Mobile SSL Failures

Tony Trummer
Tushar Dalvi

ABSTRACT

SSL and TLS are the most widely used protocols for securing data on the Internet between mobile devices and their supporting remote servers. Most people are aware of the encryption these protocols provide, but the authenticity aspect is often overlooked.

Our research, focused primarily SSL/TLS usage on the Android operating system and applications, with some additional research on iOS and Windows 8 mobile.

We believe our research has demonstrated that there are systemic issues, largely opaque, to all but the most technical users. These are with the manner in which SSL/TLS is implemented, certificate validation is performed in applications and to a lesser degree, failure to encrypt sensitive data.

While we are not the first to investigate SSL/TLS issues on mobile devices, we demonstrate methods by which implementations, intended to make these protocols more resource efficient, open mobile devices up to the possibility of novel attacks.

Keywords
SSL, TLS, HTTPS, vulnerabilities, Android, iOS, Windows Mobile, mobile application security, certificate validation, SSL session cache

INTRODUCTION

Commonly used in web browsing, SSL and TLS have served as the standard means for securely transferring data across untrusted networks such as the Internet. They are commonly used for general purpose computing, where one or more of the parties, or IP addresses, is not known prior to the connection and the connections are generally transient. Whereas solely symmetric encryption, is commonly deployed between two or more previously known endpoints where shared secrets can be exchanged prior to the initial connection being made.

One of the main drivers for the ubiquitous adoption of SSL/TLS is the fact that it facilitates the ability for two parties, having no established relationship with one another, to rely on presumed indifferent, “trusted” third-party certificate authorities to validate the identities of one, or both of the communicating parties.

Mobile devices, are by their nature transient, connecting to many different networks, even within the same day, including automatically connecting to some carrier configured, default SSIDs for WiFi networks, partly by design, to reduce cellular network traffic usage. We believe this makes them more susceptible to Man-in-the-Middle attacks, compared to most types of common computing platforms.

Our research covers common implementation errors related to SSL/TLS certificate validation, lack of encryption and insecurely implemented features such as SSL session caching.

OVERVIEW OF OUR RESULTS

Our research has uncovered that numerous well-known organizations responsible for publishing many of the most popular mobile applications have failed to properly protect data, transmitted by their applications , from interception and eavesdropping, via man-in-the-middle attacks or passive network sniffing. This data included authentication tokens, passwords, credit card numbers and personally identifiable information (PII).

We believe this is mainly due to simple human error, compounded by inadequate quality assurance and security review practices. Further, we believe the “roll your own” mentality for mobile applications, is a particularly dangerous one, which assumes a level of uniform technical security proficiency, which beyond the current state in most organizations. In a broader sense, we believe this approach also fails to leverage the lessons learned and communal knowledge acquired from decades of browser security vulnerabilities.

In a brief, non-exhaustive examination, we discovered many of the most popular applications failed to either validate that certificates were signed by trusted Certificate Authorities, that the Subject Alternate or Common Names on the received certificates matched the DNS hostname they were attempting to contact, on certificates that were from trusted CAs, or relay any warning indicator to the user.

Since many organizations are accustomed to the development paradigms related to web applications, in many cases, they appear to mistakenly carry over the same thought processes into their mobile development practices. As pointed out by Moxie Marlinspike[6], there is really no reason for mobile applications to use third-party certificate authorities at all. By forgoing their use, many of the vulnerabilities we discovered would be eliminated altogether.

Lastly, we discovered the ability to perform nearly undetectable, albeit limited, MitM attacks against mobile devices leveraging the fact that they implement SSL session caching mechanisms to increase efficiency. This was made possible due to the fact that these mechanisms, by design, only check the certificate validity on the initial connection, but the both the application and operating systems fail to properly invalidate sessions when a certificate has been removed from a device. This presents an opportunity for a would-be attacker. While this vulnerability is not unique to mobile devices, it's exploitation is substantially more likely on a mobile device.

