

Portable Wireless Network Penetration with Android

Zach Yannes

Florida State University
yannes@cs.fsu.edu

Yaroslav Kechkin

Florida State University
kechkin@cs.fsu.edu

Abstract

In today's internet, servers can distribute potentially surreptitious data to several clients, now using predominantly wireless networks. A wireless network has the advantage of allowing many clients to wirelessly authenticate their credentials and connect to the network, where the data transactions can then take place. However, unlike a wired network, clients can forge the authentication in order to connect to the network. As a result, wireless networks implement one or more security measures in order to prevent unauthorized clients from connecting to a network.

The security measures of a wireless network can be tested to reveal potential vulnerabilities, known as wireless network penetration testing. Most wireless network penetration test programs provide the necessary features for testing common vulnerabilities. However, these programs are often very expensive and too complicated to be used by a non-expert of computer security. As a result, we propose a new Android application which performs the necessary capabilities to test the security of a wireless network. The application will provide the following features: Network scanning, Man-in-the-Middle attacks, identifying network devices, fingerprinting operating systems, determining the strength and security protocols of a network, penetrating WEP/WPA/WPA2, impersonating SSID, detecting spoofed IPs, and sniffing traffic [1–3].

The application is developed for Android with the focus in mind for distribution and ease of use. Many of the features and algorithms for network penetration testing use the Aircrack-ng library. As a result, the Aircrack-ng library is cross-compiled for Android support. Furthermore, the application requires a rooted phone and custom kernel functions in order for the WiFi adapter to support promiscuity mode, however this can be avoided using an external WiFi adapter.

1. Introduction

The security measures of a wireless network can be tested to reveal potential vulnerabilities, known as wireless network penetration testing. Most wireless network penetration test programs provide the necessary features for testing common vulnerabilities. However, these programs are often very expensive and too complicated to be of any practical use by a non-expert of computer security. As a result, we propose a new Android application which performs

the necessary capabilities to test the security of a wireless network. The application will provide the following features: identifying network devices, sniffing traffic, network scanning, fingerprinting operating systems, determining the strength and security protocols of a network, penetrating WEP/WPA/WPA2, detecting spoofed IPs, impersonating SSID, and performing Man-in-the-Middle attacks.

In order to provide certain packet monitoring functionality, the network interface must be put into monitor mode, although not all network interface cards (NICs) support this mode. Therefore, it is important to list the network interfaces in order to select the NIC which can be properly set in monitor mode.

Once the NIC has been configured, it can begin intercepting packets without connecting to a network. This is important for capturing handshake packets, the packets sent between router and client to establish a connection, which can be used to crack WEP, WPA, and WPA2 network passwords.

Network scanning provides a portable method for enumerating nearby access points. Using this feature, the application collects enough data to provide a list of targets to the user.

One key step to penetrating a server is to collect data about the operating system and public services provided by the server. With this information, a specific attack can be selected which exploits known vulnerabilities in the operating system or services, such as an out-of-date openssh.

In order to crack the network password, a specific method must be selected based on the network's security protocols. For instance, WEP networks can be cracked simply by the brute-force method given a captured packet file, whereas WPA/WPA2 networks use an algorithm which requires multiple captured packets and a password file and/or mangler rules file.

Since different network protocols are available, it is important to be able to determine vulnerabilities in each configuration. As a result, we evaluate the WEP, WPA, and WPA2 network protocols for penetration vulnerabilities. By providing different password files and mangler rules files, the probability of successful cracking and cracking duration can be varied.

One method ubiquitous in malicious attacks is spoofing an IP in a packet sent to a server or client. It is used to trick the receiving party into believing they are communicating with the owner of the spoofed IP, rather than the attacker. Therefore, a penetration test should evaluate whether incoming packets contain spoofed IP addresses. One such technique is to send a test packet when a packet is received to the sender. Then trace the path taken to determine if it matches the IP address of the packet.

While many attacks use spoofed IPs, a penetration can also be through impersonating a router's SSID. This allows an attacker to establish a fake copy of the router's network in order to lure clients into the false network.

The final attack evaluated in our penetration test is a Man-in-the-Middle (MitM) attack. This attack impersonates both the client and router in order to intercept and modify the packets sent between the two parties.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](http://permissions.acm.org).

CONF'yy, 2014.

Copyright © 2014 ACM 978-1-xxxx-xxxx-n/yy/mm...\$15.00.

