bitinspect

Security Test der Mobile App xXx (Android/iOS)

Technischer Bericht

XXX AG

3.7.2017

Inhaltsverzeichnis

Zusammenfassung

An dieser Stelle möchten wir die gewonnenen Erkenntnisse zusammenfassen und die Ergebnisse in einer Übersicht präsentieren.
Die untersuchte mobile Applikation (App) xXx erlaubt es dem Benutzer sein Warenkorb bequem mit dem Smartphone zu bezahlen. Zum Bezahlen der Einkäufe wird die KundenID mit dem Smartphone eingescannt oder von Hand eingetippt. Der Barcode auf der Kundenkarte dient zur Abfrage des Status beim Backend-Server, der dann die Informationen zum Kunden (Wann wurde zuletzt eingekaufe/ Wieviele Käufe wurden bereits getätigt/ Wie hoch waren die Einkäufe usw.) liefert. Anschließend kann der Einkauf bezahlt werden und die bezahlten Einkäufe werden nach Hause geliefert. Frühere Einkäufe werden dann in einer Historie auf dem Smartphone gespeichert.
Bei der Analyse der App haben wir folgende Probleme entdeckt:

- Speicherung der Zugangsdaten
  Die Android-Applikation speichert die Benutzer-Zugangsdaten unverschlüsselt auf der SD-Karte. Dadurch kann ein Angreifer beim physischen Zugriff diese Daten auslesen und missbrauchen, in dem er im Namen des Benutzers einkauft.
- TLS Man-in-the-Middle
  Die Applikation prüft nicht, ob die Kommunikation mit dem korrekten Backend-Server stattfindet. Angreifer können so unter bestimmten Voraussetzungen einen Man-in-the-Middle Angriff durchführen und die gesamte Kommunikation zwischen App und Backend abhören und beliebig manipulieren.
- Brute-Force von gültigen BenutzerID
  Der Backend-Server erlaubt das Durchprobieren von BenutzerID. So können Angreifer gültige BenutzerID erraten und diese für Betrugszwecke verwenden. Damit ist es möglich statt der eigenen BenutzerID im Warenkorb eine der gültigen BenutzerID eines anderen Benutzers mit niedrigerem Zahlbetrag zu erraten. Wird eine günstigere BenutzerID „erraten" und bezahlt, wird in Folge die gewünschte (teuerere) Ware geliefert, obwohl ein geringerer Betrag bezahlt wurde.
- TLS Verschlüsselung
  Der Backend-Server erlaubt zwar grundsä tzlich nur verschlü sselte Kommunikation, die konkrete Konfiguration der TLS-Verschlüsselung erlaubt jedoch die Benutzung von einigen veralteten Verschlüsselungsalgorithmen, die bekannte Schwachstellen enthalten.
- IPv6-Fähigkeit
  Es soll außerdem beachtet werden, dass Apple seit dem 1. Juni 2016 in App Store nur noch (Updates für) Applikationen aufnimmt, die in reinem IPv6-Netzwerk funktionieren. Derzeit nutzt das Backend nur IPv4 Adressen und erfüllt daher die nötigen Voraussetzungen nicht.
- Reverse Engineering
  Die Applikation lässt sich mit geringem Aufwand dekompilieren. Angreifer können so leicht an lesbaren Quelltext kommen und damit die Funktionsweise der Applikation in Details nachvollziehen.

OWASP MobileTop10-2016

Die OWASP MobileTop10 zeigt die am häufigsten vorkommenden Sicherheitsprobleme vom mobilen Apps. Wir beziehen uns hier auf den aktuellen Release-Candidate 2016. Leider existiert derzeit keine offizielle deutsche Übersetzung. Wir verwenden hier daher die englischsprachigen Bezeichnungen.

Beschreibung	Risiko
M1 - Improper plattform usage	Keine Probleme gefunden.
M2 - Insecure Data Storage	Kritisch
M3 - Insecure communication	Hoch
M4 - Insecure Authentication	Keine Probleme gefunden.
M5 - Insufficient Cryptography	Keine Probleme gefunden.
M6 - Insecure Authorization	Hoch
M7 - Client Code Quality Issues	Keine Probleme gefunden.
M8 - Code tampering	Nicht vorhanden
M9 - Reverse Engineering	Keine Probleme gefunden.
M10 - Extraneous functionality	Keine Probleme gefunden.

