internet, an Android device can be prompted to abandon its mobile broadband data connection to establish another one that does not provide any data. This, in turn, denies the user of any type of data service.



(a) Real AP (b) Fake AP

Figure 1: Real vs. fake AP

The DoC attack previously described can be executed in a variety of ways. One simple approach is through a wireless router that is not connected to the Internet. This method can be implemented with little resources and technical knowledge. Another possible approach is to configure a laptop as an access point, which can be achieved by using the free network software suite aircrack-ng, and making sure it is not allowing any data traffic (d'Otreppe). This approach allows the attacker to be unhindered when executing the attack.

Regardless of the implementation used, this type of attack can be very effective because: (i) The attacker has the ability to deny data services to a large number of victims simultaneously. (ii) From the victim's perspective, it is difficult to detect this type of attack because the device would display an optimal connection status. That is, Android would show that a Wi-Fi connection has been successfully established and that it is working properly. (iii) This type of attack can be executed without the use of sophisticated equipment or extensive technical expertise.

4.2 Attack 2: Fake Validation Response

A defense against attack 1 can be successfully mounted by implementing network awareness features similar to those used by traditional operating systems. These features test the Internet connectivity of an access point by sending a challenge to a validation server and comparing a key obtained in the response against the expected result (Appendix). For this to work (i) the key must be known by the device before performing the validation and (ii) the key stored in the validation server must remain constant. However, these conditions allow the attacker to obtain the key from the smartphone or the server. And, once the key is known, it can be used to the fool the feature by selectively responding to the probing packages at the time of validation.

In practice, there are a variety of ways in which such attack can be implemented. In this paper, we implement attack 2 by configuring a computer as an access point and redirecting all probing packets to a fake validation server. Because Android does not currently support ad-hoc

IBSS networks (MIC_888), it is necessary to configure the computer as a full Wi-Fi access point. This can be achieved by using aircrack-ng. Then, use the Linux application iptables to redirect all probing packages to a local server that mimics the validation server (Russell).

Using this implementation, the network awareness protocol is able to successfully retrieve the key. And, even though the key is retrieved from the attacker's computer, the access point is classified as valid. The connection to the fake Wi-Fi point is maintained, the device does not return to the functioning mobile broadband connection, and the victim is deprived of all data services. This approach also has the advantage of requiring few resources. Any laptop computer with a wireless card and a UNIX/Linux operating system is sufficient to successfully execute this attack 2.

4.3 Attack 3: Selective Internet Traffic Throttling

A successful defense against attack 2 could be formulated by implementing a challenge-response mechanism that relies on a dynamic key. That is, the key is different for every validation test performed. Even if such defense is possible, an attacker with Internet access could defeat it. Just like in the network awareness protocol previously discussed, the smartphone would still need to access a server to retrieve a validation key. Therefore, if the attacker has Internet access, it could fool the protocol by permitting the probing packets to reach the validation server while blocking all other traffic. This would allow the device to successfully retrieve the dynamic key. As a result, the device would remain connected to a fake Wi-Fi access point that does not allow other forms of traffic, thus successfully executing a DoC attack.

There are several ways of implementing this particular type of attack. One possible approach is almost identical to our attack 2 implementation. Essentially, it requires configuring a computer as a Wi-Fi access point, redirecting all probing packages to the validation server, and blocking all other traffic. This can be achieved by using aircrack-ng and iptables. However, this approach requires additional hardware. Two separate network interface cards are needed to establish the Wi-Fi access point and to maintain Internet connectivity. However, this additional hardware requirement can be easily satisfied by using the computers 10/100/1000 Gigabit Ethernet port or an inexpensive USB wireless adapter along with its internal wireless card.

