Best Practice Guide (SSL Implementation) for Mobile App Development

流動應用程式 (SSL 實施) 最佳行事指引

Jointly published by

Hong Kong Computer Emergency Response Team Coordination Centre (HKCERT)
［香港電腦保安事故協調中心］
https://www.hkcert.org

Professional Information Security Association (PISA)
［專業資訊保安協會］
http://www.pisa.org.hk

Publication version 1.0 September, 2015

Content

Abstract

Mobile platform is increasingly become a choice for delivering services. As more sensitive data and transaction data will be transported on mobile communication channels, the security risks associated with untrusted communication, such as public Wi-Fi have to be addressed, for example, fraudster can set up a fake Wi-Fi access point and fake Secure Sockets Layer (SSL) certificates to conduct man-in-the-middle (MITM) attack[1] to capture sensitive data.

Secure Sockets Layer / Transport Layer Security (SSL/TLS) has been widely used for authentication and encryption. However, if it is implemented via mobile apps, users have not got the same transparency of SSL as in the browser where visual alerts can be given (a colour padlock icon indicator shown in the address bar). The quality of SSL/TLS implementation in mobile app is thus crucial to detect and deny MITM attacks.

This document mentions common practices which help mobile application developers to handle SSL connection with appropriate ways to provide secure channel between mobile app and server and also prevent from MITM attack.

Who should read

This document targets specifically the following parties:

1. Mobile app owners who would implement a mobile app service which involves transmission of sensitive data, such as personal data, credential, and payment information.

2. App developers who provide codes for the mobile apps, especially on iOS and Android platforms, which make HTTPS connection to the servers on SSL/TLS protocol.

When to read

This document should be read:

1. before mobile app owners and app developers start planning mobile app development project;

2. during the review of the SSL/TLS implementation and verification of coding for the mobile app developers.

[1] MITM attack, Wikipedia https://en.wikipedia.org/wiki/Man-in-the-middle_attack

Relevance of the Personal Data (Privacy) Ordinance to the Security of SSL Implementation in Mobile Application Development

Data Protection Principles - Security ("DPP4")[2] under the Personal Data (Privacy) Ordinance requires a data user to take all reasonably practicable steps to implement security precautions, the level of which should be commensurate with the seriousness of the potential harm that could result from a data breach.

The "Personal data privacy protection: what mobile apps developers and their clients should know"[3] stated that data user should consider the use of technological safeguards, including encrypting personal data being transmitted to prevent unauthorized interception or access.

For the effective protection of data via encryption technological, the data user and the mobile app developer should answer to the following questions:

1. Is the transmission of sensitive data properly protected by encryption?

2. Is the strength of encryption technology proportional to the security risks associated?

   For critical services such as financial application, cybercriminals have the incentive to use more advanced attacks to circumvent normal SSL certificate validation. They might trick the user to install a fake certificate on to a mobile device. In this case, mobile app owner and developer should consider adopting more advanced technology such as Certificate Pinning[4] to combat against such attack.

3. Is the encryption properly implemented so that it cannot be easily circumvented?

   In the mobile app development, SSL/TLS encryption protocol is commonly used to encrypt sensitive data during transmission. There are many ways that a faulty implementation can give opportunity for attackers. For example, if the mobile app does not validate digital certificate for expiry date, the proper signing certificate authority, and use of strong state-of-art strong encryption algorithm, attacker can use an expired certificate, a fake certificate, or a known attack to force the use of a low end encryption algorithm.

To ensure proper protection of sensitive data in transmission, mobile app owner should put down the requirements on the selected implementation of encryption and validation of digital

[2] Data Protection Principles, Office of the Privacy Commissioner for Personal Data https://www.pcpd.org.hk/english/data_privacy_law/ordinance_at_a_Glance/ordinance.html#4
[3] Personal data privacy protection: what mobile apps developers and their clients should know, Office of the Privacy Commissioner for Personal Data https://www.pcpd.org.hk/english/resources_centre/publications/files/apps_developers_e.pdf
[4] Certificate Pinning, Wikipedia https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning

certificate, and verification of implementation into the tender or quotation specification of the mobile app development project.

