Fraunhofer SIT

FRAUNHOFER-INSTITUT FÜR SICHERE INFORMATIONSTECHNOLOGIE

WHITE PAPER

4 JAHRE, 200 SCHWACHSTELLEN

Werden wir wirklich besser? Erfahrungen aus vier Jahren Offensive Security

ATHENE
Nationales Forschungszentrum
für angewandte Cybersicherheit

4 JAHRE, 200 SCHWACHSTELLEN

Werden wir wirklich besser? Erfahrungen aus vier Jahren Offensive Security

Impressum

Autor
Dr.-Ing. Steven Arzt

Kontakt
Dr.-Ing. Steven Arzt
Fraunhofer-Institut
für Sichere Informationstechnologie SIT
Rheinstraße 75
64295 Darmstadt

Inhalt

1 Einleitung

Viele Geräte und Dienste des täglichen Lebens verarbeiten sensible Daten oder führen sicherheitskritische Operationen durch. Insbesondere Mobilgeräte haben jederzeit Zugriff auf Standort, Zahlungsdaten (Mobile Payment) oder Textnachrichten des Benutzers. Für den Benutzer ist es essenziell, dass diese Daten nicht in die Hände von Unbefugten gelangen. Aus nur wenigen Datenpunkten lassen sich bereits eindeutige Kennungen für Geräte und Benutzer ableiten, die sich mit anderen Datenquellen verknüpfen lassen. Standortdaten ermöglichen Rückschlüsse auf Arbeits- und Wohnort des Benutzers und damit mittels statistischer Daten auf Einkommens- und Lebensverhältnisse. Für die Durchsetzung der informationellen Selbstbestimmung steht deshalb nicht nur die Datenschutzerklärung des Anbieters im Fokus, also die geplante Datennutzung, sondern auch die IT-Sicherheit und damit Möglichkeiten zum herstellerseitig ungeplanten Missbrauch der Daten.

Neben dem Datenschutz zeigen sich Themen der IT-Sicherheit auch bei der Durchführung von alltäglichen Aktivitäten im Sinne einer Steuerung oder Beeinflussung eines Systemzustands. Gelingt es einem Angreifer beispielsweise, über das Online oder Mobile Banking Geld vom Konto eines Benutzers zu stehlen, erleidet das Opfer einen Schaden, auch wenn der Angreifer keinen Einblick in den Kontostand erhält.

Anwendungen müssen daher sowohl die verarbeiteten Daten als auch die angebotenen Operationen vor Missbrauch schützen. Für den privaten Nutzer ist das Sicherheitsniveau einer Anwendung oftmals nicht oder nur indirekt erkennbar, z.B. über Presse- oder Testberichte zu einzelnen Anbietern oder Produkten. Für Unternehmen ist eine detaillierte Prüfung jeder eingesetzten Anwendung auf Sicherheitslücken mit signifikantem Aufwand und dadurch mit Kosten verbunden. Um die fortschreitende Digitalisierung von Wirtschaft, Gesellschaft und öffentlicher Verwaltung zu unterstützen, müssen Produkte ein hinreichendes Mindestmaß an IT-Sicherheit bieten. Dadurch lastet auf den Herstellern eine besondere Verantwortung zur Durchführung sicherer Entwicklungsprozesse. Leider weisen immer noch zahlreiche Produkte des täglichen Gebrauchs – privat wie im Unternehmen – zahlreiche kritische Sicherheitslücken auf, wie unsere Erfahrungen mit Sicherheitstests zeigen.

In diesem Whitepaper fassen wir die Sicherheitsanalysen der Offensive-Security-Gruppe TeamSIK am Fraunhofer SIT aus den letzten vier Jahren zusammen, ziehen Vergleiche zwischen den Schwachstellen aus unterschiedlichen Domänen und Jahren und leiten Vorschläge zur Verbesserung der IT-Sicherheit daraus ab.

2 TeamSIK Offensive Security

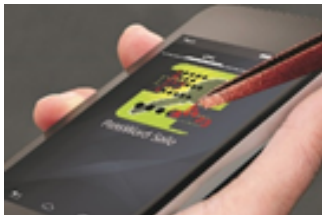
Fraunhofer SIT betreibt als führendes Institut für angewandte IT-Sicherheit in Deutschland seit 2015 die Offensive-Security-Gruppe TeamSIK. In dieser Gruppe untersuchen Studierende und wissenschaftliches Personal des Instituts gemeinsam Anwendungen, Dienste, und Geräte auf Sicherheitslücken. Im Gegensatz zu anderen Gruppen, die sich auf die Teilnahme an Ctf-Wettbewerben (Capture the Flag) spezialisiert haben, und im Wettkampfumfeld nach bewusst integrierten Lücken in akademischen Beispielen suchen, fokussiert TeamSIK sich seit seiner Gründung auf reale Systeme. Wir verwenden Apps, Geräte, und Dienste, die jeder Bürger erwerben kann, und greifen unsere eigenen Geräte und Daten an, immer auf der Suche nach Schwachstellen und Datenschutzproblemen. Die gefundenen Probleme melden wir an die jeweiligen Hersteller, damit sie im Interesse aller Nutzer behoben werden können.