OVERVIEW OF SSL HANDSHAKE

The basic flow of the SSL Handshake is that first the client will send a "Client Hello" message, to which the server will reply with a "Server Hello" message. It is at this point that certificate validation occurs as well as a check of cryptographic parameters. After this, the client sends a secret key encrypted with the server's public key and optionally a client certificate. The remainder is irrelevant to our topic. This should not be taken to be an exhaustive explanation. See http://tools.ietf.org/html/rfc6101 Section 5.5 for additional detail [4]

SSL CERTIFICATE VALIDATION

Upon receiving a server's certificate, a client will verify whether the CA name on the certificate is a trusted CA, or traverse up the hierarchy of trusted CAs until it finds a trusted CA that issued the untrusted CA's certificate. If it fails to identify a trusted root CA in the chain, validation will fail. If it does find a match, it will then verify the certificate's signature, using the public key, to ensure it was actually signed by the private key of the CA.

SSL SESSION CACHING

Following a successful, full SSL handshake, in which certificate validation is performed and cipher suites are exchanged, the server and client can cache a session identifier. This allows subsequent connections to skip the certificate validation and cipher suite exchange, speeding up the process and saving computing resources on both ends. By default, this cached session has a lifetime determined by the server's configuration.

FAILURE TO VALIDATE TRUST CHAIN

Every application in this section failed to validate a certificate received, with the correct hostname, had a trust chain, which led back to a trusted root CA. This would make exploitation simple, as they would accept a certificate, with the correct hostname (in most cases), signed by any CA, trusted or not. In our testing, we used the BurpSuite application's Proxy feature with the "*Generate CA-signed per-host certificates*" setting, without first installing the PortSwigger CA certificate on our devices. Nearly all of the examples below would have allowed for interception and decryption of passwords and/or credit card numbers. In a few cases it may have been limited to authentication tokens, PII and/or allowed for malicious content injection, but for legal reasons, we do not wish to distinguish exactly, unless otherwise noted. Lastly, we have not categorized these applications by operating system, so the vulnerability may have been in the app for Android, iOS or both.

Disabling trusted CA validation is routinely accomplished, in Android apps, by creating a custom X509TrustManger interface that ignores any CertificateException exceptions raised.

Separately, SSL certificate errors can also be disabled in WebViews via the SslErrorHandler class, by invoking the proceed() method[15]. List of vulnerable applications:

1. Hootsuite
2. ClubLocal
3. Pocket
4. OKCupid
5. Sylphone (a Salesforce Partner)
6. Slack
7. Pocket
8. StumbleUpon
9. Uber
10. Starbucks
11. Pizza Hut
12. Walgreens
13. CostCo
14. Staples
15. SouthWest Airlines
16. Sears
17. Macy's
18. Office Depot
19. Kmart
20. iTunes Connect
21. Android's Google Cloud Messaging[1]
22. Microsoft Skype
23. Cisco Webex
24. TimeWarner Cable
25. Piwik
26. Piwik2
27. CNNMoney
28. NewEgg
29. Zappos
30. SecureAuth OTP
31. Authy
32. SafeNet (VPN client)
33. SplashID
34. SonicWALL Mobile Connect
35. Cisco Technical Support
36. Kayako (helpdesk software)
37. Honeywell TC
38. Bing (login)
39. Outlook.com
40. US Bank
41. ADP

1 May have been discovered independently in [1], but Boneh did not recall when asked informally

42. CapitalOne Spark Pay
43. Amazon Kindle

FAILED TO VALIDATE HOSTNAME MATCHED

All of the applications in the following list, failed to validate that the certificate actually matched the hostname they were contacting. This was tested using Burpsuite's Proxy with the *Generate a CA-signed certificate with a specific hostname* enabled, but specifying a mismatched hostname. Any duplications from the previous list, mean either they were vulnerable to both or had different vulnerabilities on different platforms (eg. iOS vs. Android).

Disabling of hostname validation is routinely accomplished, in Android apps, by creating a HostnameVerifier interface which always returns true.