For broader consideration on personal data privacy during the design and development of mobile apps, you may refer to the "Best Practice Guide for Mobile App Development"[5] provided by Office of the Privacy Commissioner for Personal Data.

Possible reasons of invalidation of SSL

Older Mobile Platform

For the Android 2.2 or below version, the SSL support contains some issues on Server Name Indication (SNI)[6] and Multiple Chain[7]. It causes SSL connection become invalid, due to the rejection of missing intermediate certificate authority (CA) certificate in the older Android system.

Missing Certificate Authority (CA) Certificate

The SSL connections of some websites become invalid, because the signed CA certificate may be missing in the default CA list of the mobile platform.

Self-signed Certificate

This situation always occurs during development. A self-signed certificate is commonly used in the testing environment for testing purpose. Companies and developers are not willing to pay a certificate for internal testing.

Expired Certificate

Expired certificate is usually a management issue. The administrator of the production website has not tracked and updated the digital certificate from time to time. An expired certificate will be treated as untrusted certificate.

Man-in-the-Middle (MITM)

When attacker conducts a MITM attack, the traffic will be intercepted or redirected. In this situation, the attacker's self-signed certificate will replace the legitimate certificate. This makes the SSL connection become invalid.

The following figure shows the situation of MITM attack. Attacker can setup a fake Wi-Fi access point. When mobile user connects to the Internet via the access point, MITM intends to intercept

[5] Best Practice Guide for Mobile App Development, Office of Privacy Commissioner for Personal Data https://www.pcpd.org.hk/english/resources_centre/publications/files/Mobileapp_guide_e.pdf
[6] Issues on SNI, Google https://code.google.com/p/android/issues/detail?id=12908
[7] Issues on Multiple Chain, Google https://code.google.com/p/android/issues/detail?id=26542

the traffic. If the mobile client (mobile app) does not handle the validation of SSL certificate, an invalid SSL connection will be made and the MITM can intercept the traffic successfully.

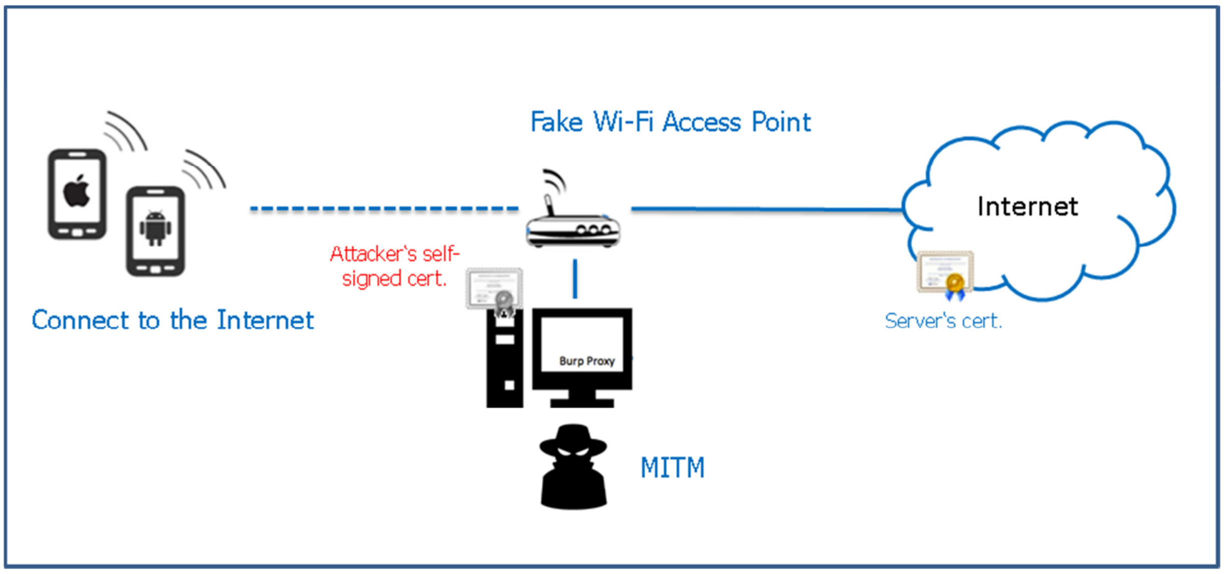


Fig 1. MITM attack

Secure SSL/TLS connection checklist

To figure out if your mobile app has made a secure connection, mobile app developers should check the following items:

