AN APPROACH TO DETECT SSL VULNERABILITIES IN ANDROID APPLICATIONS

By,
Shah, Prachi (pracshah@indiana.edu)
Veeramachaneni, Yugandhar (yugveera@indiana.edu)

CSCI-B 649 - TOPICS IN SYSTEMS: Malware - Threat and Defense
Final Report
2014

1. INTRODUCTION

Problem

Google's Android eco-system is all about mobile applications that communicate with their respective services via the Internet. This communication has to be secure as a lot of personal and sensitive information sharing occurs between the web services and the mobile applications. Such a communication is achieved by using *Secure Sockets Layer/ Transport Layer Security* (SSL/TLS) cryptographic protocols. However, the recently published SSL vulnerabilities like the Heartbleed [1] bug make the application eco-system scream for help. One of the most popular attacks, the *man-in-the-middle* (MITM) attack is causing a havoc. The vulnerabilities in SSL lead to attackers gaining hands-on all the personal data that our applications pass through its web service to fulfill the process of communication. While, this communication is necessary in most cases to personalize the user experience, it could cause great damage to the user both emotionally and monetarily. Imagine a scenario when a user loses his/her personal/private photos when a hacker hacks the vulnerable application used by the user and shares those photos over the Internet. The recent Snapchat [2] application photo leaks are a good example of this. Such events are not acceptable. A lot of similar attacks are caused due to expired or incorrect digital certificates, use of old encryption protocols and improper data privacy policies and so forth, which allow an attacker to easily intercept intermediate traffic and tamper it in order to attack the target user. If an attacker is interested in performing MITM attacks, they're already doing it. That cat is already out of the bag. They've likely set up a rogue access point and are already capturing all of the traffic that passes through it. Further supporting this suspicion is the fact that the FTC has already filed charges against the authors of two mobile applications that fail to validate SSL certificates. Knowing which specific applications are affected does not give any advantage to an attacker. If end users have vulnerable applications on their phones, knowing which applications are affected does give an advantage to the defenders. They can choose to uninstall vulnerable applications until fixes are available, or if they must, they can choose to use said applications only on trusted networks.

Proposed Approach

Our project aims at identifying those Android applications that continue to use SSL 3.0 protocol to secure their communication. *SSL 3.0* is an old encryption standard which is now replaced by Transport Layer Security (TLS) protocol that is considered to be more secure. Using SSL 3.0, an attacker can easily decrypt and extract information from within an encrypted communication. Hence, disabling SSL 3.0 support completely from all the existing systems/ applications and replacing them by TLS is the need of the hour. We use *MITMProxy* [4] to detect vulnerable applications.

Keywords: *Man-in-the-middle, HTTPS, proxy, AVD, Android, SSL.*

2. RELATED WORK

Although in contemporary times, TLS is the de facto standard for securing communications, there exist several servers/ applications that still continue to support old encryption protocols like SSL 3.0 to ensure backward compatibility. When the three-way Transmission Control Protocol (TCP) handshake process starts, the client offers the highest protocol version that it can support for communication. If this fails, it assumes that the server does not support this version and hence falls back to a lower protocol version. This is not a proper protocol version negotiation and so, if the server says that it supports TLS 1.0 and the client starts the handshake with TLS 1.2, the server will instead downgrade the protocol version to TLS 1.0 for the communication. Now, such a communication failure can occur due to multiple reasons like lower protocol support by the server, incorrect protocol version negotiation between the client and the server, a network failure and so forth. So, if an attacker that controls the network between the client and server interferes with any attempted handshake that offers TLS 1.0 or later, such situation can lead to an immediate fall back to SSL 3.0 protocol. SSL Poodle Vulnerability attempts to attack and thereby detect all the systems and applications that utilize the Secure Socket Layer (SSL) 3.0 with cipher-block chaining (CBC) mode ciphers for communication.

A Little Overview of Poodle
More about this in the paper - This POODLE Bites: Exploiting The SSL 3.0 Fallback [3].