Vorgehensweise

Details

Kunde:	XXX AG
Ziele:	Mobile App "xXx" für Android Version 4.3 und iOS Version 4.6
Zeitraum:	01.01.2017
Durchgeführt von:	Jens Liebau, Wladimir Paulsen

Durchführung

Die Applikation wurde auf mehrere Arten getestet. An dieser Stelle möchten wir das Vorgehen näher beschreiben:

1. Die App (iOS/Android) wurde auf den Testgeräten installiert und es wurde sowohl das normale als auch das Verhalten bei lokalen Angriffsversuchen mit gezielt manipulierten Fehleingaben getestet.
2. Die Android-Version der mobilen Applikation wurde aus dem Google App Store heruntergeladen. Die APK-Datei wurde dekompiliert (dex2jar), um an den Quellcode der Applikation zu gelangen. Der Quellcode wurde dann erst automatisiert mit einem Tool (AndroBugs) auf Sicherheitsschwachstellen untersucht. Im nächsten Schritt wurde der Quelltext manuell begutachtet (JD-GUI), um weitere Schwachstellen und Fehler in der Logik zu entdecken.
3. Zuletzt wurde die Applikation dynamisch untersucht, indem die gesamte Kommunikation durch ein Proxy (ZAP) geleitet wurde. Die Aufrufe der App wurden mitgeschnitten und bei einem Man-in-the-Middle Angriff sowohl die Aufrufe als auch die Server-Antworten aktiv manipuliert wurden. Dabei wurde auch das Verhalten des Backend-Servers getestet.

Die gefundenen Schwachstellen wurden dann der OWASP MobileRisk Top10 zugeordnet. Diese Liste beruht auf der Auswertung von Sicherheitsunternehmen, die mobile Applikationen auf Sicherheitsschwachstellen analysieren, und beschreibt die häufigesten Schwachstellen in den mobilen Anwendungen. Die Liste selbst liegt nur in englischer Sprache vor. Wir verwenden hier daher die englischsprachigen Bezeichnungen.

Risikobewertung

Im ersten Schritt werden den Schwachstellen die Eintrittswahrscheinlichkeiten und Schadenshöhen zugeordnet:

Bewertungsstufen der Eintrittswahrscheinlichkeit und Schadenshöhe	
Wert	**Beschreibung**
x <= 0	Nicht vorhanden
0 < x <= 3	Gering

3 < x <= 6	Mittel
6 < x <= 9	Hoch

Im zweiten Schritt werden diese Werte miteinander multipliziert. Das Ergebnis der Multiplikation wird entsprechend der Tabelle bewertet:

Bewertungsstufen des Gesamtrisikos	
Wert	**Beschreibung**
x <= 0	Nicht vorhanden
0 < x <= 3	Vernachlässigbar
3 < x <= 9	Gering
9 < x <= 21	Mittel
21 < x <= 45	Hoch
45 < x <= 81	Kritisch

Die Risikobewertung wurde anhand folgender Matrix vorgenommen:

Risikobewertung		Eintrittswahrscheinlichkeit			
		Nicht vorhanden	**Gering**	**Mittel**	**Hoch**
Schadenshöhe	**Hoch**	Nicht vorhanden	Mittel	Hoch	Kritisch
	Mittel	Nicht vorhanden	Gering	Mittel	Hoch
	Gering	Nicht vorhanden	Vernachlässigbar	Gering	Mittel
	Nicht vorhanden	Nicht vorhanden	Nicht vorhanden	Nicht vorhanden	Nicht vorhanden

Tests

Test: M1 - Improper plattform usage

Beschreibung:

This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Mittel	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Test: M2 - Insecure Data Storage

Beschreibung:

This can result in data loss, in the best case for one user, and in the worst case for many users. It may also result in the following technical impacts: extraction of the app's sensitive information via mobile malware, modified apps or forensic tools.
This covers insecure data storage and unintended data leakage. Data stored insecurely includes, but is not limited to, the following:

- SQL databases
- Log files
- XML data stores or manifest files;
- Binary data stores
- Cookie stores
- SD card
- Cloud synced

Unintended data leakage includes, but is not limited to, vulnerabilities from:

- The OS
- Frameworks
- Compiler environment
- New hardware

Betroffene Systeme: Mobile Application,

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Hoch	Kritisch

Details für Mobile Application:

Die Applikation benutzt den externen SD-Speicher zur Speicherung von Passwörtern. Dazu wird die Methode **storeAuthPersistent** der Klasse **UserData** aufgerufen:

```
void storeAuthPersistent(){
   File file = new File (Environment.getExternalFilesDir(), "authData.txt");
   String userID = this.getUserID();
   String password = this.getCurrentPassword();
   try{
      FileOutputStream fos;
      fos = new FileOutputStream(file);
      fos.write((userID+":"+password).getBytes());
      fos.close();
   } catch (IOException e){
      Log.w("ExternalStorage", "Error writing " + file, e);
   }
}
```

Diese gespeicherten Passwörter lassen sich leicht auslesen, entweder über USB-Debugging oder durch das Auslesen der SD-Karte:

```
wladimir@kalle:/usr/share/android-sdk/platform-tools$ ./adb shell cat /sdcard/authData.txt
123456:SuperS3cur3
```

Screenshot von Mobile Application

Gegenmaßnahmen:

The cardinal rule of mobile apps is to not store data unless absolutely necessary. As a developer you have to assume that the data is forfeited as soon as it touches the phone. You also have to consider the implications of losing mobile users' data to a silent jailbreak or root exploit. If the usability versus security trade-off is too much for you, OWASP recommends scrutinizing your platforms data security APIs and making sure you're calling them appropriately. The lesson here is to know what data is being stored and protect it appropriately.

For iOS:

- Never store credentials on the phone file system. Force the user to authenticate using a standard web or API login scheme (over HTTPS) to the application upon each opening and ensure session timeouts are set at the bare minimum to meet the user experience requirements.
- Where storage or caching of information is necessary consider using a standard iOS encryption library such as CommonCrypto
- If the data is small, using the provided apple keychain API is recommended but, once a phone is jailbroken or exploited the keychain can be easily read. This is in addition to the threat of a bruteforce on the devices PIN, which as stated above is trivial in some cases.
- For databases consider using SQLcipher for Sqlite data encryption
- For items stored in the keychain leverage the most secure API designation, kSecAttrAccessibleWhenUnlocked (now the default in iOS 5) and for enterprise managed mobile devices ensure a strong PIN is forced, alphanumeric, larger than 4 characters.
- For larger or more general types of consumer-grade data, Apple's File Protection mechanism can safely be used (see NSData Class Reference for protection options).
- Avoid using NSUserDefaults to store senstitve pieces of information as it stores data in plist files.
- Be aware that all data/entities using NSManagedObects will be stored in an unencrypted database file.

For Android:

- For local storage the enterprise android device administration API can be used to force encryption to local file-stores using "setStorageEncryption"
- For SD Card Storage some security can be achieved via the 'javax.crypto' library. You have a few options, but an easy one is simply to encrypt any plain text data with a master password and AES 128.
- Ensure any shared preferences properties are NOT MODE_WORLD_READABLE unless explicitly required for information sharing between apps.

Test: M3 - Insecure communication

Beschreibung:

This risk covers all aspects of getting data from point A to point B, but doing it insecurely. It encompasses mobile-to-mobile communications, app-to-server communications, or mobile-to-something-else communications. This risk includes all communications technologies that a mobile device might use: TCP/IP, WiFi, Bluetooth/Bluetooth-LE, NFC, audio, infrared, GSM, 3G, SMS, etc.

Betroffene Systeme: Mobile Application,

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Mittel	Hoch

Details für Mobile Application:

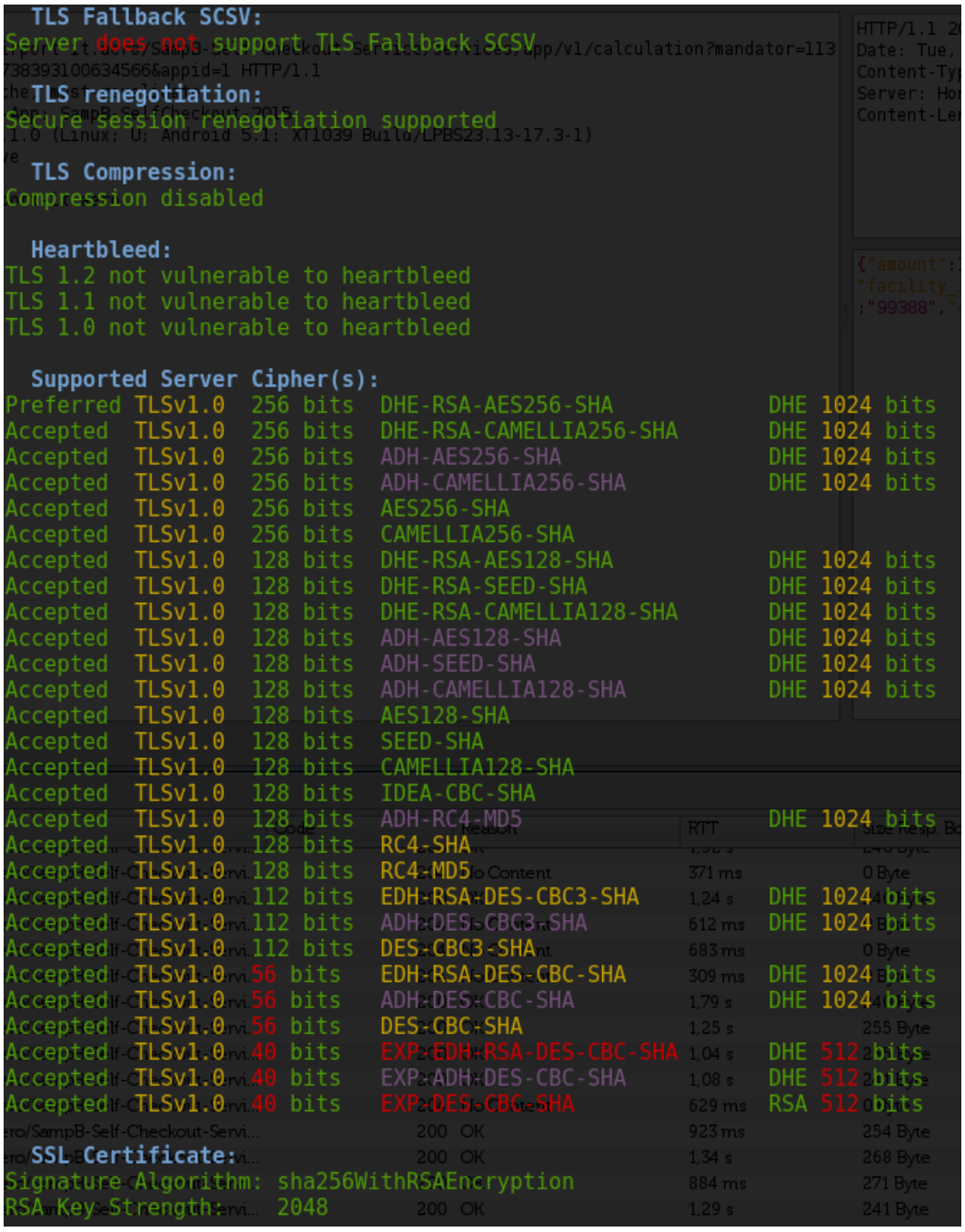
Das Backend lässt auch schwache TLS-Verschlüsselung zu:

- RC4-Verschlüsselung
- TLS-Protokoll in der Version 1.0
- Schlüsselaustauschalgorithmen mit unter 2048-Bit Schlüssellänge
- Symmetrische Verschlüsselung mit unter 128-Bit Schlüssellänge

Dies lässt sich einfach entweder mit dem Tool

```
sslyze --regular mobile-api.xxxag.net
```

oder online unter https://www.ssllabs.com/ssltest/ überprüfen



Screenshot von Mobile Application

Gegenmaßnahmen:

- Assume that the network layer is not secure and is susceptible to eavesdropping.
- 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.
- 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.
- Use strong, industry standard cipher suites with appropriate key lengths.
- Use certificates signed by a trusted CA provider.
- Never allow self-signed certificates, and consider certificate pinning for security conscious applications.
- 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.
- Do not send sensitive data over alternate channels (e.g, SMS, MMS, or notifications).
- 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.

Test: M4 - Insecure Authentication

Beschreibung:

This category captures notions of authenticating the end user or bad session management. This can include:

- Failing to identify the user at all when that should be required
- Failure to maintain the user's identity when it is required
- Weaknesses in session management

Authentication issues are a reasonable place to put privacy issues related to authentication. For example, if a mobile app is using device-specific data like IMEI, Bluetooth MAC address, or other hardware identifiers as an authenticator for the user, that may create privacy expectations for the app developer/owner.

