Building Secure Mobile Applications

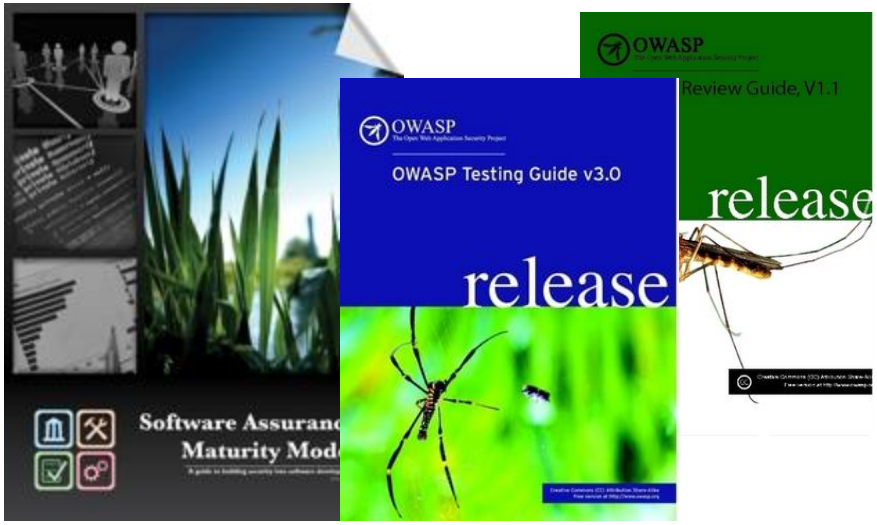
Eoin Keary
OWASP Board Member
CTO BCC Risk Advisory
www.bccriskadvisory.com

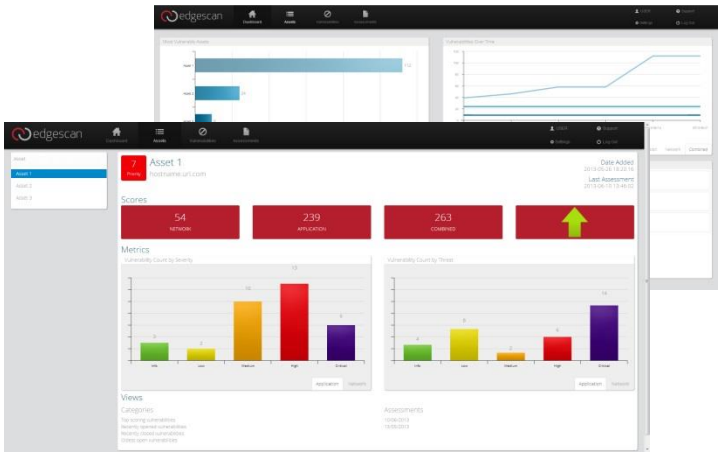
RISK ADVISORY

@eoinkeary
eoin@bccriskadvisory.com

Eoin Keary

- CTO BCCRISKADVISORY.COM
- OWASP GLOBAL BOARD MEMBER
- OWASP Project Lead/Contributor





edgescan

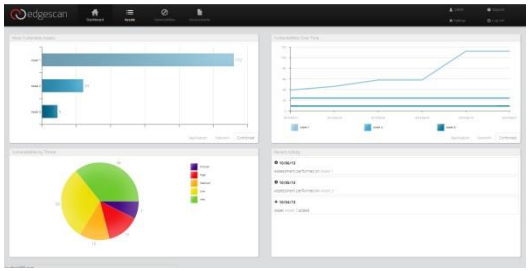
edgescan™

Digital Security Radar™

- **Digital security solution** to improve your defence against cyber-attacks
  - has helped many of the world's biggest organisations to improve their digital security

- **Detects security weaknesses** in your entire digital-asset-estate:
  - websites, apps (mobile/web/cloud), software, servers and networks