To work with legacy servers, many TLS clients implement a downgrade dance: in a first handshake attempt, offer the highest protocol version supported by the client; if this handshake fails, retry (possibly repeatedly) with earlier protocol versions. Unlike proper protocol version negotiation (if the client offers TLS 1.2, the server may respond with, say, TLS 1.0), this downgrade can also be triggered by network glitches, or by active attackers. So if an attacker that controls the network between the client and the server interferes with any attempted handshake offering TLS 1.0 or later, such clients will readily confine themselves to SSL 3.0.

Encryption in SSL 3.0 uses either the RC4 stream cipher, or a block cipher in CBC mode. RC4 is well known to have biases [RC4-biases]. The most severe problem of CBC encryption in SSL 3.0 is that its block cipher padding is not deterministic, and not covered by the MAC (Message Authentication Code): thus, the integrity of padding cannot be fully verified when decrypting.

In the web setting, this SSL 3.0 weakness can be exploited by a man-in-the middle attacker to decrypt “secure” HTTP cookies, using techniques from the BEAST attack [BEAST]. To launch the POODLE attack (Padding Oracle On Downgraded Legacy Encryption), run a JavaScript agent on evil.com (or on http://example.com) to get the victim’s browser to send cookie-bearing HTTPS requests to https://example.com, and intercept and modify the SSL records sent by the browser in such a way that there’s a non-negligible chance that example.com will accept the modified record. If the modified record is accepted, the attacker can decrypt one byte of the cookies.

The attacker proceeds to the next byte by changing the sizes of request path and body simultaneously such that the request size stays the same but the position of the headers is shifted , continuing until it has decrypted as much of the cookies as desired.

The expected overall effort is 256 SSL 3.0 requests per byte. As the padding hides the exact size of the payload, the cookies’ size is not immediately apparent, but inducing requests GET /, GET /A, GET /AA, ... allows the attacker to observe at which point the block boundary gets crossed: after at most 16 such requests, this will reveal the padding size, and thus the size of the cookies.

3. IMPLEMENTATION

Vulnerabilities
We identified those Android applications that are vulnerable and potentially harmful in the following two ways:

1. Applications that do not check for the validity of the SSL certificate before establishing a server connection and ignore SSL errors.
2. Applications that still communicate with the servers using SSL 3.0 protocol.

An attacker on the same network as the Android device may be able to view or modify network traffic that should have been protected by HTTPS. The impact varies based on what the application is doing. Possible outcomes include credential stealing or arbitrary code execution.

Tools and Technology
Eclipse IDE, Java 7, Oracle VM, Genymotion emulator, Lunanode server running Ubuntu 14.04 LTS, and MITMProxy software.

Design
1. We used Lunanode [7], a Cloud Virtualization Platform that provides high performance servers wherein, we installed Oracle Virtual Machine Server [9]. Genymotion [8], a faster Android emulator (Android Virtual Device) is installed on this server. The AVD is

UNIX/ Windows compatible. This is our sandbox environment. The virtual machine will have no other software installed except for the minimum setup required to run the AVD.

2. We then configured the networking of this virtual machine in a way that all the network traffic goes through a special proxy known as the MITMProxy. Since, all the applications are installed on this server, all the traffic of the application passes through MITMProxy.

3. MITMProxy is an open source software that has the ability to intercept, modify, replay and save web traffic over HTTP/ HTTPS connections, while acting as a normal web proxy. This software dumps the web traffic information in to a log file which we used to extract all the HTTPS connections the application made over the internet during run-time.

4. Next, we determined if the remote server contacted by the application supported SSL 3.0 or not by initiating a connection request. This is a standard TCP three-way handshake. Android application communicates with remote servers to fetch updated data as well as provide information for analytical purposes. If the connection request was successful, this means that the server still supports SSL 3.0 which is open to SSL Poodle Vulnerability.

MITMProxy Process

A brief explanation of how the MITMProxy works:

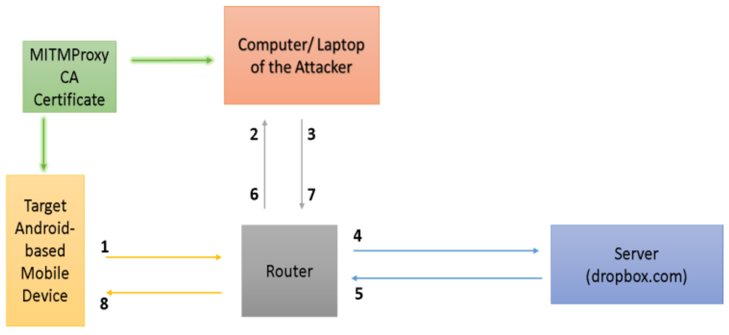

MITMProxy CA Certificate
Computer/ Laptop of the Attacker
2
3
6
7
Target Android-based Mobile Device
1
8
Router
4
5
Server (dropbox.com)

Figure 1: MITMProxy mechanism.

