SSL certificate validation vulnerabilities in non-browser applications

David Soendoro[1], Gu Min[2], Rohit Goyal[3]
Technical University of Denmark

Abstract

SSL(Secure Socket Layer) is a default standard for secure communication on the internet. We demonstrate that the SSL certificate validation is broken in a lot of critical applications, especially non-browser apps. Through a series of experiments, we find out that some applications do not use SSL at all while others implement the server certificate validation incorrectly. As a consequence, it is possible to do man-in-the-middle(MITM) attack and steal important information without much effort. We present our experiment results for two apps from Google playstore.

Introduction

SSL (or TLS) is a transport level protocol, originally deployed in web browsers meant to provide secure client-server communication over an insecure channel. The protocol uses Public Key Infrastructure to authenticate the parties involved in communication and thereby, negotiating a symmetric session key. As a part of the protocol design, authenticating the server is a critical part of the handshake procedure. A server presents its public certificate to the client. The client is supposed to verify the validity of this certificate based on the validity or trustworthiness of the Certification Authority (CA), the common name and the expiry date of the certificate. Most of the browsers implement the validation logic correctly but the non-browser applications which use SSL for secure communications rely on external libraries for implementation. It has been found that most of these implementations are broken and have potential vulnerabilities[1,2] which could be exploited to do MITM attack .

In this poster, we describe the experimental set-up to conduct such an attack. We focus on two type of exploits in certificate validation logic:-

- When the certificate is not signed by a trusted CA
- When there is a mismatch in certificate common name (CN) and the domain accessed by the client

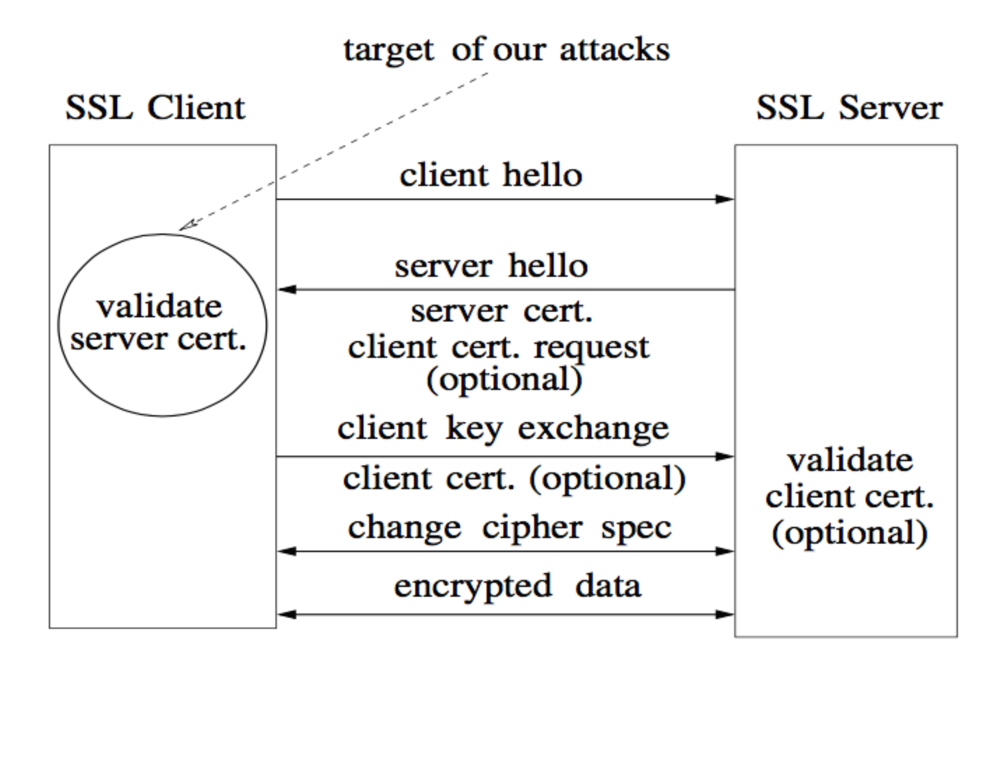


Figure 1: Simplified version of SSL handshake[1]

Methods and Materials

We set up a server side proxy on our system and redirect our mobile traffic to this proxy. Once we enable the proxy for SSL traffic interception, we can see if the app accepts or rejects the SSL server certificate generated by the proxy.