<http://dx.doi.org/10.1145/nnnnnnnn.nnnnnnn>

2. Application Development Process

The application is developed for Android with the focus in mind for distribution and ease of use. Many of the features and algorithms for network penetration testing use the Aircrack-ng library. As a result, the Aircrack-ng library is cross-compiled for Android support. Furthermore, the application requires a rooted phone and custom kernel functions in order for the Wi-Fi adapter to support promiscuity mode, however this can be avoided using an external USB Wi-Fi adapter.

We have currently finished the cross compilation of Aircrack-ng and Nmap. These modules have been completed using the Android NDK, Revision 10c for android-17 ARMV7 devices. This build can be used by any device running Android 4.2.2 or later. The application has been separated into three development modules: the user interface, the network analysis, and the expert system.

The user interface contains the Listviews and Fragments which display the tests and test results, as well as the gesture and action listener functions for handling user interaction and navigation through the application. The data collection modules are required to run before the network analysis modules. Each data collection module runs on an individual Fragment, which provides more efficient memory and user interface management.

The network analysis modules perform the actual hacking tasks of the networks. The asynchronous threading was implemented using the Android AsyncTask class, which provides background functionality and notifications upon completion. By running the tasks asynchronously, the user can navigate away from that page, start other modules, and return whenever the task completes. A screenshot of the application starting an *airodump* module is seen in Figure 1.

The expert system is the heart of this project. It is what converts the methods of network penetration into a systematic list of operations. The expert system runs in the background whenever an AsyncTask is started. Therefore, the system can asynchronously iterate through potential attacks and relay them back to the Fragment to be displayed. Using this framework, we can easily add new attacks and vulnerabilities to search for.

3. Network Discovery

Most phones and tablet PCs are not capable of searching for hidden networks on their own. There are several apps on the market: Wi-Fi analyzer, G-Mon, Wififorum, WiFi Buddy, but those apps don't always work properly. The reason is that wireless adapters on most android devices are not capable of entering promiscuous mode to capture and analyze packets that are sent by routers. They work

only with routers which broadcast their SSID. In order to have this capability, the phone or tablet PC must have Broadcom BCM43xx or Ralink 28xx/307x wireless chipsets, require device to be rooted, and have several modifications to existing ROM. Some known devices that have said chipsets installed are Nexus 7, Nexus One, Galaxy S1 through S5, and HTC Desire HD.

Since none of us owns said devices, we had to buy an external wireless Network Interface Card (NIC) adapter with a compatible chipset. In order for the device to interact with the external NIC adapter it is connected using an on-the-go (OTG) cable which also provides power to the external NIC adapter. OTG support also requires specific drivers, but fortunately our test devices (LG G2 and Samsung Galaxy Nexus S 4G) support OTG from the stock kernel. Putting a Wi-Fi device into monitor mode allows it to capture all sorts of interesting data about the wireless network that would otherwise be invisible. Monitor mode packet captures are extremely useful for network diagnostics and penetration testing. The external NIC adapter will capture packets and scan networks to collect data and then send it to the device to be displayed to the user. Then, the user can select a specific access point to target in an attack.

4. Cracking Router Security

In order to have password cracking capabilities, we took the Aircrack-ng from Kali Linux distribution, and recompiled it for the android. Aircrack-ng is an 802.11 WEP and WPA-PSK keys cracking program that can recover keys once enough data packets have been captured. It implements the standard FMS attack along with some optimizations like KoreK attacks, as well as the PTW attack, thus making the attack much faster compared to other WEP cracking tools.

Our program starts with using Airodump-ng module of Aircrack-ng suite. It is used for packet capturing of raw 802.11 frames and is particularly suitable for collecting WEP initialization vector (IV) for the intent of using them with aircrack-ng. Additionally, airodump-ng writes out several files containing the details of all access points and clients seen. After running, airodump-ng will display a list of detected access points, and also a list of connected clients.

Figure 2 shows that the program captures and displays the BSSID (MAC address of the wireless access point (AP)), the MAC address of the connected client, the signal strength, number of packets sent by client, ESSID (the name of wireless AP), rate data rate from AP to client and back (36 Mbits per second and 24 Mbits per second). We can further limit all data captures to only one specific AP by specifying the mac address of the selected AP. To speed up the process of cracking wireless security we can run aircrack-ng together with airodump-ng. This way aircrack-ng will attempt to crack the security as soon as enough IVs had been captured. To crack WPA/WPA-2 security we need to capture full handshake protocol messages and then use password dictionaries to crack the password. For this task we obtained two password lists of 20,000 and 20,000,000 passwords and ran our program with both. Since we are using the phone, to run through 20 million passwords close to 24 hours. To speed up the process of password cracking, we cross compiled John-the-Ripper password cracker. John-the-Ripper takes a password list and a mangler rules list as input and attempts to crack the password by generating a dynamic list of passwords by applying each mangler rule to each password. This can be used to generate a massive list of potential passwords. However, it also has a greater chance of success due to the ability to insert information about the network owner or company. Since a password is likely to reflect an individual's behavior or interests, it is important to profile the target to generate a more specific password list.