Step 1: A request for dropbox.com (for example) is launched by a vulnerable application from an Android-based mobile device.

Step 2: An attacker (a man-in-the-middle) intercepts this intermediate traffic from the router before it reaches the server. In order for the attacker to be able to sniff this connection, MITMProxy acts as a certificate authority and dynamically generates certificates needed for the attacker. Here, it generates a certificate for dropbox.com.

Step 3: The attacker can tamper the traffic and sends it to the router.

Step 4: The router sends this tampered traffic to the actual dropbox.com server.

Step 5: The server responds with legitimate data to the router.

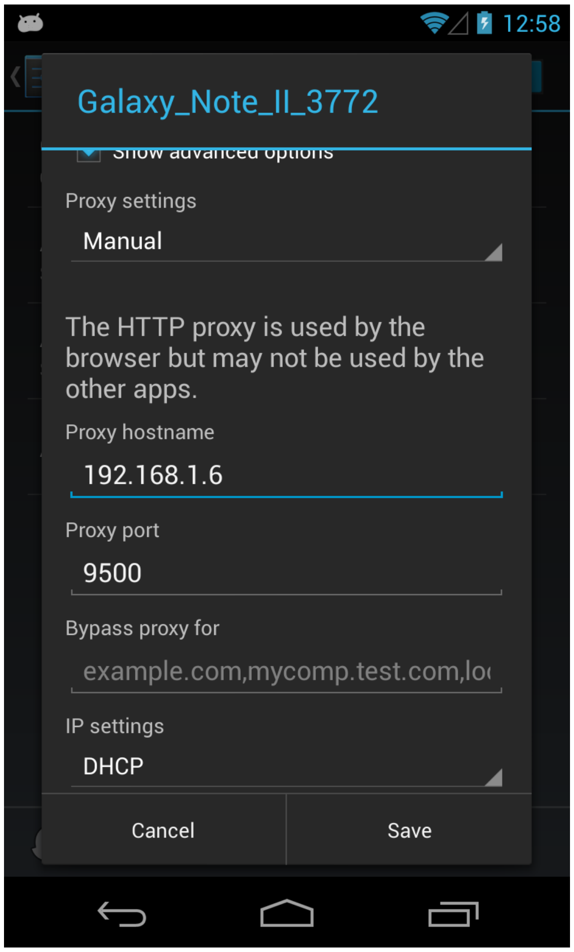
Step 6: This traffic is again intercepted by the attacker suing the certificates provided by MITMProxy.

Step 7: The attacker can read all the information and tamper the traffic. This traffic is sent back to the router.

Step 8: This traffic is then accepted by the target victim device.

Successfully reception of this tampered traffic data by the victim machine signifies that the man-in-the-middle attacks has occurred, thereby concluding that the given application is vulnerable to such attacks.

A device setting pointing to the MITMProxy:



Galaxy_Note_II_3772

Show advanced options

Proxy settings
Manual

The HTTP proxy is used by the browser but may not be used by the other apps.