- Owned by **EU based** international digital security company - **BCC Risk Advisory**
  - professional team of technical consultants
  - lead by a world-renowned expert in digital security
  - involved in shaping web security in the industry
  - involved in OWASP, a non-profit organisation working to make the internet more secure



- Dedicated to **accuracy, quality** and the believe in **clear communication**

edgescan

Top 10 Mobile Threats/Risks

1	Insecure Data Storage	6	Improper Session Handling
2	Weak Server Side Controls	7	Insecure Data Storage
3	Insufficient Transport Layer Protection	8	Security Decisions Via Untrusted Inputs
4	Client Side Injection	9	Side Channel Data Leakage
5	Poor Authorization and Authentication	10	Sensitive Information Disclosure

Reference https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

edgescan

Top 10 Mobile Defenses

1	Identify and protect/erase sensitive data on the mobile device
2	Handle password credentials securely on the device
3	Ensure sensitive data is protected in transit
4	Implement user authentication, authorization and session management correctly
5	Keep backend APIs (services) and the platform (server) secure
6	Secure data integration with third party services and applications
7	Pay specific attention to the collection and storage of consent for the collection and use of the user's data
8	Implement controls to prevent unauthorized access to paid-for resources (wallet, SMS, phone calls etc.)
9	Ensure secure distribution/provisioning of mobile applications
10	Carefully check any runtime interpretation of code for errors

Reference https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

edgescan

Protect Sensitive Data on Mobile Device

- 113 cell phones are lost every minute
- 17 million $ worth of cell phones lost daily
- Developers often store sensitive data on the phone in mobile applications
- Platform controls to encrypt data are often weak and can be circumvented (i.e: keychain)

Reference https://www.owasp.org/index.php/OWASP Mobile_Security_Project

edgescan

M1 Insecure Data Storage

- Account name & password stored in flat text file

Risks & mitigating factors

- App accessed private information
- Password reuse likely
- App used in Arab Spring and other protests

edgescan

M1 Insecure Data Storage

Sensitive data

1. Authentication data
2. Regulated information
3. Business-specific

Recommendations

1. Business must define, classify, assign owner & set requirements.
2. Acquire, transmit, use and store as little sensitive data as possible.
3. Inform and confirm data definition, collection, use & handling.

Protections

1. Reduce use and storage
2. Encrypt or store one-way (scrypt)
3. Platform-specific secure storage with restricted permissions

Protect Sensitive Data on Mobile Device

- **General Recommendations**
  - Review third party libraries used to store and transmit sensitive information
  - Use SQLCipher to fully encrypt SQLite databases where possible
  - **Minimize the amount of data you store on the device to that which absolutely MUST be there**
- **iOS**
  - Avoid using NSUserDefaults to store sensitive information
  - Use the KeyChain instead, but avoid storing what you don't have to (e.g.- passwords, financial information)
- **Android**
  - **Never store sensitive data on the SD card**
  - Enforce least-privilege access to files with the MODE_WORLD_PRIVATE property (this is the default value…you have to explicitly use MODE_WORLD_READABLE to mess it up)

edgescan

KeyChain

Saving info in the Keychain is probably the most secure way of storing data on a non-jailbroken device.

It is also advisable to use your own encryption methods to encrypt the string that needs to be protected and then save on the keychain.

Plist

Plist files should also be never used to store confidential information like passwords etc

- Data can be fetched easily from inside the application bundle even on a non-jailbroken device.
- All the content inside a plist file is stored in unencrypted format.