CHAPTER FIVE: DEFENSES

5.1 Defense Against Attack 1: Static Identifier Validation Protocol

In order to counteract attack 1, we implement a network awareness protocol on Android in the form of a lightweight app. This app, which we call Wi-Fi Authenticator, automatically verifies that Wi-Fi access points have a functioning Internet connection without the need for any user intervention. To achieve this, Wi-Fi Authenticator relies on the following two-step process: (i) Every time a connection is established with an access point, Wi-Fi Authenticator sends a challenge to a validation server. If a response is not obtained within some time period, the access point is considered invalid. On the other hand, if a response is received, Wi-Fi Authenticator proceeds to step 2 of the validation process. (ii) Wi-Fi Authenticator retrieves a key from the validation response and compares it with a key stored in the device. If the keys match, the access point is considered valid. Otherwise, it is considered invalid. Step 2 prevents an attacker from easily fooling the authentication protocol by sending an arbitrary response to the challenge.

If the Wi-Fi access point is considered invalid in either step, Wi-Fi Authenticator terminates and disables the connection. This prompts the Android smartphone to transition back to a mobile broadband data connection returning data services to the victims. Also, it maintains the Wi-Fi capabilities of the device enabled allowing it to connect to other Wi-Fi access points that might become available in the future.

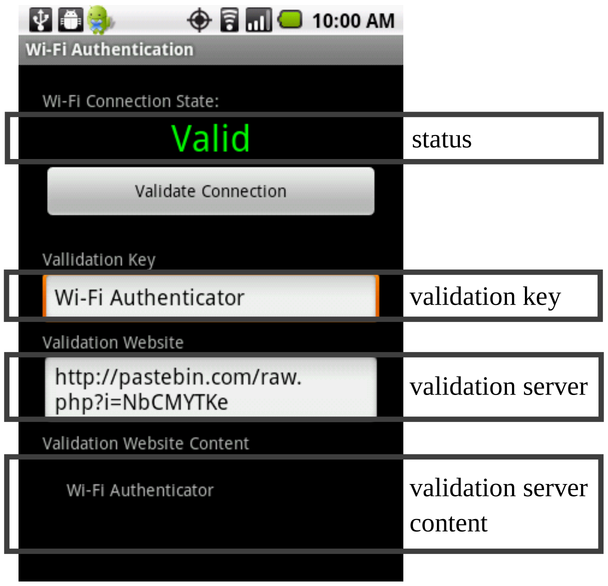


Figure 2: Wi-Fi Authenticator validation test

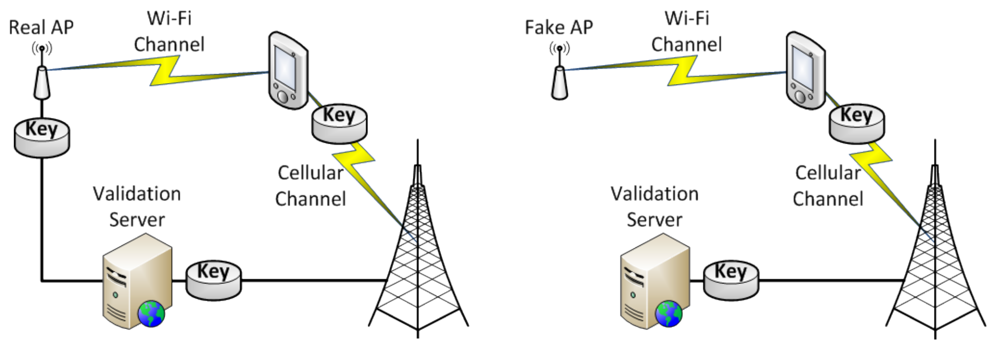
Implementing Wi-Fi Authenticator as an Android app provides several advantages. It allows the use of Android libraries in the development process. Thus, it improves its robustness and compatibility across different versions of the operating system. Also, it allows for easy distribution to existing smartphones through the use of Google Play. Finally, because it can be installed like any other application, it facilitates user implementation.

5.2 Defense Against Attack 2: Dynamic Identifier Validation Protocol

As demonstrated by attack 2, a more sophisticated attacker with greater technical knowledge could overcome defense 1. This is primarily due to the fact that the validation key used in this method of defense is always constant. In order address this weakness, we propose a novel network awareness protocol in which the validation key is dynamic. That is, it changes every time a validation test is performed. We achieve this by relying on the smartphone to generate a different key for every validation test it performs. An attacker, therefore, is unable to fool the protocol by supplying the expected response because it is not known by them.