1. Does your mobile app connect to one or more servers in proper encryption?
2. Are the SSL certificates in date?
3. Are the SSL certificates signed by trusted CA providers or self signed?
4. Does the SSL use high enough cipher strengths?
5. Does your mobile app accept user-accepted certificates as authorities?

By the above questions, mobile app developers should be guided to consider the implementation of SSL/TLS connection. Firstly, encrypted connection is necessary. Then, a valid SSL connection is established. Strong cipher suites are also required. Finally, an advanced MITM resistant protection, such as Certificate Pinning[8], can be applied to enhance the security of end-to-end connection.

[8] Certificate Pinning, Wikipedia https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning

Best Practices

To prevent from the MITM attack and to build a secure connection, a correct SSL implementation is necessary. About the network security on mobile, OWASP had issued guidelines[9] to prevent insufficient transport layer protection. This document will highlight the best practices on the SSL implementation for mobile developers.

General Best Practices

- Assumption
  - Assume that the network layer is not secure and is susceptible to eavesdropping.

- Dos
  - Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service.
  - Use strong, industry standard cipher suites with appropriate key lengths, like SHA256.
  - Use certificates signed by a well-known and trusted CA provider.
  - Always require SSL chain verification. Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.
  - Alert users through the UI if the mobile app detects an invalid certificate.
  - Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser/webkit. Avoid mixed SSL sessions as they may expose the user's session ID.
  - If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation.
  - Regular review of security risk and protection level on the mobile app and its platform.

- Don'ts
  - Never allow self-signed certificates, and consider certificate pinning for security conscious applications.
  - Do not send sensitive data over alternate insecure channels (e.g, SMS, MMS, or notifications).

[9] Insufficient Transport Layer Protection, OWASP
https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

iOS Specific Best Practices

- Do not add code to bypass these defaults to accommodate development hurdles
- Ensure that certificates are valid and fail closed.
- When using `CFNetwork`, consider using the Secure Transport API to designate trusted client certificates. To make secure connections, `NSStream` is used instead of using sockets directly. In almost all situations, `NSStreamSocketSecurityLevelNegotiatedSSL` should be used. If you need to work around compatibility bugs, you can also specify a more specific protocol, such as `NSStreamSocketSecurityLevelTLSv1`.
- After development, ensure all `NSURL` calls (or wrappers of `NSURL`) do not allow self signed or invalid certificates such as the `NSURL` class method `setAllowsAnyHTTPSCertificate`.
- Consider using certificate pinning by doing the following: export your certificate, include it in your app bundle, and anchor it to your trust object. Using the `NSURL` method `connection:willSendRequestForAuthenticationChallenge:` will now accept your cert.

Android Specific Best Practices

- Remove all code after the development cycle that may allow the application to accept all certificates such as `org.apache.http.conn.ssl.AllowAllHostnameVerifier` or `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER`. These are equivalent to trusting all certificates.
- If using a class which extends `SSLSocketFactory`, make sure `checkServerTrusted` method is properly implemented so that server certificate is correctly checked.