Figure 1: Running an airodump scan

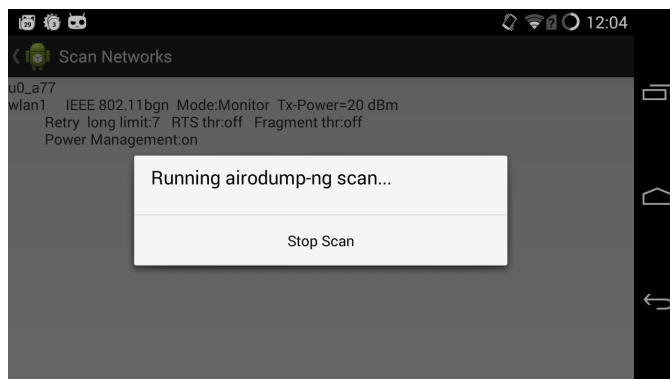
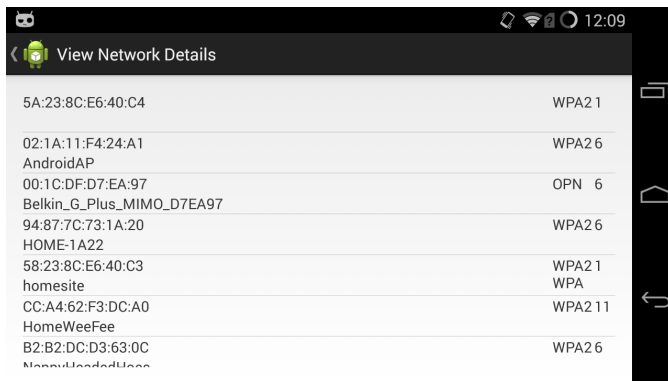


Figure 2: Results of network scan



MAC Address	Security Protocol
5A:23:8C:E6:40:C4	WPA21
02:1A:11:F4:24:A1	WPA26
AndroidAP	
00:1C:DF:D7:EA:97	OPN 6
Belkin_G_Plus_MIMO_D7EA97	
94:87:7C:73:1A:20	WPA26
HOME-1A22	
58:23:8C:E6:40:C3	WPA21
homesite	WPA
CC:A4:62:F3:DC:A0	WPA211
HomeWeeFee	
B2:B2:DC:D3:63:0C	WPA26

5. Network Configuration Analysis

To discover the network layout, we cross compiled Network Mapper otherwise known as NMAP. NMAP has a wide range of options to map networks and machines that are on the same network. It scans all IP addresses on the subnet and discovers which are assigned to each machine. It also performs a port scan, which determines the open ports are on the machines. With options -A and -O, NMAP detects the version of operating systems which is running at the scanned IP addresses. NMAP also detects if a machine is protected by a firewall or intrusion detection system (IDS). If a firewall or IDS is discovered, nmap can use fragmented packets, spoofed MACs, and proxy servers to bypass the obstacle.

These tests provide vital information for planning a viable attack on the targeted computer. The network configuration data is applied to an artificial-intelligence-style expert system which uses the data to narrow down the best possible attacks on the system. For instance, provided the OS configuration it will select a subset (or union of the subsets) of the attacks: Windows-specific, MAC-specific, Linux-specific, Non-OS specific, etc. Then, by providing a list of open ports, the user can choose to use a port-specific attack, such as ARP or SSH injection.

Why do we need to detect clients and fingerprint which operating systems they are running? It is useful for several reasons: network inventory, detecting unauthorized devices, social engineering, determining vulnerabilities of target hosts, and creating specific exploits. It is important to know what is running on the network for many administrative reasons. Unauthorized devices can pose a serious threat to the security of the network: employees can buy a cheap AP and install it in a cubicle without realizing that the protected corporate network had just been opened to any attacker.