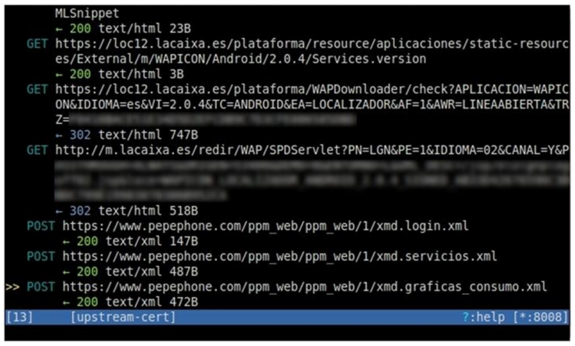
Proxy hostname
192.168.1.6

Proxy port
9500

Bypass proxy for
example.com,mycomp.test.com,lo

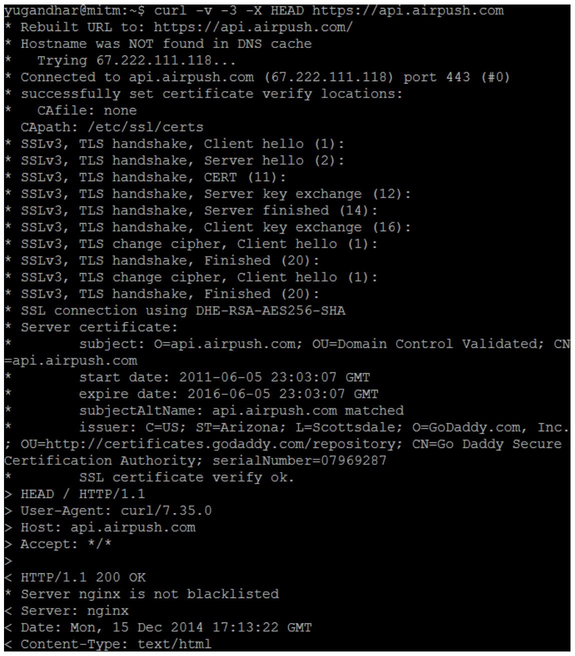
IP settings
DHCP

An example of how MITMProxy displays network traffic:



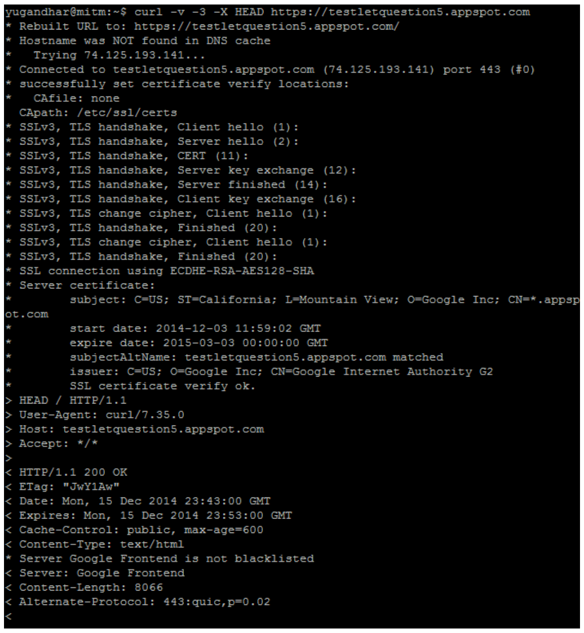
```
        MLSnippet
        ← 200 text/html 23B
   GET https://loc12.lacaixa.es/plataforma/resource/aplicaciones/static-resourc
        es/External/m/WAPICON/Android/2.0.4/Services.version
        ← 200 text/html 3B
   GET https://loc12.lacaixa.es/plataforma/WAPDownloader/check?APLICACION=WAPIC
        ON&IDIOMA=es&VI=2.0.4&TC=ANDROID&EA=LOCALIZADOR&AF=1&AWR=LINEAABIERTA&TR
        Z=[redacted]
        ← 302 text/html 747B
   GET http://m.lacaixa.es/redir/WAP/SPDServlet?PN=LGN&PE=1&IDIOMA=02&CANAL=Y&P
        [redacted]
        ← 302 text/html 518B
   POST https://www.pepephone.com/ppm_web/ppm_web/1/xmd.login.xml
         ← 200 text/xml 147B
   POST https://www.pepephone.com/ppm_web/ppm_web/1/xmd.servicios.xml
         ← 200 text/xml 487B
>> POST https://www.pepephone.com/ppm_web/ppm_web/1/xmd.graficas_consumo.xml
         ← 200 text/xml 472B
[13]     [upstream-cert]                                     ?:help [*:8008]
```

https://api.airpush.com – An analytics company which many apps use to track metrics is vulnerable. We performed an SSLv3 handshake and it went well:



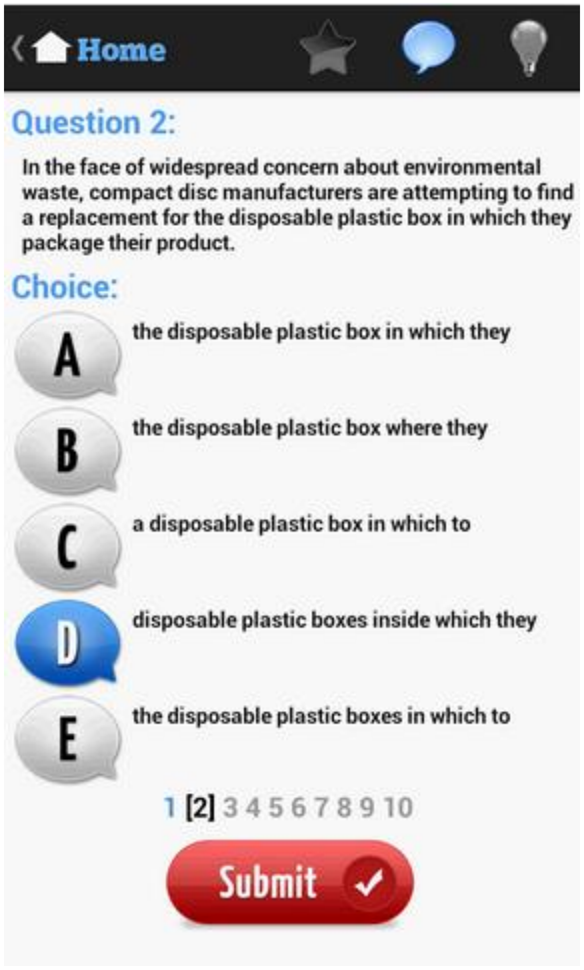
```
yugandhar@mitm:~$ curl -v -3 -X HEAD https://api.airpush.com
* Rebuilt URL to: https://api.airpush.com/
* Hostname was NOT found in DNS cache
*   Trying 67.222.111.118...
* Connected to api.airpush.com (67.222.111.118) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using DHE-RSA-AES256-SHA
* Server certificate:
*        subject: O=api.airpush.com; OU=Domain Control Validated; CN
=api.airpush.com
*        start date: 2011-06-05 23:03:07 GMT
*        expire date: 2016-06-05 23:03:07 GMT
*        subjectAltName: api.airpush.com matched
*        issuer: C=US; ST=Arizona; L=Scottsdale; O=GoDaddy.com, Inc.
; OU=http://certificates.godaddy.com/repository; CN=Go Daddy Secure
Certification Authority; serialNumber=07969287
*        SSL certificate verify ok.
> HEAD / HTTP/1.1
> User-Agent: curl/7.35.0
> Host: api.airpush.com
> Accept: */*
>
< HTTP/1.1 200 OK
* Server nginx is not blacklisted
< Server: nginx
< Date: Mon, 15 Dec 2014 17:13:22 GMT
< Content-Type: text/html
```

A vulnerable app – handshake successful:



```
yugandhar@mitm:~$ curl -v -3 -X HEAD https://testletquestion5.appspot.com
* Rebuilt URL to: https://testletquestion5.appspot.com/
* Hostname was NOT found in DNS cache
*   Trying 74.125.193.141...
* Connected to testletquestion5.appspot.com (74.125.193.141) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using ECDHE-RSA-AES128-SHA
* Server certificate:
*        subject: C=US; ST=California; L=Mountain View; O=Google Inc; CN=*.appsp
ot.com
*        start date: 2014-12-03 11:59:02 GMT
*        expire date: 2015-03-03 00:00:00 GMT
*        subjectAltName: testletquestion5.appspot.com matched
*        issuer: C=US; O=Google Inc; CN=Google Internet Authority G2
*        SSL certificate verify ok.
> HEAD / HTTP/1.1
> User-Agent: curl/7.35.0
> Host: testletquestion5.appspot.com
> Accept: */*
>
< HTTP/1.1 200 OK
< ETag: "JwY1Aw"
< Date: Mon, 15 Dec 2014 23:43:00 GMT
< Expires: Mon, 15 Dec 2014 23:53:00 GMT
< Cache-Control: public, max-age=600
< Content-Type: text/html
* Server Google Frontend is not blacklisted
< Server: Google Frontend
< Content-Length: 8066
< Alternate-Protocol: 443:quic,p=0.02
<
```

The front end of the detected vulnerable application:



Question 2:

In the face of widespread concern about environmental waste, compact disc manufacturers are attempting to find a replacement for the disposable plastic box in which they package their product.