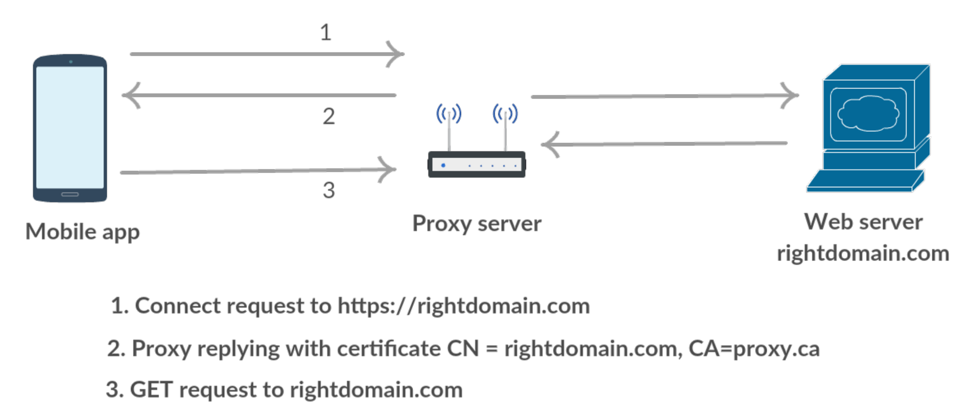


Figure 2: Server certificate with untrusted CA

Another vector of compromise could be a certificate with incorrect common name but signed by a trusted CA. This was tested by DNS poisoning the network, so that the traffic is redirected to our server, which returns its own certificate for all the requests made to external servers.