This approach relies on the following five step process to validate a Wi-Fi access point: (i) After encountering an accessible Wi-Fi access point, the smartphone generates a random key and sends it along with its MAC address to the validation server through the cellular network. Depending on the user’s billing agreements, this data can be send as a SMS or TCP package. (ii) The validation server stores the random key in a table using the MAC address of the smartphone as the index. (iii) After transitioning from the mobile broadband to the Wi-Fi connection, the smartphone sends a challenge to the validation server. (iv) The validation server responds with the key corresponding to the smartphone’s MAC address. (v) The key obtained from the validation response is compared against that generated earlier by the smartphone. If equal, the Wi-Fi connection is considered valid. Otherwise, it is considered invalid.

Similarly to defense 1, this validation test is performed automatically without the need for any user intervention. Also, if a Wi-Fi access point is considered invalid, the connection is terminated and disabled. This allows the device to regain Internet services by reconnecting to the mobile broadband while maintaining its Wi-Fi capabilities enabled.



(a) ID validation with a legitimate AP (b) ID validation with a fake AP

Figure 3: Illustration of the dynamic ID validation protocol

5.3 Defense Against Attack 3: Network Performance Monitoring

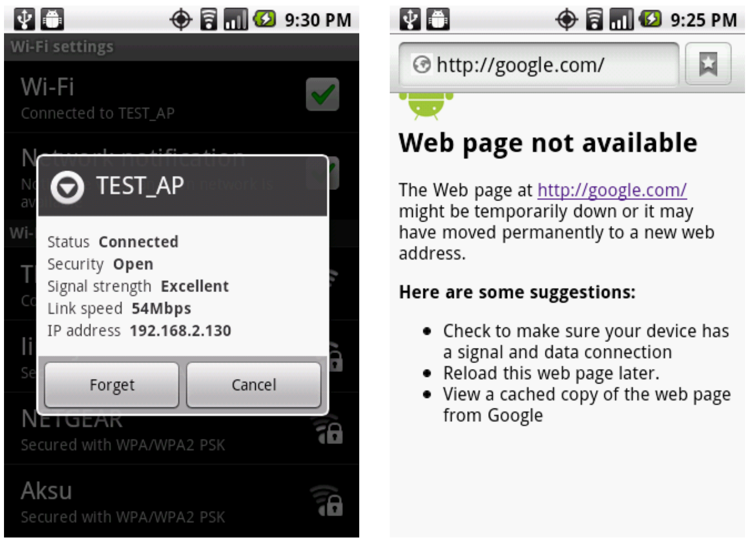
Given the case in which the attacker has Internet access, defense 2 might not be able to counteract attack 3. The effectiveness of defense 2 lies on the fact that the key is not known prior to the validation test. As a result, even if the attacker has the ability to intercept the validation challenge, it is not able to fool the protocol by returning the correct response. However, if the attacker has an Internet connection, it could execute a successful DoC attack by allowing the challenge to reach the validation server and blocking or throttling all other traffic. However, this weakness can be eliminated by expanding defense 2 into considering network metrics. Thus, if traffic blocking or throttling is occurring, the network awareness feature would measure the network performance is below a predetermined threshold. Then, using this information, it would regain data services by prompting the smartphone to transition back to the mobile broadband.

CHATER SIX: EVALUATION

6.1 Attack 1: Passive Access Point

In order to simulate a realistic scenario, we implement attack 1 by configuring a Linux laptop computer as a Wi-Fi access point. Although attack 1 could be easily implemented by using a wireless router without an Internet connection, we believe this particular setup is the most effective at simulating a real attack in which mobility is highly desired. Aircrack-ng is used to create the fake Wi-Fi access point through the native wireless adapter of the laptop computer. In order to prompt the targeted device to connect to our fake access point, we automatically assign an IP address to it after entering our coverage area. We achieve this by creating a DHCP server using the free Linux package dhcp3-server (dhcp3). When running both of these applications combined, we are able to successfully emulate a Wi-Fi access point.