Authentication over an insecure channel also belongs in this category if that results in spoofing, replaying or otherwise attacking the authentication. Storing the password badly in the mobile app belongs in "insecure data storage" and transmitting passwords in the clear belongs in "insecure communications". For example, if observing the enrolment of one user into a service gave an attacker insight into data that would be used in subsequent enrolments, that would be an authentication issue belonging in this category.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Test: M5 - Insufficient Cryptography

Beschreibung:

The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3 – Insecure Communication . Also, if the app fails to use cryptography at all when it should, that probably belongs in M2 – Insecure Data . This category is for issues where cryptography was attempted, but it wasn't done correctly.

Common failures in this category include:

- Weak ciphers
- Small keys
- Wrong type of crypto (e.g., symmetric when asymmetric is more appropriate)
- Vulnerability to a well known cryptanalytic attack like
    - Chosen plaintext attack
    - Known plaintext attack
    - Poor key selection (e.g., predictable randomness)

All issues in this category share a common quality: The app did attempt to protect the data, and the protection probably works to some degree. But the finding is usually that the protection is deemed insufficient for one reason or another.

The data that is being protected with cryptography is likely to be exposed. Typical results are related to either the confidentiality or the integrity of the data that was protected. Confidentiality risks might include an offline brute force attack against the data or inference attacks against the encrypted data. Such risks mean that an attacker learns some or all the protected information. Integrity risks might include replay attacks if a cryptographic signature can be forged. Insecure code might get loaded if a code signature can be forged. Guessable or predictable cryptographic tokens might enable fake transactions.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Test: M6 - Insecure Authorization

Beschreibung:

This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.).

If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.

Typical risk impacts relate to granting access to unauthorized users. Users may be able to perform CRUD operations that they should not be able to. They may be able to invoke services or receive service that their credentials do not entitle them to.

Betroffene Systeme: Mobile Application,

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Mittel	Hoch

Details für Mobile Application:

Das Backend der mobilen Anwendung führt eine Authorisierung anhand eines HTTP-Headers "XXX-Auth" durch. Wird dieser Header nicht gesendet, reagiert das Backend nicht auf Anfragen. Allerdings lässt sich dieser Header entweder durch einen Man-in-the-middle Angriff oder durch Dekompilierung der mobilen Anwendung ermitteln. Damit kann ein Angreifer weitere Angriffe unabhängig von der mobilen Anwendung durchführen. Unter anderem lassen sich dann Brute-Force-Angriffe durchführen, mittels derer weitere Kundennummern bestimmt werden können.

```
GET https://mobile-api.xxxag.net/get-customer-record?recordid=123456 HTTP/1.1
Cache-Control: no-cache, must-revalidate
XXX-Auth: NoOneCanFindIt
Host: mobile-api.xxxag.net
-----------------------------------
HTTP/1.1 200 OK
Date: Mon, 01 Jan 2017 00:00:00 GMT
Content-Type: application/json;charset=UTF-8
Server: Apache 2.4.10
{"Name":"Test","Forename":"User","Amount":100,"RecordID":123456}
```

Gegenmaßnahmen:

- Developers should assume all client-side authorization controls can be bypassed by malicious users. Authorization controls must be re-enforced on the server-side whenever possible.
- Due to offline usage requirements, mobile apps may be required to perform local authorization checks within the mobile app's code. If this is the case, developers should instrument local integrity checks within their code to detect any unauthorized code changes.

Test: M7 - Client Code Quality Issues

Beschreibung:

This is the catch-all for code-level implementation problems in the mobile client. This captures the risks that come from vulnerabilities like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.

This is distinct from M1 – Improper Platform Usage because it usually refers to the programming language itself (e.g., Java, Swift, Objective C, JavaScript). A buffer overflow in C or a DOM-based XSS in a Webview mobile app would be code quality issues.

Bad code can allow attackers to exploit the business logic, perhaps bypassing security controls enforced on the device. Code-level bugs reveal sensitive data in unexpected ways.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Mittel	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Test: M8 - Code tampering

Beschreibung:

This category covers:

- binary patching
- local resource modification
- method hooking
- method swizzling
- dynamic memory modification

Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.