Core

- SQLLite Storage:
  - Horror Stories…..
  - Core Data files are also stored as unencrypted database files in your application.
  - The Core Data framework internally uses Sql queries to store data, stored as .db files.
  - One can easily copy db files to their computer and using freely available tools (sqlite3) to examine all the content in these database files.
  - Use SQLCipher
    - http://www.zetetic.net/sqlcipher/open-source

edgescan

M2 Weak Server Side Controls

Mobile App Servers
RESTful API
SOAP
Web Service
Web XML

Recommendations

1. Always validate input and parameterize your database queries!
2. Don't trust the client
3. Harden mobile app servers & services
4. Beware information disclosure
5. Understand host & network controls
6. Perform integrity checking regularly

OWASP Top 10 Web Application Risks 2013

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

edgescan

M3 Insufficient Transport Layer Protection

Impact

1. Expose authentication data
2. Disclosure other **sensitive information**
3. Injection
4. Data tampering

Recommendations

1. Use platform-provided cryptographic libraries
2. Force strong methods & valid certificates
3. Test for certificate errors & warnings
4. Use pre-defined certificates, as appropriate
5. Encrypt sensitive information before sending
6. All transport, including RFID, NFC, Bluetooth Wifi, Carrier
7. Submit sensitive data over HTTPS POST

Protect Sensitive Data in Transit

- **Self Signed Certificates (DANGER)**
  - Developers use self-signed SSL certificates in dev or UAT environments (which is often pushed live)
- **Recommendations**
  - Provide feedback to users when a certificate failure is detected
  - Encrypt both cellular and wifi communications…don't assume that one is more secure than the other…trust nothing and no one!
  - By default, NSURLConnection (iOS) and HttpClient (Android) provide inherent protection against common certificate problems (fails against self-signed certs, untrusted CA, etc)
  - Pinning a certificate associates a remote host with an expected X509 certificate or public key

Certificate Pinning

What is Pinning

- Detect when an imposter with a fake but CA validated certificate attempts to act like the real server

- Typically certificates are validated by checking the signature hierarchy; MyCert is signed by IntermediateCert which is signed by RootCert, and RootCert is listed in my computer's "certificates to trust" store.
- Certificate Pinning is where you ignore that whole thing, and say trust *this certificate only* or perhaps trust only certificates *signed by this certificate*.

2 Types of pinning

- Carry around a copy of the server's public key;
- Great if you are distributing a dedicated client-server application since you know the server's certificate or public key in advance

•Note of the server's public key on first use (Trust-on-First-Use, Tofu)

- Useful when no *a priori* knowledge exists, such as SSH or a Browser

•https://www.owasp.org/index.php/Pinning_Cheat_Sheet

edgescan

Where? How?

Android:
Accomplished through a custom **X509TrustManager**. X509TrustManager should perform the customary X509 checks in addition to performing the pin.

iOS:
Perfromed through a **NSURLConnectionDelegate**.

edgescan

Example Code : Android

```
public final class PubKeyManager implements X509TrustManager
{
 private static String PUB_KEY = "30820122300d06092a864886f70d0101" +
  "0105000382010f003082010a0282010100b35ea8adaf4cb6db86068a836f3c85" +
  "5a545b1f0cc8afb19e38213bac4d55c3f2f19df6dee82ead67f70a990131b6bc" +
  "ac1a9116acc883862f00593199df19ce027c8eaaae8e3121f7f329219464e657" +
  "2cbf66e8e229eac2992dd795c4f23df0fe72b6ceef457eba0b9029619e0395b8" +
  "609851849dd6214589a2ceba4f7a7dcceb7ab2a6b60c27c69317bd7ab2135f50" +
  "c6317e5dbfb9d1e55936e4109b7b911450c746fe0d5d07165b6b23ada7700b00" +
  "33238c858ad179a82459c4718019c111b4ef7be53e5972e06ca68a112406da38" +
  "cf60d2f4fda4d1cd52f1da9fd6104d91a34455cd7b328b02525320a35253147b" +
  "e0b7a5bc860966dc84f10d723ce7eed5430203010001";

 public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException
 {
  if (chain == null) {
   throw new IllegalArgumentException("checkServerTrusted: X509Certificate array is null");
  }

  if (!(chain.length > 0)) {
   throw new IllegalArgumentException("checkServerTrusted: X509Certificate is empty");
  }

  if (!(null != authType && authType.equalsIgnoreCase("RSA"))) {
   throw new CertificateException("checkServerTrusted: AuthType is not RSA");
  }

  // Perform customary SSL/TLS checks
  try {
   TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");
   tmf.init((KeyStore) null);

   for (TrustManager trustManager : tmf.getTrustManagers()) {
    ((X509TrustManager) trustManager).checkServerTrusted(chain, authType);
   }
  } catch (Exception e) {
   throw new CertificateException(e);
  }

  // Hack ahead: BigInteger and toString(). We know a DER encoded Public Key begins
  // with 0x30 (ASN.1 SEQUENCE and CONSTRUCTED), so there is no leading 0x00 to drop.
  RSAPublicKey pubkey = (RSAPublicKey) chain[0].getPublicKey();
  String encoded = new BigInteger(1 /* positive */, pubkey.getEncoded()).toString(16);

  // Pin it!
  final boolean expected = PUB_KEY.equalsIgnoreCase(encoded);
  if (!expected) {
   throw new CertificateException("checkServerTrusted: Expected public key: "
          + PUB_KEY + ", got public key:" + encoded);
   }
  }
 }
}
```