Choice:

A the disposable plastic box in which they

B the disposable plastic box where they

C a disposable plastic box in which to

[✓ D] disposable plastic boxes inside which they

E the disposable plastic boxes in which to

1 [2] 3 4 5 6 7 8 9 10

SSL Detection Mechanism

1. We accessed many applications and performed several functions that required remote server communication. For example GRE Tests application fetched latest quiz questions on initiation of a GRE test.

2. On the Oracle VM, a log of all the incoming/ outgoing traffic was observed. We store this data in a log file.

3. We developed a program that automatically fetched the 'HTTPS' domain names from the log file. For example, the log file has a URL https://api.airpush.com/v2/api.php that an application accessed. The program will automatically fetch the domain name api.airpush.com rom this URL. [11] [12] [13][14]

4. Using '*curl*' [10] command, we initiated communication with HTTPS remote servers that the applications communicated with. The curl command is as follows:

```
bash> curl -v 3 -x HEAD https://
api.airpush.com
```

5. If the remote server completes the handshake, a message is displayed in the command line which means that the application is vulnerable. If the handshake is not completed, it means that the application is secure.

Outcome

The detection identified Android applications that are potentially prone to either of the two above mentioned SSL vulnerabilities.

4. LIMITATIONS

MITMProxy is useful only for sniffing HTTP/ HTTPS connections. Sniffing non-HTTP(S) is not supported. Also, there is a need to automate the process of storing the network traffic details in a log file. Currently, we manually perform this function.

The current approach is dynamic analysis which doesn't help in case of scanning millions of apps. It just isn't feasible to do it this way in terms of costs as well as time.

It is possible to automatically detect whether an application uses one of the three Android SDK packages named for establishing network connections, and to check if any of the methods from those classes are overriden by the application. It is not feasible to automatically determine the intent of the app or the environment the apps are used in.

5. EVALUATION

We tested our application using an Android Virtual Machine (AVD). We installed Google Services and several applications on the AVD. We opened each application and performed several functions that required remote server communication. The communication was monitored and logged to test the applications for potential SSL vulnerability. We tested about 75 android applications from the top categories of various verticals. We could find one malicious app – GRE Tests and most of the other apps belong to top corporations and it is expected of them to secure their servers. Also, most of the servers are using CloudFlare which has automatically started filtering out all the vulnerable requests even before they reach the server. In response to the recent Poodle vulnerability in SSHv3, CloudFlare has disabled SSLv3 across our network by default for all customers.

6. CONCLUSION

When communicating via HTTPS, an application should validate the SSL chain to be sure that the certificate produced by the site was provided by a trusted root certificate authority (CA). Multiple Android applications fail to properly validate SSL certificates. Not properly verifying the server certificate on SSL/TLS may allow apps to connect to an imposter site, while fooling the user into thinking that the user is connected to an intended site. One example of associated risks is that this could expose a user's sensitive data.

Man-in-the-middle attack is one of the oldest and still prevalent network attacks for sniffing traffic. This attack is dangerous because the attacker can firstly, tap into phone connections at an Internet Exchange Point and capture all the traffic passing through that node (HTTP, SMTP and so forth). Secondly, if the attacker also has access to a Certificate Authority (CA) whose certificates are globally trusted, the attacker can use the man-in-the-middle attack to decipher and break any TLS/SSL connection successfully. Use of MITMProxy allowed us to detect Android applications that are possibly vulnerable to such dangerous attacks. In contemporary times, when data and user information, privacy and application/systm security are challenging (due to ever increasing data and many other reasons) and of utmost importance, vulnerable applications that compromise user data and identity are strictly unacceptable. Our approach tried to contribute towards secure Android eco-system by detecting applications that were vulnerable to such attacks and could possibly harm this eco-system.

When alternate means of information is available via a browser, people should use it. The reason behind this is that browsers are meant to be a secure way of communicating with the server. Applications abstract away the background process. Many Android applications are unnecessary in that the content they provide access to is available via other means. For example, while a bank may provide an Android application for accessing its resources, those same resources are usually available by using a web browser. By using a web browser to access those resources, you can help avoid situations where SSL may not be validated.