We setup our fake Wi-Fi access point in an average sized room outside the coverage area of any other Wi-Fi access point. We then enter the room with an Android smartphone that has its Wi-Fi capabilities enabled. Finally, we perform an arbitrary Google query using the default Google Search app of the device. After erasing the fake access point entry from the Wi-Fi stack of the device, we repeat this test 20 times. In all cases, the smartphone connects to the fake access point, disconnects from the mobile broadband, and displays the Wi-Fi connection has a strong signal. However, at the time of the query, no information is returned nor a notification shown. Thus, a successful DoC attack is achieved.



Wi-Fi
Connected to TEST_AP

TEST_AP

Status Connected
Security Open
Signal strength Excellent
Link speed 54Mbps
IP address 192.168.2.130

NETGEAR
Secured with WPA/WPA2 PSK

Aksu
Secured with WPA/WPA2 PSK

http://google.com/

Web page not available

The Web page at http://google.com/ might be temporarily down or it may have moved permanently to a new web address.

Here are some suggestions:

- Check to make sure your device has a signal and data connection
- Reload this web page later.
- View a cached copy of the web page from Google

Figure 4: Wi-Fi connection status displayed with attack 1

6.2 Defense Against Attack 1: Static Identifier Validation Protocol

We implement our defense 1 network awareness feature as an Android app which we call Wi-Fi Authenticator. Wi-Fi Authenticator tests the connectivity of an access point by accessing a website, retrieving its content, and comparing it against a key phrase. This validation scheme is performed automatically every time a Wi-Fi connection is established. If Wi-Fi Authenticator is unable to access the website or if the content retrieved does not contains the key phrase, the access point is considered invalid and disabled. For these tests, we use the default Google page as the validation website (74.125.227.1) and the word "google" as the key phrase.

To evaluate the performance of defense 1, we execute an experiment very similar to that of attack 1. Just like before, we setup a fake access point in an average sized room outside the coverage area of any other Wi-Fi access point. We then expose a regular Android smartphone with Wi-Fi Authenticator installed to the fake access point. In all cases, Wi-Fi Authenticator is able to determine the Wi-Fi access point is invalid, disconnect from it, and reconnect to the mobile broadband.

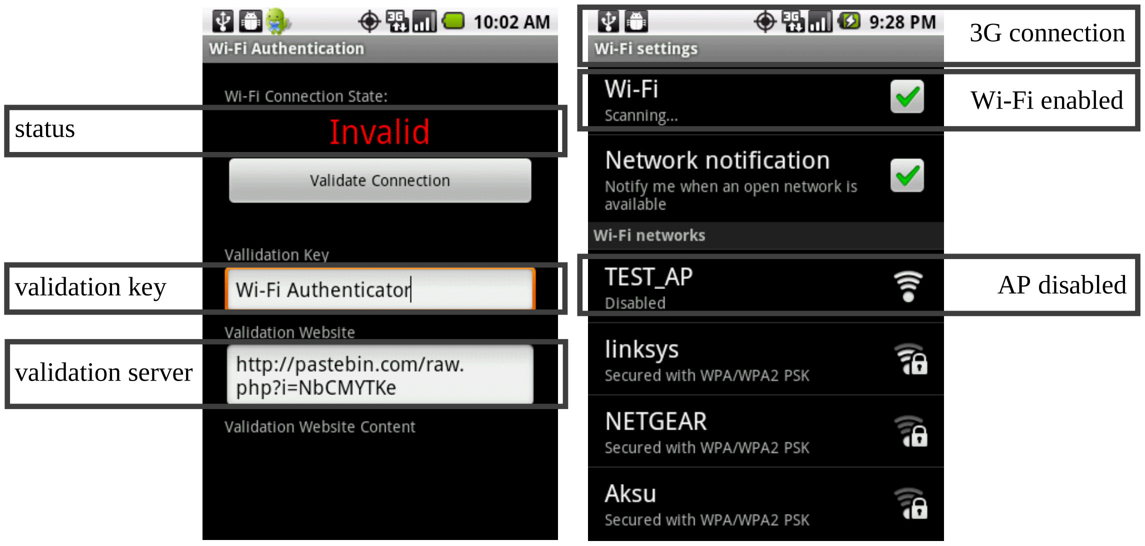


Figure 5: Detection of invalid AP using Wi-Fi Authenticator

Wi-Fi Authenticator is able to determine if a Wi-Fi access point is valid almost immediately. However, the time it takes for Wi-Fi Authenticator to detect a fake access point varies depending on the environment. The following figure shows the validation times for fake Wi-Fi access points. Three different devices were used to generate the fake access points in this experiment.