Again, SSL certificate errors can also be disabled in WebViews via the SslErrorHandler class, by invoking the proceed() method[15].

1. Yahoo! Mail
2. Yahoo! Screen (iPad and iPhone)
3. GoDaddy
4. Microsoft Lync 2010 and 2013
5. Slack(twice)
6. Cisco OnPlus (remote access)
7. Serve AMEX
8. MA SolarWinds
9. WesternDigital MyCloud
10. Cisco Webex
11. Intuit Tax Online Accountant
12. Intuit TurboTax Snap Tax
13. American Express BlueBird
14. Ask
15. WesternUnion
16. MedScape (medical information)
17. WordPress
18. myAT&T
19. AT&T U-Verse
20. AT&T Global Network Client
21. Orbitz
22. Huntington Mobile (Bank)
23. AMC (Theaters)
24. Kayak
25. Weibo
26. Angie's List Mobile
27. Oracle Now
28. Dominos Pizza
29. Swivel Secure (OTP)
30. Groupon
31. Citrix Receiver
32. OfficeMax
33. OK Cupid
34. **Vine**
35. **Groupon**
36. **E-Trade**
37. **Uber**
38. **Yahoo! Finance**
39. **Vimeo**
40. **Relate IQ**
41. **Wordpress**
42. **Pinterest**
43. **Google Earth**
44. **Yammer**
45. **Shopify**
46. **Freelancer**
47. **SplashID Safe “Teams”**
48. **Onavo Extend**
49. **Myntra**
50. **Juniper Innov8**
51. **Buffer**
52. **Buffer Daily**
53. **BitCasa**
54. **USBank – Access Online**
55. **Yelp**
56. **SoundCloud**
57. **ADP Dashboard**
58. **Hootsuite**
59. **WesternDigital MyCloud**

The bold items in the list above were all iOS applications, found over one year after our initial round of testing. They all failed to properly validate SAN/CNs matched the DNS hostname they were connecting to. Many of these applications failed because of one of numerous insecure configurations in AFNetworking, a widely used networking library for iOS applications.

Our report to the Yelp security team led directly to an unsafe default configuration being patched in version 2.5.3, when a member of their team reported the issue to the AFNetworking maintainers.

While there were many certificate validation issues in AFNetworking, we believe, ultimately, the burden is on the implementer to ensure their application functions as intended. We have found organizations, aware of the frequency of this type of issue, discovered the unsafe defaults themselves and made the necessary updates to their software to avoid being impacted by the vulnerabilities. We however are unable to cite specific examples of this for reasons outside of our control.

Lastly, following our disclosure to CERT, they issued an advisory entitled “Multiple Android applications fail to properly validate SSL certificates” [19] in an effort to draw greater attention to the issue, citing us as contributors. In a presentation at RSA 2015, Will Dormann, a researcher at CERT claimed to automate the testing process using a tool named *Tapioca* to scan over 1 million applications, finding 23,667 vulnerable. However, without context, these findings are misleading as there may be situations in which improper SSL/TLS usage is non-impactful from a security perspective. Mr. Dormann's presentation included evidence of this as he received correspondence from numerous developers claiming that their applications contained no sensitive data, or otherwise were not impacted by their lack of proper certificate validation. While we applaud this effort, we believe it shows that a manual approach, using dynamic testing, or thorough source code analysis, with context as to the impact of the findings, is at times preferable.

FAILED TO ENCRYPT

The following lists contains apps that failed to encrypt sensitive data, such as credit cards, passwords and/or authentication tokens/cookies.

1. RockBot
   - Passwords
   - Full credit card information
2. Angie's List Business Center
   - Passwords
3. Skype
   - Auth cookies over plain-text HTTP
4. Quora
   - Auth cookies over plain-text HTTP
5. Cisco WebEx
   - Passwords[2]
6. Redbox
   - Millions of installations
   - Passwords
   - Credit cards, including CVV, full PAN and expiration
7. Nearby Live
   - Passwords
8. American Express AXConnect
   - Passwords

In fairness, we'd like to highlight that although, we did not perform as much testing on Windows 8 (mobile), none of the applications we did test, showed this behavior.