Public Key of site

Check Key

edgescan

Transport Layer Protection Problems (iOS)

- canAuthenticateAgainstProtectionSpace and didReceiveAuthenticationChallenge are deprecated.
- Newer code uses willSendRequestForAuthenticationChallenge

```
- (BOOL)connection:(NSURLConnection *)connection
    canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)protectionSpace {

    return [protectionSpace.authenticationMethod isEqualToString:
        NSURLAuthenticationMethodServerTrust];
}

- (void)connection:(NSURLConnection *)connection
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {

    [challenge.sender useCredential:[NSURLCredential credentialForTrust:challenge.
        protectionSpace.serverTrust]
         forAuthenticationChallenge:challenge];
}

@end
```

OWASP iGoat Example

edgescan

Transport Layer Protection Problems (Android)

- Android's HttpClient class fails when it encounters certificate issues by default.
- Don't override the default fail-closed behavior if you don't have to!

```
public class CustomSSLSocketFactory extends SSLSocketFactory {
    SSLContext sslContext = SSLContext.getInstance("TLS");

    public CustomSSLSocketFactory(KeyStore truststore)
            throws NoSuchAlgorithmException, KeyManagementException,
            KeyStoreException, UnrecoverableKeyException {
        super(truststore);

        TrustManager tm = new X509TrustManager() {
            public java.security.cert.X509Certificate[] getAcceptedIssuers() {
                return null;
            }

            @Override
            public void checkClientTrusted(
                    java.security.cert.X509Certificate[] chain, String authType)
                    throws java.security.cert.CertificateException {
                // TODO Auto-generated method stub

            }

            @Override
            public void checkServerTrusted(
                    java.security.cert.X509Certificate[] chain, String authType)
                    throws java.security.cert.CertificateException {
                // TODO Auto-generated method stub

            }
        };

        sslContext.init(null, new TrustManager[] { tm }, null);
    }
```

OWASP GoatDroid Example

edgescan

M4 Client Side Injection

Impact

1. App or device compromise
2. Abuse resources or services (SMS, phone, payments, online banking)
3. Extract or inject data
4. **Man-in-the-Browser (MITB)**

Recommendations

1. Always validate input
2. Thin client programming
3. Harden mobile app clients
4. Beware information disclosure
5. Perform integrity checking regularly

edgescan

- iOS: Any data to be rendered via loadHTMLString should be strictly validated by the ViewController component.

See
https://developer.apple.com/library/ios/#documentation/Security/Conceptual/SecureCodingGuide/Articles/ValidatingInput.html#//apple_ref/doc/uid/TP40007246-SW3

- **Android**:

Disable JavaScript and Plugin support if they are not needed.