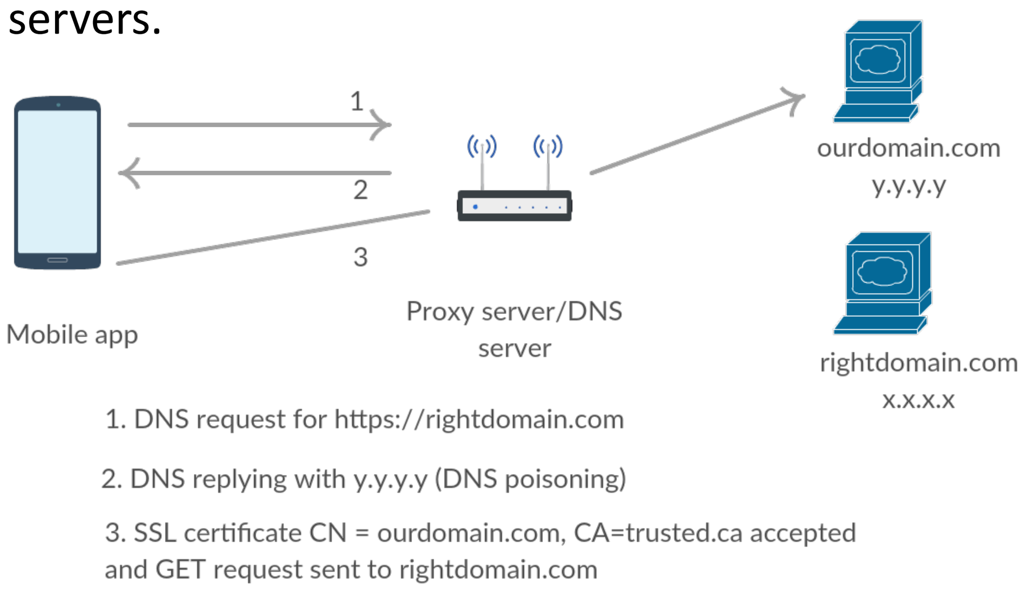


Figure 3: Server certificate with incorrect common name

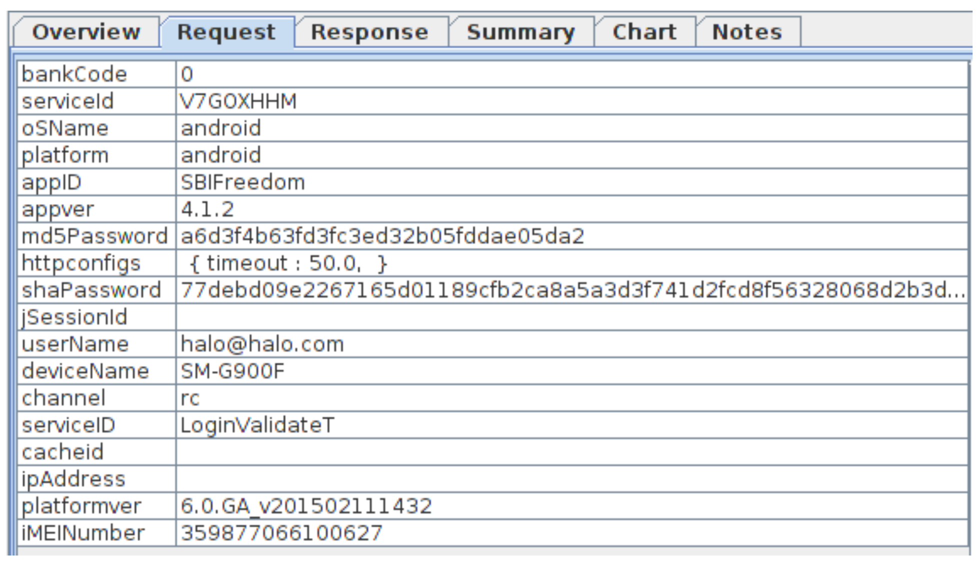
The following are the apparatus we need for this experiment:

- Charles Proxy
- CA and server certificate generated by OpenSSL

Results

We ran our first experiment with a major bank application in Android. The request status was complete and the certificate provided by the proxy was accepted. This fault indicates the app did not verify the validity of the certificate. Consequently, the app started making API requests to the server. This is a potential threat because the intermediate proxy could read all the request parameters as shown in Figure 4.

Second experiment is done with a digital wallet application. The system is exploited using DNS poisoning where the requested domain is translated to 127.0.0.1 (localhost) instead of the real IP address.