We are uncertain at the time of this paper, what the explanation is for this. It could be that Microsoft has more stringent requirements for checking apps into their store, something unique in their coding or build processes or simply chance.

We did also notice on our Windows 8 test device from Verizon, there was a device level toggle to disable certificate validation, which we've incorporated in our recommendations as a good practice to help avoid human error being baked into the code.

While the Windows 8 (mobile) environment seemed to display the best overall results, we'd like to point out that determining what certificates are actually installed on a device would be next to impossible for an ordinary user.

Since certificates can be installed from clicking on an email attachment, this seems to us to be a dangerous combination.

2 This was due to a redirect from HTTPS to HTTP

We'd like to see the list of installed certificates made more accessible.

CONSEQUENCES

While we feel we've enumerated some of the technical security risks of these types of vulnerabilities throughout this paper, we wanted to highlight a recent decision on a case related to this topic. On March 28th, 2014. The FTC released a statement that they had settled a case against Fandango and Credit Karma, where certificate validation failures in their mobile apps was listed as one of the main complaints[14]. Additionally, due to security assurances they made to consumers, regarding their use of SSL, there were allegations stating they had "misrepresented the security of their mobile apps".[3]

The results of this were that both Fandango and Credit Karma are to establish comprehensive security programs and undergo independent security assessments every other year for the next 20 years. While there were no direct financial penalties, in the form of fines, the costs of additional oversight and legal fees will undoubtedly be significant and far greater than any conceivable benefit gained from allowing certificate validation to be disabled in development.

SSL/TLS SESSION CACHING

During our research, we noticed that after rebooting an Android device, via either the "Restart" or "Power Off" options and subsequently powering it back on, we were repeatedly still able to see encrypted traffic from some applications, such as Google Maps, being intercepted by our proxy, despite not being vulnerable to attacks mentioned elsewhere in this paper. Unlike in the previous scenarios, the proxy's CA certificate had been installed and trusted when a connection was initially made from the apps, but subsequently removed prior to reboot.

Upon discovering this, we tested the same applications on iOS and they exhibited the same behavior when a previously installed CA certificate was removed, but only up to the point that the device was rebooted.

This implied that the tested applications must have been using file based storage on Android, but not on iOS, which we have since confirmed[2][7].

Digging deeper and with help from the Android security team, it was determined this was due to SSL/TLS session caching. Android has a class named SSLSessionCache which [14]implements a "*File-based cache...which can span executions of the application*"[3]. This also means that it can persist when there is no power to the device.

3 http://www.ftc.gov/news-events/press-releases/2014/03/fandango-credit-karma-settle-ftc-charges-they-deceived-consumers

Because, in both cases, there is no visible indication to the user that there was a previously installed certificate, on most Android and iOS devices, an attacker, with the ability to install and remove certificates on the device, could instantiate a network connection with any application using this feature and, in effect, create a staged MitM attack.

Due to the persistence "feature" in Android, this could potentially allow installation of "invisible" certificates anywhere in the supply chain, possibly in the same manner that malicious Netflix apps appeared earlier this year on brand new devices[5].

At the time of this paper, we are currently unaware of any reasonable programmatic means to install certificates via a malicious application, on non-rooted devices, that does not require user interaction and therefore social engineering. Remote certificate installation without user interaction is an area of active research for us. Additionally, it is assumed that any remote social engineering attacks would work regardless of the vulnerabilities outlined in this paper, thereby negating their relevance to this topic.

Despite the fact that physical access is currently thought to be required, we feel this is a plausible attack, specifically, because physical control of mobile devices can be harder to maintain when generally compared to fixed assets.

With regards to screen-locking being a further deterrent to physical attacks, in a recent study[13] Google found that 52% of users used a "simple slide or gesture" to unlock their devices. Even if reasonably complex screen-lock passwords or drawings are implemented, researchers have shown these can be determined with 68% accuracy[10]. Finally, there have been numerous bypasses historically, including in the last year[8][9].