Disable local file access. This restricts access to the app's resource and asset directory and mitigates against an attack from a web page which seeks to gain access to other locally accessible files.

Prevent loading content from 3rd party hosts.: Override **shouldOverrideUrlLoading** and **shouldInterceptRequest** to intercept, inspect, and validate most requests initiated from within a WebView.

A whitelist scheme can also be implemented by using the URI class to inspect the components of a URI and ensure it matches a whitelist of approved resources.

M5 Poor Authorization and Authentication

Impacts

1. Account takeover
2. Confidentiality breach
3. Fraudulent transactions

Recommendations

1. Use appropriate methods for the risk
2. Unique identifiers as additional (not only) factors
3. Differentiate client-side passcode vs. server authentication
4. Ensure out-of-band methods are truly OOB (this is hard)
5. Hardware-independent identifiers (ie. Not IMSI, serial, etc.)
6. Multi-factor authentication, depending on risk
7. *Implement Mobile Session Management Properly*

M6 Improper Session Handling

Impacts

1. Account takeover
2. Confidentiality breach
3. Fraudulent transactions

Recommendations

1. Allow revocation of device/password
2. Use strong tokens and generation methods
3. Consider appropriate session length (longer than web)
4. Reauthenticate periodically or after focus change
5. Store and transmit session tokens securely

edgescan

Implement Session Management Properly

- **Ensure Session Management is Handled Correctly**
  - Require authentication credentials or tokens to be passed with any subsequent request
  - Use unpredictable session identifiers with high entropy
  - Reseed your random number generator per message
  - Limit the life of a session identifier
    - idle and absolute timeout
    - session fixation protection

edgescan

Implement Session Management Properly

- **Other Recommendations**
  - Use authentication that ties back to the end user identity (rather than the device identity).
  - Use context to add security to session management
  - IP location, general Geo-location, etc

edgescan

M7 Security Decisions via Untrusted Inputs

Description

Reliance on files, settings, network resources or other inputs which may be modified.

Recommendations

1. Validate all inputs
2. Digitally sign decisioning inputs, where possible
3. Ensure trusted data sources for security decisions

edgescan

M8 Side Channel Data Leakage

Side channel data

1. Caches
2. Keystroke logging (by platform)
3. Screenshots (by platform)
4. Logs

Recommendations

1. Consider server-side leakage
2. Reduce client-side logging
3. Consider mobile-specific private information
4. Consider platform-specific data capture features
5. Securely cache data (consider SSD limitations)

edgescan

M9 Broken Cryptography

Cryptography
…is not encoding
…is not obfuscation
…is not serialization
…is best left to the experts

"The only way to tell good cryptography from bad cryptography is to have it examined by experts."
-Bruce Schneier

Recommendations

1. Use only well-vetted cryptographic libraries
2. Understand one-way vs. two-way encryption
3. Use only well-vetted cryptographic libraries (not a typo)
4. Use only platform-provided cryptographic storage
5. Use only well-vetted cryptographic libraries (still not a typo)
6. Protect cryptographic keys fanatically
7. Use only well-vetted cryptographic libraries (seriously - always do this)

M10 Sensitive Information Disclosure

Sensitive application data

1. API or encryption keys
2. Passwords
3. Sensitive business logic
4. Internal company information
5. Debugging or maintenance information

Recommendations

1. Store sensitive application data server-side
2. Avoid hardcoding information in the application
3. Use platform-specific secure storage areas

edgescan

Additional mobile security issues…

edgescan

Securing Inter-process Communication

- iOS and Android make heavy use of Inter-process Communication (IPC)

- **Problems with IPC**
  - Blindly trusting the caller
  - Validating and sanitizing data sent across IPC trust boundaries
  - Ensuring that users consent to sensitive actions triggered by IPC calls

- **Recommendations when using IPC**
  - Validate what gets sent access IPC boundaries (check for null, proper data types, malicious strings that may result in XSS, SQLi, format string exploits, etc.)
  - Prompt the user before allowing sensitive actions to succeed (making a phone call, sending an SMS, etc.)
  - For Android, favor a declarative approach to permission checking vs. programmatic. Much easier to get right and implement uniformly as opposed to runtime checks within code for permissions.