Recommendations on Practical Programming

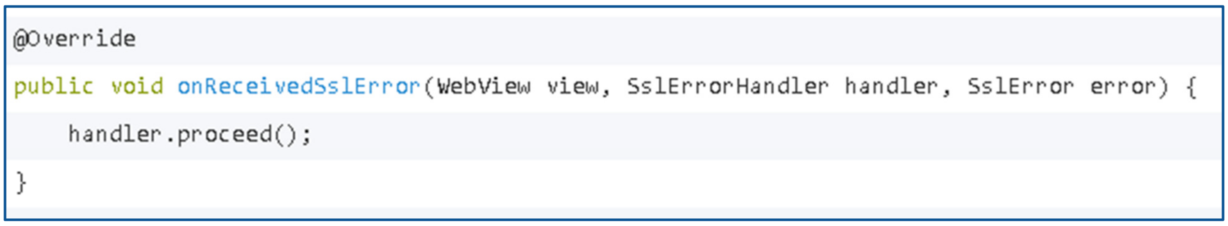
In general, mobile app developer can implement a secure channel SSL using standard library provided by official Software Development Kit (SDK). Sometimes, mobile developer would bypass the validation error during testing; or mobile developer mishandle the validation of certificate in the program.

The following scenarios[10] are commonly found when a developer works on the SSL connection in Android and iOS application.

[10] 手機應用程式開發上被忽略的 SSL 處理, DevCore http://devco.re/blog/2014/08/15/ssl-mishandling-on-mobile-app-development/

Android Scenario 1: Mishandle SSL error in onReceivedSslError function

If there is a SSL error when WebView connects to a HTTPS server, the function `onReceivedSslError` will be triggered. Developer can decide the establishment of the connection by using `handler.proceed()` or `handler.cancel()`. The function `handler.proceed()` will be only used to bypass the error and process the connection in testing environment. `handler.cancel()` should be used to deny the connection if SSL error occurs.



```
@Override
public void onReceivedSslError(WebView view, SslErrorHandler handler, SslError error) {
    handler.proceed();
}
```

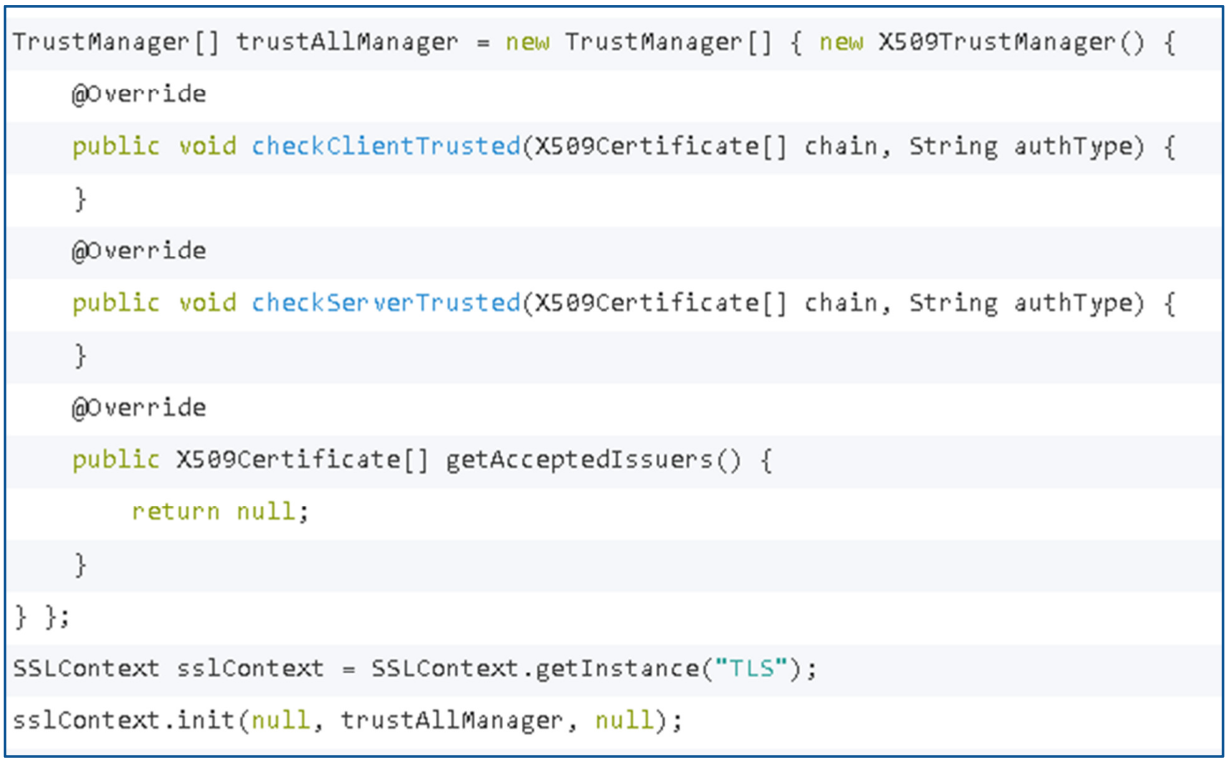
Fig 2. Mishandling in `onReceiveSslError` function