We believe there are several plausible scenarios where an individual may be compelled or social engineered into relinquishing control of their device, if even for a short time, and having a session staged on it. Examples could include having your device seized while being detained by law enforcement or governmental agencies, losing the device or having it stolen, only to have it "miraculously" returned later or if the device is purchased second-hand. Additionally, a session could be staged anytime prior to a device being first given to a user, either by their cellular carrier or IT department.

We have confirmed the ability for persisting sessions in excess of 24 hours and are currently researching the feasibility of increasing session cache timeouts, to arbitrarily high values, to create an enduring MitM situation. A duration of about 2 years, is assumed sufficient to persist for any individual owner, in most cases. This would presumably align with cell phone contract renewals and is well within the maximum DNS TTL[11].

Additionally, we believe it is reasonable to assume that because the certificate validation only occurs on the initial handshake any of these connections, this ability to endlessly cache sessions would allow for certificates that are revoked or expired to remain active indefinitely. This is an area of ongoing research for us.

We believe this leaves maintaining the MitM position as the main obstacle to wide-spread abuse. It is assumed this would require some means to consistently poison DNS responses, by modify the hosts file, changing the DNS server settings or possibly configuring a VPN on the device. If the device were rooted, this obstacle is easily overcome, but is believed to be non-trivial on non-rooted devices, provided the adversary is not a governmental entity, an ISP or in the supply chain. As an example, in 2013 Nokia was found to be performing a massive MitM on their customer's traffic.[4]

Some interesting possibilities, for the most determined of attackers, could be using a drone similar to the "Snoopy"[12] drone, to follow the victim pretending to be a trusted SSID or strategically placed hot-spots in areas they are likely to use their apps.

A recently published article claims that researchers have discovered novel attacks that make cracking WPA2 security[17] possible. If accurate, this may make obtaining or maintaining MitM position significantly easier.

RECOMMENDATIONS

For Organizations

Train development, quality assurance and security staff on the importance of SSL certificate validation and how to test for it. Implement policies to prohibit disabling these validations in code at any point in the release process. Invest in moving away from the use of public CAs in mobile applications or at a minimum implement certificate pinning.

For Developers

Remove the need for certificate authorities altogether by locally verifying the received certificate. If that is not possible, consider implementing certificate pinning and add your test servers' certificates to the list of trusted certificates, rather than disabling certificate validation globally. If none of these options work for your organization, install a trusted CA certificate, from your development environment, on your development device or emulator, which only take a few seconds. Weigh the risk/reward scenarios cautiously before implementing any SSL session caching functionality in client apps, especially if they are persistent across reboots.

4 http://www.ftc.gov/news-events/press-releases/2014/03/fandango-credit-karma-settle-ftc-charges-they-deceived-consumers

For Security and QA Testers

Ensure certficate validation is included in part of your pre-release testing. Familiarize yourself with how this is defeated in code for the platforms you support and perform pre-release code reviews for this specific issue.

For the Public

Never trust that an application is as safe as a browser, until such a time as there are mandatory visual indicators that warn of potential issues, similar to the way the padlock works in a browser. Uninstall any pre-installed applications you can, when acquiring a new device. Disable all automatic connections to WiFi networks, especially those that are easily guessed by attackers.

For SmartPhone OS developers

Force a visual indicator, similar to the browser padlock on all secure connections. Remove the ability to disable certificate validation from the developer's hands. Alternatively, make the ability to ignore certificates a toggle on the device and/or emulator, rather than in the code. Clear all SSL/TLS session caches when a certificate is removed from a device, or force a reboot in iOS. Force an app to get permission from the user before allowing it to disable certificate validation.

App store owners

Perform static code analysis for all submitted applications to ensure they do not have certificate validation disabled prior to releasing them. Refuse to accept any that do not.