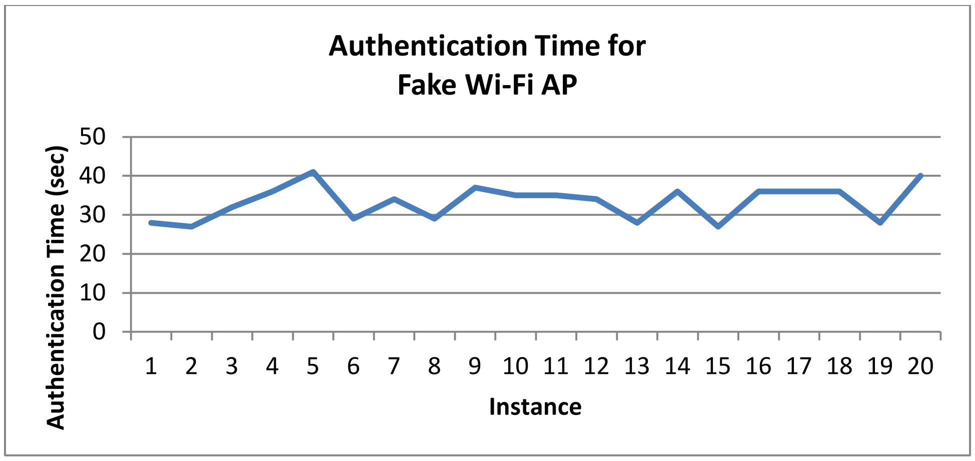


Figure 6: Authentication time for a fake Wi-Fi AP

6.3 Attack 2: Fake Validation Response

To implement attack 2, we first create a fake validation server. This is achieved by setting up an Apache HTTP Server in the attacker's Linux laptop computer (McCool). Because Wi-Fi Authenticator uses "google" as the key phrase of the validation scheme, our Apache server is configured to display the word "google". Just like in our attack 1 implementation, we configure the Linux laptop computer as a Wi-Fi access point using aircrack-ng. Then, we set up a DHCP server using dhcp3-server. This automatically assigns IP addresses to smartphones entering the coverage area and, thus, prompts them to connect to our fake Wi-Fi access point. Finally, we use iptables to redirect all track sent to 74.125.227.1, Google's homepage, to the IP address of the fake Wi-Fi access point network interface.

To measure the performance of attack 2, we deploy our attacker's laptop computer in an average sized room outside the coverage area of any other Wi-Fi access point. We then enter the room with an Android smartphone that has Wi-Fi Authenticator installed. This test is repeated 20 times erasing the Wi-Fi stack of the smartphone between each test. In all cases, Wi-Fi

Authenticator is unable to determine the fake Wi-Fi access point is invalid. Thus, the connection is preserved and the smartphone is denied Internet connectivity.

CHAPTER SEVEN: CONCLUSIONS

In this paper, we have considered a denial of service attack targeted at popular smartphone platforms. We present three possible approaches for executing this attack along with three defenses capable of counteracting them. We demonstrate, through implementation an evaluation, that such attacks are successful at achieving their purpose. Also, we demonstrate how each proposed defense is capable of counteracting the different implementations of the attacks. Our network awareness implementation is able to validate a Wi-Fi access points in less a minute.

REFERENCES

Yarow, Jay. “GARTNER: Android Market Share Doubles, iOS Drops In Q3.” Business Insider 15 Nov. 2011. 2 Dec. 2011 <http://articles.businessinsider.com/2011-11-15/tech/30400455_1_ios-iphone-smartphone-market>.

Smith, Aaron. “Smartphone Adoption and Usage.” Pew Internet 11 Jul. 2011. 2 Dec. 2011 <http://www.pewinternet.org/Reports/2011/Smartphones.aspx>.