Control Access to Paid for Mobile Services

- **Limit access to SMS, Phone Calls, Wallet Apps, etc**
  - Apps with privileged access to such API's should take particular care to prevent abuse, considering the financial impact of vulnerabilities
  - Maintain logs of access to paid-for resources in a non-reputable format
    - (e.g. a signed receipt sent to a trusted server backend – with user consent)
  - Check for anomalous usage patterns in paid-for resource usage and trigger re- authentication.
    - E.g. when significant change in location occurs, user-language changes etc.

edgescan

Control Access to Paid for Mobile Services

- **Limit access to SMS, Phone Calls, Wallet Apps, etc**
  - Consider using a white-list model by default for paid-for resource addressing - e.g. address book only unless specifically authorised for phone calls.
  - Authenticate all API calls to paid-for resources (e.g. using an app developer certificate and proper authentication and session mangement).
  - Ensure that wallet API callbacks do not pass cleartext account/pricing/ billing/item information.
  - Warn user and obtain consent for any cost implications for app behaviour.
  - Implement best practices such as fast dormancy (a 3GPP specification), caching, etc. to minimize signalling load on base stations.

edgescan

Reverse Engineering

- Do not store sensitive infromation such as authentication credentials, PII or system settings in the source code of the application.
- Android: Android applications should have android:debuggable=”false” set in the application manifest to prevent easy run time manipulation by an attacker or malware.

edgescan

Android Intent hijacking

- Intents are used for inter-component signaling and can be used:
  - To start an Activity, typically opening a user interface for an app.
  - As broadcasts to inform the system and apps of changes
  - To start, stop, and communicate with a background service To access data via ContentProviders As callbacks to handle events.
- If public intents are used other applications can intercept and manipulate or spoof user content etc.
- It is possible to set component as android:exported=false in the app's Manifest to prevent this.

edgescan

Tap Jacking

- Leveraging Android Toasts, tapJacking occurs when a malicious application displays a fake user interface that seems like it can be interacted with, but actually passes interaction events such as finger taps to a hidden user interface behind it.
  - setFilterTouchesWhenObserved method or set the android:filterTouchesWhenObscured property in your layout XML to true.
  - For more fine-grained control, you can override the onFilterTouchEventForSecurity method on a View subclass and discard specific MotionEvents to your liking.

edgescan

Keyboard Caching

- iOS: Keyboard Caching Should be disabled for any potentially sensitive fields Set UITextField property autocorrectionType = UITextAutocorrectionNo
- Android: Android contains a user dictionary, where words entered by a user can be saved for future auto correction. This user dictionary is available to any app without special permissions.

  - For UITextField, look into setting the autocorrectionType property to UITextAutocorrectionNo to disable caching. Such settings may change over time as the SDK updates so ensure it is fully researched.
  - Add an enterprise policy to clear the keyboard dictionary at regular intervals. This can be done by the end user by simply going to the Settings application, General > Reset > Reset Keyboard Dictionary.

edgescan

Info Leakage via Copy and Paste

- iOS now supports copy/paste. Sensitive data may be stored and recoverable from clipboard in clear text, regardless of whether the data was initially encrypted. The copy/paste API is still maturing and may leak sensitive data.

- Android also supports copy/paste by default and the clipboard can be accessed by any application.

edgescan

Info Leakage via Copy and Paste

Where appropriate, disable copy/paste for areas handling sensitive data.

Eliminating the option to copy can help avoid data exposure.

iOS: Clear pasteboard on applicationWillTerminate pasteBoard.items = nil

edgescan

Additional Resources

- images.apple.com/ipad/business/docs/**iOS_Security**_Feb14.pdf

edgescan