Android Scenario 2: Mishandle validation in a custom TrustManager

With a custom `TrustManager`, developer may not implement the function of `checkServerTrusted`, then the checking of server's certificate will be bypassed. An invalid SSL connection could be established. Google provided an example code[11] to takes a specific CA to create a KeyStore, which is then used to create and initialize a `TrustManager`. In addition, OWASP provided the sample program using custom `TrustManager` to implement certificate pinning[12].

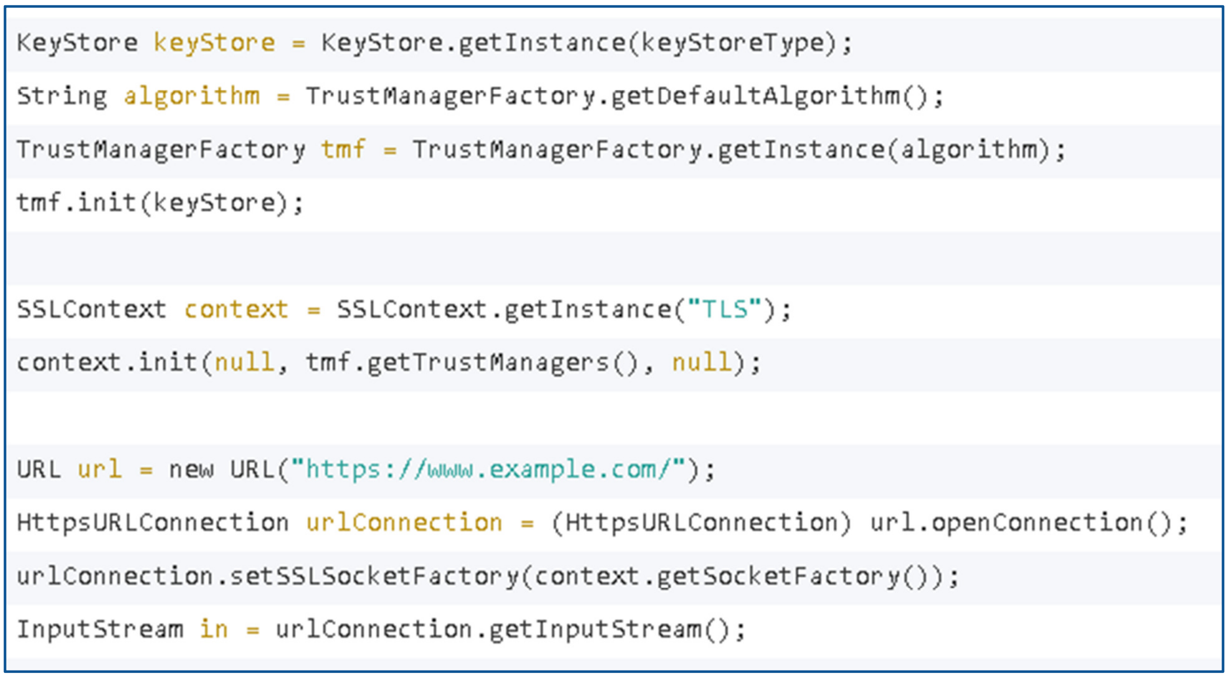
[11] Security with HTTPS and SSL, Android Developer https://developer.android.com/training/articles/security-ssl.html#UnknownCa

[12] Certificate Pinning for Android, OWASP https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#Android



```java
TrustManager[] trustAllManager = new TrustManager[] { new X509TrustManager() {
    @Override
    public void checkClientTrusted(X509Certificate[] chain, String authType) {
    }
    @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType) {
    }
    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return null;
    }
} };
SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, trustAllManager, null);
```