It is sometimes a very difficult to determine remotely whether an available service is susceptible or patched for a certain vulnerability. Even obtaining the application version number doesn't always help, since OS distributors often back-port security fixes without changing the version number. The surest way to verify that a vulnerability is real is to exploit it, but that risks crashing the service and can lead to wasted hours or even days of frustrating exploitation efforts if the service turns out to be patched. OS detection can help reduce these false positives. Even after you discover a vulnerability in a target system, OS detection can be helpful in exploiting it. Buffer overflows, format-string exploits, and many other vulnerabilities often require custom-tailored shellcode with offsets and assembly payloads generated to match the target OS and hardware architecture. In some cases, you only get one try because the service crashes if you get the shellcode wrong. If OS detection is not

used first you may end up sending Linux shellcode to a FreeBSD server.

6. System Initialization

The Aircrack-ng cross-compilation steps are fairly straightforward. First, an Android cross-compiler must be created to perform the compilation using the Android libraries. This is done by using the *make-standalone-toolchain.sh* command provided in the *Android NDK*. Second, the *Makefile* values must be converted to match the Android environment and cross-compiler. Third, the compilation is performed with the cross-compiler, which creates the statically-linked libraries, drivers, and binaries. Finally, files are then copied to the Android device with the corrected permissions for execution.

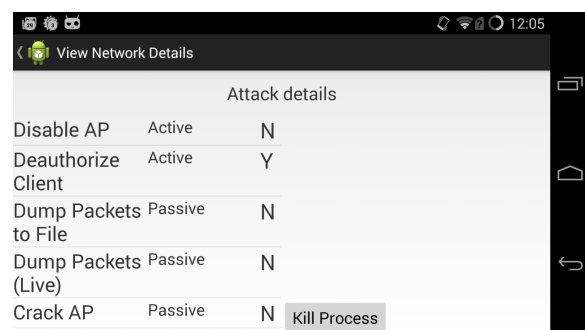
7. Application Flow

Upon first boot of the application, it will initialize the installation mode. This step installs all the necessary files to the device and corrects any user permissions. At this point the USB wireless NIC is plugged into the phone using a microUSB-to-USB female cable. Once the NIC is detected, it will set it into *monitor mode* and identify the NIC's interface name to be stored in the *SharedPreferences*, which is a local persistent database for future use in the application.

The application provides a simple interface for the complex methods of network penetration. To do this, the user first performs a scan of nearby networks displayed in a *Listview* (Figure 2). Next, the user chooses a network as the network penetration target. This moves to another *Listview* containing data collection options for analyzing the network configuration such as OS identification and port scanning. From here, the user can select multiple methods for analyzing the configuration of the network, which runs the tasks asynchronously. By running the tasks asynchronously, the network analysis can be parallelized. Once a task completes, it again stores the results as a *SharedPreferences*. This allows the user to leave the application and perform the tests in a background task, which then notifies the user when the tasks are completed.

Once the tasks are completed, the expert system analyses the data to select a subset of viable attacks on the network. In Figure 3, a list of attack *categories* are displayed to the user. Each category uses a different aircrack-ng command. However, the commands are customized to the attack target through flags and configuration files. These are all supplied automatically through the collected data stored in the *SharedPreferences*. Therefore, the user can perform a specific attack on a system without any knowledge of how the attack works. This also allows the application to easily support more commands by allowing the developer to abstract the commands through the *category* class.

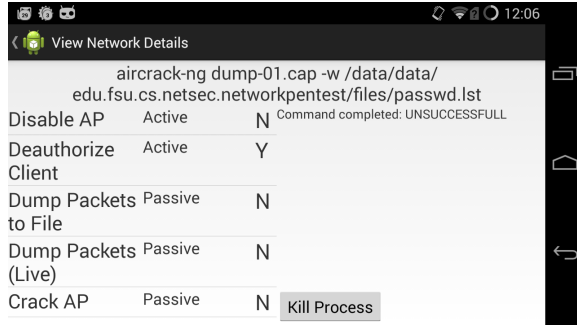
Once again, the application runs each of the tasks on individual asynchronous threads and notify the user upon completion. Each



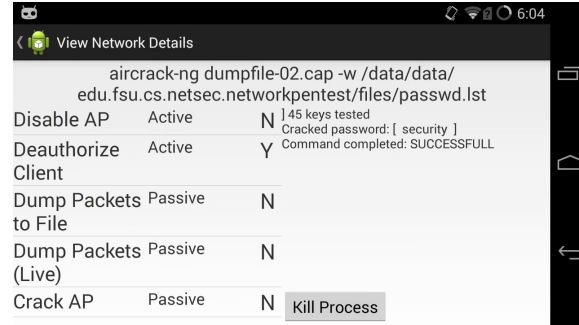
Attack details		
Disable AP	Active	N
Deauthorize Client	Active	Y
Dump Packets to File	Passive	N
Dump Packets (Live)	Passive	N
Crack AP	Passive	N