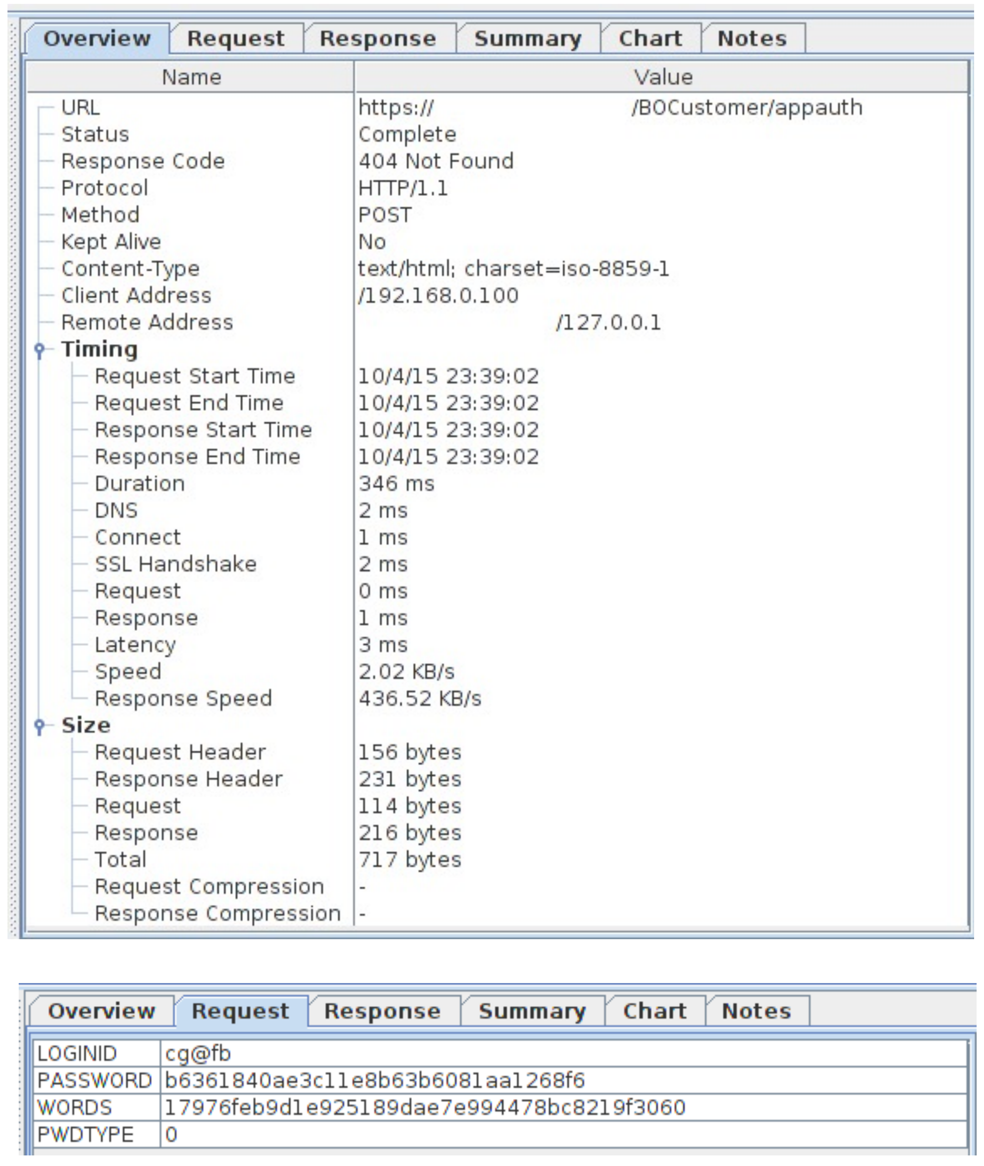


Overview | Request | Response | Summary | Chart | Notes

bankCode	0
serviceId	V7GOXHHM
oSName	android
platform	android
appID	SBIFreedom
appver	4.1.2
md5Password	a6d3f4b63fd3fc3ed32b05fddae05da2
httpconfigs	{ timeout : 50.0, }
shaPassword	77debd09e2267165d01189cfb2ca8a5a3d3f741d2fcd8f56328068d2b3d...
jSessionId	
userName	halo@halo.com
deviceName	SM-G900F
channel	rc
serviceID	LoginValidateT
cacheid	
ipAddress	
platformver	6.0.GA_v201502111432
iMEINumber	359877066100627

Figure 4: Request sent from client through proxy for experiment 1

In this request, the client believes that the certificate it gets from the fake server is a legitimate certificate which actually is its own certificate signed by trusted CA. Therefore the client sends the login request to the fake server. Response 404 (Not Found) is returned since the fake server has no implementation of the request. However, the request gets logged on the fake server as shown in Figure 5.



Overview | Request | Response | Summary | Chart | Notes

Name	Value
URL	https:// /BOCustomer/appauth
Status	Complete
Response Code	404 Not Found
Protocol	HTTP/1.1
Method	POST
Kept Alive	No
Content-Type	text/html; charset=iso-8859-1
Client Address	/192.168.0.100
Remote Address	/127.0.0.1
Timing	
Request Start Time	10/4/15 23:39:02
Request End Time	10/4/15 23:39:02
Response Start Time	10/4/15 23:39:02
Response End Time	10/4/15 23:39:02
Duration	346 ms
DNS	2 ms
Connect	1 ms
SSL Handshake	2 ms
Request	0 ms
Response	1 ms
Latency	3 ms
Speed	2.02 KB/s
Response Speed	436.52 KB/s
Size	
Request Header	156 bytes
Response Header	231 bytes
Request	114 bytes
Response	216 bytes
Total	717 bytes
Request Compression	-
Response Compression	-

Overview | Request | Response | Summary | Chart | Notes

LOGINID	cg@fb
PASSWORD	b6361840ae3c11e8b63b6081aa1268f6
WORDS	17976feb9d1e925189dae7e994478bc8219f3060
PWDTYPE	0

Figure 5: Request sent from client through proxy for experiment 2

Conclusion

It is quite common to see that SSL certificate validation is completely broken in many critical non-browser softwares. It is feasible for intruders to carry out the MITM attack to the system which might result in serious consequences. As the reasons leading to the problem involve application developers along with SSL library developers, it is fundamental for both sides to develop a standardized implementation mechanism to avoid all the possible weaknesses of SSL protocol in the future. Besides, better analysis tools for discovering errors in SSL should be developed in SSL connection stage. More focus should be put on implementing formal verification techniques and programming language support for automatic checking of SSL misusage in applications.

Contact

1. David Soendoro, s146725@student.dtu.dk
2. Gu Min, s146723@student.dtu.dk
3. Rohit Goyal, s146722@student.dtu.dk

References

1. Georgiev, Martin, et al. "The most dangerous code in the world: validating SSL certificates in non-browser software." Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.
2. Conti, Mauro, Nicola Dragoni, and Sebastiano Gottardo. "Mithys: Mind the hand you shake-protecting mobile devices from ssl usage vulnerabilities." Security and Trust Management. Springer Berlin Heidelberg, 2013. 65-81.