Fig 3. Missing implementation in `checkServerTrusted` function

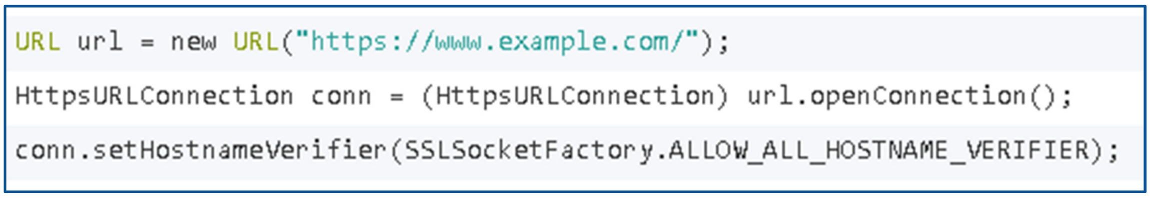


```java
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
String algorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(algorithm);
tmf.init(keyStore);

SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

URL url = new URL("https://www.example.com/");
HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();
```

Fig 4. Example code using `KeyStore` to create a `TrustManager`

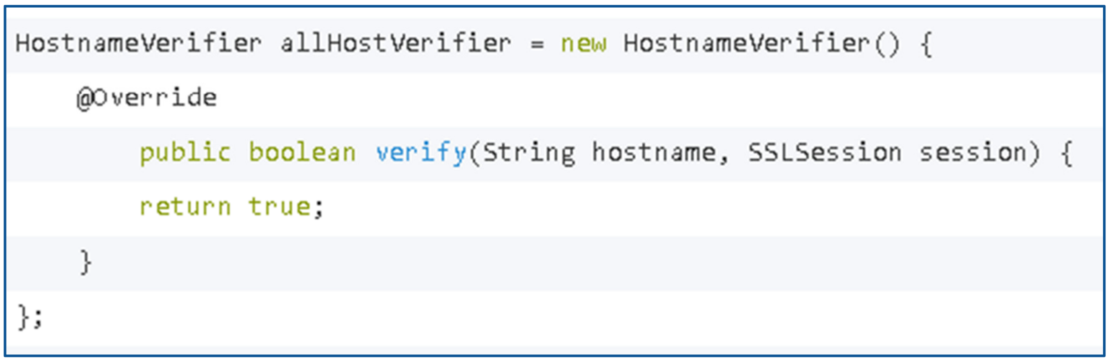
Android Scenario 3: Allow any hostname in the verification

Developer may ignore the hostname verification in the SSL session by setting `ALLOW_ALL_HOSTNAME_VERIFIER` or `return true` in the function. This allows an invalid SSL connection established when the hostname is different from the certificate. It is not a good practice. A recommended practice is apply `DefaultHostnameVerifier` or apply `StrictHostnameVerifier` in the hostname verification.



```
URL url = new URL("https://www.example.com/");
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
```

OR

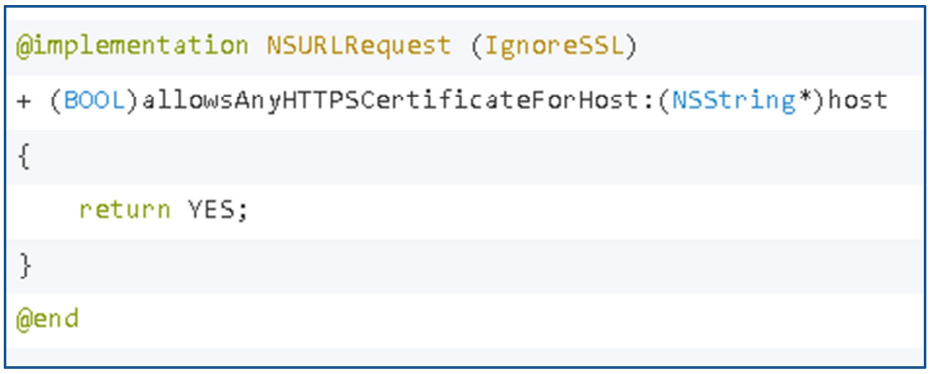


```
HostnameVerifier allHostVerifier = new HostnameVerifier() {
    @Override
        public boolean verify(String hostname, SSLSession session) {
        return true;
    }
};
```