RELATED WORK

Independent of this work, Georgiev, et al. [1], provided an in-depth look at certificate validation issues related to non-browser software, including mobile applications, but focusing on mainly mobile banking apps. By contrast, our work includes a survey of the common nature of this issue, across numerous types of apps, citing numerous specific examples. Additionally, our research focuses exclusively on mobile applications and operating systems, not a broader discussion. Finally, their paper makes no mention of attacks against session caching or failing to encrypt data all together.

Independent of this work, H. Shacham, et al. [2] , make reference to potential weaknesses for “cached” certificates, but appear to assume a compromise of PKI as a whole is necessary and neglect to consider client-side attacks.

Independent of this work, IOActive published a blog post describing very similar findings, but appears to focus on mobile banking apps on iPad and iPhone. In contrast, we did not focus on any apps in particular, other than the fact that their needed to be a reasonable expectation that sensitive data was transmitted by them. Additionally, the majority of our research was focused on Android. Additionally, they did not mention any findings around session caching.

CONCLUSION

The main takeaway from this paper would be that organizations need to ensure they are not actively defeating client-side security mechanisms in mobile applications, even during the development process.

As more and more Internet traffic moves towards mobile platforms, organizations need to re-think the way mobile applications are developed, deployed and tested. While mobile applications may commonly implement HTTP, they are not traditional web applications and present unique security issues.

Following the advice from Moxie Marlinspike[6] by either locally validating the certificate or implementing certificate pinning, would eliminate most of these certificate related vulnerabilities and has the added benefits of potentially limiting exposure to compromised certificate authorities.

REFERENCES

[1] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, V. Shmatikov. *The Most Dangerous Code in the World*. `https://crypto.stanford.edu/~dabo/pubs/abstracts/ssl-client-bugs.html, 2012`

[2] H. Shacham, D. Boneh, E. Rescorla.
Client Side Caching for TLS.
`http://crypto.stanford.edu/~dabo/abstracts/fasttrack.html`

[3] How SSLSessionCache class caches sessions
`http://developer.android.com/reference/android/net/SSLSessionCache.html`

[4] How SSL Handshake works
`http://tools.ietf.org/html/rfc6101`

[5] Pre-installed malware turns up on new phones
`http://www.pcworld.com/article/2104760/preinstalled-malware-turns-up-on-new-phones.html`

[6] Your app shouldn't suffer SSL's problems
`http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/`

[7] TLS Session Cache on iOS
`https://developer.apple.com/library/ios/qa/qa1727/_index.html`

[8] New Samsung flaw allows 'total bypass' of Android lock screen `http://www.zdnet.com/new-samsung-flaw-allows-total-bypass-of-android-lock-screen-7000012888/`

[9] Another lock screen bypass reported in iOS 7

http://www.zdnet.com/another-lock-screen-bypass-reported-in-ios-7-7000021462/

[10] Smudge attack
http://en.wikipedia.org/wiki/Smudge_attack

[11] Clarifications to the DNS Specification
http://www.ietf.org/rfc/rfc2181.txt

[12] 'Snoopy' drone which can can hack your phone and steal all your data let loose in London
http://www.mirror.co.uk/news/technology-science/technology/snoopy-drone-can-hack-your-3280351

[13] Over half of Android users fail to lock their phones
http://www.net-security.org/secworld.php?id=16577

[14] Fandango, Credit Karma Settle FTC Charges that They Decieved Consumers By Failing to Securely Transmit Sensitive Person Information
http://www.ftc.gov/news-events/press-releases/2014/03/fandango-credit-karma-settle-ftc-charges-they-deceived-consumers

[15] SslErrorHandler
http://developer.android.com/reference/android/webkit/SslErrorHandler.html

[17] WPA2 wireless security cracked
http://sciencespot.co.uk/wpa2-wireless-security-cracked.html

[18] Nokia 'hijacks' mobile browser traffic, decrypts HTTPS data
http://www.zdnet.com/nokia-hijacks-mobile-browser-traffic-decrypts-https-data-7000009655/

[19] Multiple Android applications fail to properly validate SSL certificates
http://www.kb.cert.org/vuls/id/582497