“Appendix K: Network Connectivity Status Indicator and Resulting Internet Communication in Windows Vista.” Microsoft TechNet 2 Dec. 2011 <http://technet.microsoft.com/en-us/library/cc766017%28WS.10%29.aspx>.

Backes, Michael. Sebastian Gerling, Phillip von Styp-Rekowsky. “A Local Cross-Site Scripting Attack against Android Phones”. Saarland Uniersity, Aug 2011. <https://www.infsec.cs.uni-saarland.de/projects/android-vuln/android_xss.pdf>.

Sastry, B. V. S. S. R. S. and K. Akshitha. “Authorizing Stockpile Attacks on Android.” *International Journal of Mathematical Archive* 2.11 (2011): 2475-2479. Web.

Vidas, Timothy. Daniel Votipka, Nicolas Christin. “All Your Droid Are Belong To Us: A Survey of Current Android Attacks.” *Proceeding of the 5th USENIX Workshop on Offensive Technology*, 8 Aug. 2011, San Francisco, CA. Web. 12 Nov. 2011.

Schmidt, Aubrey-Derrick. Hans-Gunther Schmidt, Leonid Batyuk, Jan Hendrik Clausen, Seyit Ahmet Camtepe, Sahin Albayrak. “Smartphone Malware Evolution Revisited: Android Next Target?”. *Proceeding of the 4th Annual Malicious and Unwanted Software (MALWARE)*, 13 Oct. 2009, Montréal, Quebec, Canada. 2 Feb. 2010. 1-7. Web. 22 Aug. 2011.

Porter Felt, Adrienne. Erika Chin, Steve Hanna, Dawn Song, David Wagner. “Android Permissions Demystified”. *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS)*, 17 Oct. 2011, Chicago, IL. 627-638. Web. 17 Dec. 2011.

Nauman, Mohammad. Sohail Khan, Xinwen Zhang. “Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints”. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 13 Apr. 2010, Beijing, China. 328-332. Web. 14 Sep. 2011.

Kumar, Naresh and Muhammad Ehtsham Ul Haq. “Penetration Testing of Android-based Smartphones.” Thesis. Chalmers University of Technology, June 2011. Web.

Portokalidis, Georgios. Philip Homburg, Kostas Anagnostakis, Herbert Bos. “Paranoid Android: Versatile Protection For Smartphones.” *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, 6 Dec. 2010, Austin, TX. Dec. 2010. 347-356. Web. 29 Aug. 2011.

“AirMagnet WiFi Analyzer.” Fluke Networks 2012. Jan 14. 2012. <http://www.flukenetworks.com/enterprise-network/wireless-network/AirMagnet-WiFi-Analyzer>.

“AirWave™ Solution Guide.” Aruba Networks 2011. 14 Jan. 2012. <http://www.arubanetworks.com/pdf/products/SG_AW.pdf>.

“WiSentry – Wireless Access Point Detetion System.” WiMetrics 2006. Jan 14. 2012. <http://wimetrics.com/Products/WAPD.htm>.

Song, Yimin. Chao Yang, Guofei Gu. “Who Is Peeping at Your Passwords at Starbucks? – To Catch an Evil Twin Access Point.” *Proceeding of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN-DCCS 2010)*, 28 Jun. 2010, Chicago, IL. 9 Aug. 2010. 323-332. Web. 12 Aug. 2011.

MIC_888. “Android Ad-hoc Wireless Network Support.” Xdadevelopers 9 Mar. 2012 <http://www.xda-developers.com/android/android-ad-hoc-wireless-network-support/>.

d’Otreppe, Thomas. *aircrack-ng*. Vers. 1.1. Computer software. Aircrack-ng.org, 2010. Ubuntu. < http://www.aircrack-ng.org/index.html >.

Russell, Rusty. *iptables*. Vers. 1.4.12.2. Computer software. Netfilter Core Team, 2012. Ubuntu. <http://www.netfilter.org/projects/iptables/index.html>.

dhcp3-server. Vers 3.1.3. Computer software. Internet Systems Consortium, 2011.

McCool, Robert. *Apache HTTP Server*. Vers. 2.4.1. Computer software. Apache Software Foundation, 2012. Ubuntu. <http://httpd.apache.org/>.