Fig 5. Mishandling verification to allow all hostname

iOS Scenario 1: Allow any HTTPS certificate

The function `allowsAnyHTTPSCertificateForHost` is not allowed and not passed in the App Store review. To prevent mis-deploy in the development environment, the program is better to be placed in the block of "`#if DEBUG`" and "`#endif`". So that, the function is only executed in debug mode.

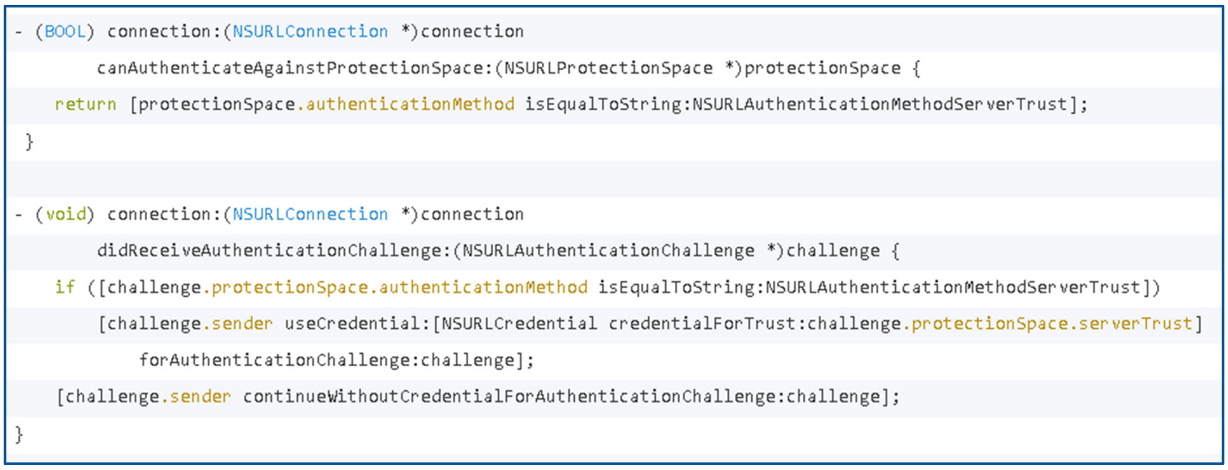


```
@implementation NSURLRequest (IgnoreSSL)
+ (BOOL)allowsAnyHTTPSCertificateForHost:(NSString*)host
{
    return YES;
}
@end
```

Fig 6. Better to be placed in the block of "#if DEBUG" and "#endif"

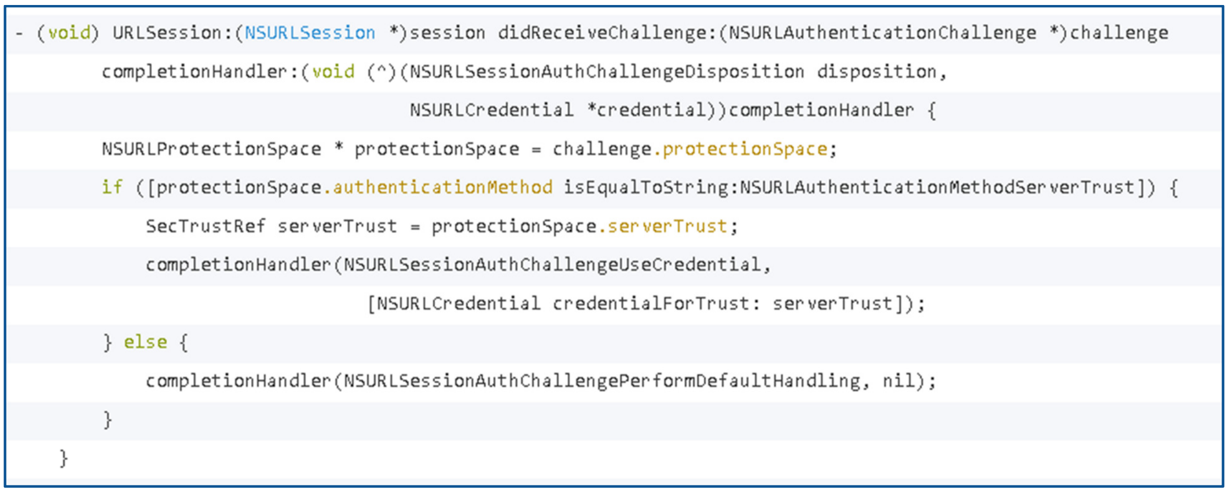
iOS Scenario 2: Mishandle validation in NSURLConnection and NSURLSession