This category of attack can change the implicit or explicit logic of the application. This could be used to subvert or short-circuit in-application purchases or licensing, leading to lost revenue for the application developer. It could also be used to clone the application into a malicious variant which could contain a malware payload under the guise of a legitimate application. Also, this method can be used to change or discontinue network traffic to the application's back end servers— or redirect the application to talk to the attacker's servers.

Betroffene Systeme: Mobile Application,

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Mittel	Nicht vorhanden	Nicht vorhanden

Details für Mobile Application:

Die Anwendung prüft vor dem Start der App, ob das Telefon “gerootet” oder mit “Debug”-Flag (re-)kompiliert wurde und bricht die Ausführung in dem Fall ab.

```
import android.os.Build;
import java.io.File;

public class b
{
  public static boolean a()
  {
    String[] arrayOfString = System.getenv("PATH").split(":");
    int i = arrayOfString.length;
    for (int j = 0;; j++)
    {
      boolean bool = false;
      if (j < i)
      {
        if (new File(arrayOfString[j], "su").exists()) {
          bool = true;
        }
      }
      else {
        return bool;
      }
    }
  }

  public static boolean b()
  {
    String str = Build.TAGS;
    return (str != null) && (str.contains("test-keys"));
  }

  public static boolean c()
  {
    String[] arrayOfString = { "/system/app/Superuser.apk", "/system/xbin/daemonsu", "/system/etc/init.d/99SuperSUDaemon", "/system/bin/.ext/.su",
    int i = arrayOfString.length;
    for (int j = 0; j < i; j++) {
      if (new File(arrayOfString[j]).exists()) {
        return true;
      }
    }
    return false;
  }
}
```

```
  private native void init();

  protected void onCreate(Bundle paramBundle)
  {
    init();
    if ((b.a()) || (b.b()) || (b.c())) {
      a("Root detected!");
    }
    if (a.a(getApplicationContext())) {
      a("App is debuggable!");
    }
    new AsyncTask()
    {
      protected String a(Void... paramAnonymousVarArgs)
      {
        while (!Debug.isDebuggerConnected()) {
          SystemClock.sleep(100L);
        }
        return null;
      }

      protected void a(String paramAnonymousString)
      {
        MainActivity.a(MainActivity.this, "Debugger detected!");
      }
    }.execute(new Void[] { null, null, null });
    this.m = new CodeCheck();
    super.onCreate(paramBundle);
    setContentView(2130968603);
  }
```

Screenshot von Mobile Application

Gegenmaßnahmen:

The application must follow secure coding techniques for the following security components within the mobile app:

- Jailbreak Detection Controls
- Checksum Controls
- Certificate Pinning Controls
- Debugger Detection Controls

Further reading:

- Technical risks and their mitigations
- Architectural principles

Test: M9 - Reverse Engineering

Beschreibung:

This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.

By reverse engineering, an adversary can enumerate and/or bypass the business logics, bypass security controls, facilitating source code piracy and enabling code tampering. If the source code is not obfuscated, then it becomes very easy for an adversary/competitor to replicate your application. Adversaries can then re-pack the application and distribute it to public (via various means). This involves a high business risk: loss of revenue, brand damage, or counterfeit copies of the application. These counterfeit copies can also facilitate phishing or credential stealing.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Gering	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Test: M10 - Extraneous functionality

Beschreibung:

Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.

There is high risk of stealing sensitive data or accessing unauthorized functions via these extra capabilities. If a banking application has malicious code implanted in it, it will not only steal customer information and funds but also damage the bank's reputation. This malicious code will initiate an outgoing connection, which will be treated as genuine connection and not malicious. Despite of all security controls, it will be very difficult for bank to detect malicious activity by the application on mobile devices.

Another example is insurance application. If a developer injects malicious code snippet in any insurance application, then there is high probability of stealing customer data which includes all personal details. Imagine if this malicious snippet is monitoring your payment transactions.

Betroffene Systeme: Keine

Schadenshöhe	Eintrittswahrscheinlichkeit	Gesamtrisiko
Hoch	Nicht vorhanden	Nicht vorhanden

Gegenmaßnahmen:

- Keine Gegenmaßnahmen notwendig oder entsprechende Schutzmaßnahmen sind bereits umgesetzt.

Testergebnisse

V3.2 Testing for custom implementation of cryptography

Es wurde eigene AES-Implementierung in der Klasse asb gefunden:

Failed