People should also avoid using untrusted connections. Avoid using untrusted networks, including public WiFi. Using your device on an untrusted network increases the chance of falling victim to a MITM attack.

7. FUTURE IMPROVEMENTS

Future improvement involves automating the logging of network traffic to a log file. We can also write a script to automatically execute the 'curl' command on all HTTPS domain names fetched from the log file. And, an individual SSL communication can be established between the application and the MITMProxy which will be different from another independent communication between the MITMProxy and the remote servers.

The current methodology implemented is dynamic analysis which is not so scalable and economical. It requires us to destroy the VM after each test and rebuild a stock version of the VM so that it can be used for the next set of tests. This doesn't help us in case of scanning millions of apps. So, an efficient way to do this would be to perform static analysis.

A solution we propose to achieve this is to write a de-compiler to break down the packaged android app and scan the code of the app for known patterns of malicious code. In this way we can perform static analysis by looking at the code rather than observing how the app behaves when executed. Also, it covers all the code of the app which may not be triggered when executed during a dynamic analysis workflow.

Also, Amazon's EC2 infrastructure is much more powerful than the basic Lunanode infrastructure. Due to the budgetary constraints, we could not deploy the MITMProxy service on Amazon. We believe the foundation infrastructure plays a significant role in the speed at which the system processes the incoming proxy requests and log all the HTTPS sub-domains.

By Chrome 41 (early 2015), any web site with a certificate that expires in 2016 or later will be shown as untrusted if either:

- The certificate is signed with a SHA-1 algorithm
- One of the certificates in its trust chain is signed with a SHA-1 algorithm (roots are exceptions)

Our system can be fine-tuned to detect those applications that are communicating with web servers having certificated signed with a SHA-1 algorithm by looking at the traffic logs in MITMProxy.

8. GLOSSARY

Transmission Control Protocol (TCP): http://en.wikipedia.org/wiki/Transmission_Control_Protocol
TLS (Transport Layer Security): http://en.wikipedia.org/wiki/Transport_Layer_Security
AVD (Android Virtual Device): http://developer.android.com/tools/help/emulator.html
Man-in-the-middle attack: http://en.wikipedia.org/wiki/Man-in-the-middle_attack

9. REFERENCES

[1] Heartbleed http://en.wikipedia.org/wiki/Heartbleed

[2] A Look Behind the Snapchat Photo Leak Claims. http://bits.blogs.nytimes.com/2014/10/17/a-look-behind-the-snapchat-photo-leak-claims/?_r=0

[3] This POODLE Bites: Exploiting The SSL 3.0 Fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf

[4] An interactive SSL-capable intercepting HTTP proxy for penetration testers and software developers. https://github.com/mitmproxy/mitmproxy

[5] How To: Use mitmproxy to read and modify HTTPS traffic. http://blog.philippheckel.com/2013/07/01/how-to-use-mitmproxy-to-read-and-modify-https-traffic-of-your-phone/
mitmproxy: a man-in-the-middle proxy. http://mitmproxy.org/

[6] SSL 3.0 Protocol Vulnerability and POODLE Attack. https://www.us-cert.gov/ncas/alerts/TA14-290A

[7] Luna Node. https://www.lunanode.com/

[8] Genymotion https://www.genymotion.com/#!/

[9] Oracle VM Server. http://www.oracle.com/us/technologies/virtualization/oraclevm/overview/index.html

[10] cURL. http://en.wikipedia.org/wiki/CURL

[11] www.java2s.com/Code/CSharp/Network/RetrievesthesubdomainfromthespecifiedURL.htm

[12] www.java2s.com/Code/Java/Network-Protocol/GetURLContent.htm

[13] www.java2s.com/Code/Java/Network-Protocol/GetURLParts.htm

[14] blog.houen.net/java-get-url-from-string/

[15]
https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=134807561

[16] http://www.kb.cert.org/vuls/id/582497

[17] http://www.zdnet.com/article/hundreds-of-android-apps-open-to-ssl-linked-intercept-fail/

[18] https://blog.cloudflare.com/sslv3-support-disabled-by-default-due-to-vulnerability/