Developer may not implement the checking of certificate in the function `NSURLConnection` and `NSURLSession`, that allows any invalid connection established in the application. This is not a good practice on SSL implementation. OWASP provided the sample program when implement certificate pinning[13], which shows the verification of certificate in the iOS development.



```
- (BOOL) connection:(NSURLConnection *)connection
        canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)protectionSpace {
    return [protectionSpace.authenticationMethod isEqualToString:NSURLAuthenticationMethodServerTrust];
 }

- (void) connection:(NSURLConnection *)connection
        didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    if ([challenge.protectionSpace.authenticationMethod isEqualToString:NSURLAuthenticationMethodServerTrust])
        [challenge.sender useCredential:[NSURLCredential credentialForTrust:challenge.protectionSpace.serverTrust]
            forAuthenticationChallenge:challenge];
    [challenge.sender continueWithoutCredentialForAuthenticationChallenge:challenge];
}
```

Fig 7. No validation of SSL certificate in `NSURLConnection`



```
- (void) URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
        completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition,
                                    NSURLCredential *credential))completionHandler {
        NSURLProtectionSpace * protectionSpace = challenge.protectionSpace;
        if ([protectionSpace.authenticationMethod isEqualToString:NSURLAuthenticationMethodServerTrust]) {
            SecTrustRef serverTrust = protectionSpace.serverTrust;
            completionHandler(NSURLSessionAuthChallengeUseCredential,
                              [NSURLCredential credentialForTrust: serverTrust]);
        } else {
            completionHandler(NSURLSessionAuthChallengePerformDefaultHandling, nil);
        }
    }
```

Fig 8. No validation of SSL certificate in `NSURLSession`

[13] Certificate Pinning on iOS, OWASP
https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#iOS

Certificate Pinning

Attackers might find ways to circumvent digital certificate validation by implanting a fake digital certificate into the certificate store of the mobile phone through mobile malware or social engineering techniques.

To prevent such advanced MITM attack, Certificate Pinning is good solutions to ensure the connection secure with a specific certificate. However, certificate pinning cannot be implemented in WebView connection yet. To ensure the connection is secure, developers should not rashly copy the sample code on the Internet and bypass the important validation functions. Strict HTTPS implementation code should be followed. Apple iOS[14] and Google Android[15] provides network security suggestion for the mobile apps developers. Besides, OWASP issued Top 10 mobile risks, which included network security recommendation.

Developers and companies have responsibility to protect the transferring data and provide a secure environment against MITM attack to mobile users.

Conclusion

SSL encryption is a key technological safeguard to protect sensitive data from unauthorized interception or access in transmission. In mobile apps, since there is no visual cue as browsers to warn the users on invalid SSL connection, the responsibility of mobile developer in the proper implementation of SSL should be highlighted.

This document has described the best practices with recommendations in practical programming in Android and iOS platforms. We hope that this is useful for mobile developers to develop good and secure code for the benefits of the community.

[14] Using Networking Securely, iOS Developer Library
https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/SecureNetworking/SecureNetworking.html

[15] Security with HTTPS and SSL, Android Developer
https://developer.android.com/training/articles/security-ssl.html