Kill Process

Figure 3: List of attack categories



(a) Password cracking unsuccessful



(b) Password cracking successful

Figure 4: The two aircrack-ng responses

test requires the Aircrack-ng library which is embedded within the application, so the user does not have to cross-compile or install anything manually. The library is accessed in Java using the Android Java Native Interface (JNI), which performs C library calls from within a Java Process running asynchronously. After test is completed, the user can view the results, ending either with "Command completed: UNSUCCESSFUL" or "Command completed: SUCCESSFUL", shown in Figure 4.

8. Future work

The idea for this project came from the need for a portable network penetration testing method. Due to the time constraints of the project, we have narrowed the scope to focusing only on the first phase of network penetration testing, gaining access to the wireless network through the access point.

While this application simplifies many of the steps in a penetration test, it does not provide a fully automated wireless penetration test of a network by analyzing the network once access has been granted. As a result, we researched the process of penetration testing from within the network. We have begun working on the following features to add to the application which support this capability.

First, the penetration test must perform distributed denial of service attacks. While the current application can provide a form of Denial of Service (DoS) attacks through deauthorization of clients from the router, it cannot perform a distributed DoS without internal network access. Once this is achieved, the attacker could send request packets to the server or router spoofed with IPs from all the clients on the network. Eventually, the server would become overwhelmed by the number of connections and either stop accepting new clients or crash. This sort of attack is common given certain vulnerabilities in the configuration of the router, server, and firewall. Therefore, this could be a useful feature for detecting such vulnerabilities.

Second, the penetration test must perform code injection and cross site scripting (XSS) attacks and analysis. Code injection attacks modify the execution of a program in order to change how the user interacts with the server. Many attacks are so powerful that it enables the attacker to upload to and execute any files on the server, including granting ownership of directories. XSS attacks typically involve exploiting a browser form in order to inject a malicious script into the site's code. This allows the attacker to execute scripts between otherwise trusted parties, giving the attacker permission to then execute scripts on the client's machine. Many open source scanners are available which can detect code injection and XSS vulnerabilities, such as RIPS and Maxisploit[6, 7]. The functionality of

these scanners can be included in the application to detect and alert users of potential vulnerabilities.

Finally, the penetration test must evaluate the test results to provide actual solutions to the vulnerabilities. The current state of the application simply exposes the possibility of entry into the wireless network. However, from the perspective of the network's developer, the application does not provide much in the form of preventing the attacks. An important feature is for the application to evaluate all of the collected results, and provide a single document of detected points of entry into the system. In creating a list of all known vulnerabilities, each can be evaluated based on attack method and network configuration in order to suggest a solution. For instance, if the application cracks a WEP network through brute-force cracking and discovers the password is "password", the application would suggest the user to change the password to be longer and more unique.

While this was designed to demonstrate the availability of a portable wireless network penetration method, it is only a proof-of-concept. The application provides methods of penetration which, if in the hands of a malicious user, could be used for illegal activity. As a result, we have decided not to release the application on the Google Play Store.

9. Conclusion

The purpose of this application is to provide a proof-of-concept for a portable wireless network penetration testing device. This application wraps the aircrack-ng and nmap libraries in a Java-based Android application to abstract the long and complicated process of penetrating a wireless network into a simple user interface. The application scans for nearby networks, evaluates and profiles the network configuration, and then performs the attacks. The scope of this application was focused on simply gaining unauthorized access to a wireless network. However, we discussed the future features we can implement to further improve and expand the penetration test. This project has demonstrated that a mobile device is capable of performing the steps necessary in penetrating a wireless network.

References

- [1] Core Security. "Wireless Network Penetration Testing". (2014).
- [2] Kali Linux. "Official Kali Linux Documentation." (2014).
- [3] Secure State. "Wireless Attack and Penetration." (2014).
- [4] Aircrack-ng. "<http://www.aircrack-ng.org/documentation.html>". (2014).
- [5] Nmap. "<http://nmap.org/book/man.html>". (2014).
- [6] RIPS. "<http://rips-scanner.sourceforge.net/>". (2014).
- [7] Maxisploit. "<https://code.google.com/p/maxisploit-scanner/>". (2014).