In den vergangenen fünf Jahren hat TeamSIK sich den folgenden Bereichen gewidmet:

2015: Backend-Dienste. Viele Apps greifen auf standardisierte Clouddienste zu, um ihre Dienste zu erbringen. Sicherheitslücken betreffen potenziell sämtliche in der Cloud hinterlegten Daten. Daher haben wir geprüft, wie die Clouddienste eingebunden wurden und ob/wie sie sich in das Sicherheitskonzept der Anwendung integrieren.

2016: Smart Home. Mit Smart-Home-Anwendungen lassen sich Haushaltsgeräte wie Steckdosenleisten, Thermostate, Rollläden, usw., aber auch Überwachungskameras fernsteuern. Sicherheitslücken in diesem Bereich ermöglichen Einblicke in private Lebensumstände, erleichtern Einbrüche oder können Menschen unmittelbar gesundheitlich schädigen.

2016: Antivirus. Antivirensoftware soll Computer und Smartphones vor Schadsoftware schützen. Sicherheitslücken in diesen Anwendungen können den vermeintlichen Schutz jedoch unerkannt deaktivieren. Sie können sogar so ausgenutzt werden, dass über die Antivirensoftware selbst Schadcode eingeschleust wird.



2017: Passwortmanager. Passwörter speichern zu können ermöglicht Benutzern, unterschiedliche sichere Passwörter für verschiedene Dienste verwenden zu können. Auf der anderen Seite ermöglichen erfolgreiche Angriffe auf Passwortmanager den Zugriff auf alle diese Passwörter und Dienste.

2018: Tracking-Apps. Mit Trackern können Eltern den Standort ihrer Kinder kontrollieren oder Freunde sich auf Karten finden. Sicherheitslücken in solchen Anwendungen ermöglichen den Zugriff auf Standortdaten und die Bildung von Bewegungsprofilen der betroffenen Benutzer auch ohne Einverständnis.

2019: VoIP-Telefone. Moderne Telefone übertragen Sprachdaten über das Internet und benötigen keine gesonderten Telefonleitungen mehr. Hierdurch werden aus Telefonen jedoch IoT-Geräte, die potenziell von Angreifern übernommen werden können, um Sprachdaten zu extrahieren und damit Gespräche abzuhören.

Das TeamSIK-Projekt aus 2020 ist noch nicht abgeschlossen, da noch nicht alle Lücken an die jeweiligen Hersteller gemeldet werden konnten. Entsprechend können wir an dieser Stelle nicht auf Details eingehen. Allerdings lassen sich die Erkenntnisse aus früheren Projekten auf das aktuelle Projekt übertragen, d.h. wir erhalten dasselbe Gesamtbild.

Während diese Projekte keinen repräsentativen Überblick über Software im Allgemeinen geben, bieten sie einen Einblick in ausgewählte Bereiche, speziell in mobile Anwendungen und eingebettete Systeme und die dazugehörigen Dienste. In diesem Whitepaper betrachten wir die Entwicklung häufig vorkommender Sicherheitslücken aus diesen Projekten.

3 Sicherheitslücken aus vier Jahren

Abbildung 1 zeigt die häufigsten Schwachstellen aus den verschiedenen Projekten. Im Folgenden gehen wir auf die einzelnen Kategorien ein. Bereits in der Abbildung erkennt man wiederkehrende Kategorien, die über unterschiedliche Projekte gleichermaßen zu finden waren, auch wenn die jeweiligen Anwendungen aus unterschiedlichen Domänen stammten und mit mehreren Jahren zeitlichem Abstand analysiert wurden.

- **Zugangsdaten im Code hinterlegt**. Bei dieser Kategorie von Schwachstellen werden Zugangsdaten zu Backend-Systemen wie Clouddiensten oder Datenbanken im Code der Clientanwendung hinterlegt. Dort können sie von einem Angreifer extrahiert werden, der sich anschließend mit diesen Zugangsdaten beim entsprechenden Backend-System anmelden kann. Diese Schwachstelle offenbart das fundamentale Missverständnis, statt eines Benutzers eine Anwendung zu authentifizieren. Dies ist jedoch nicht möglich, wenn sich die Anwendung unter der Kontrolle eines Angreifers befindet, z.B. auf einem Clientsystem / Mobilgerät in nicht vertrauenswürdiger Umgebung.

- **Konfigurationsfehler im Backend.** Bei dieser Kategorie von Sicherheitslücken interagiert eine Anwendung mit einem Clouddienst oder einer Datenbank, wobei die Sicherheitseinstellungen dieses Backend-Dienstes unzureichend sind.

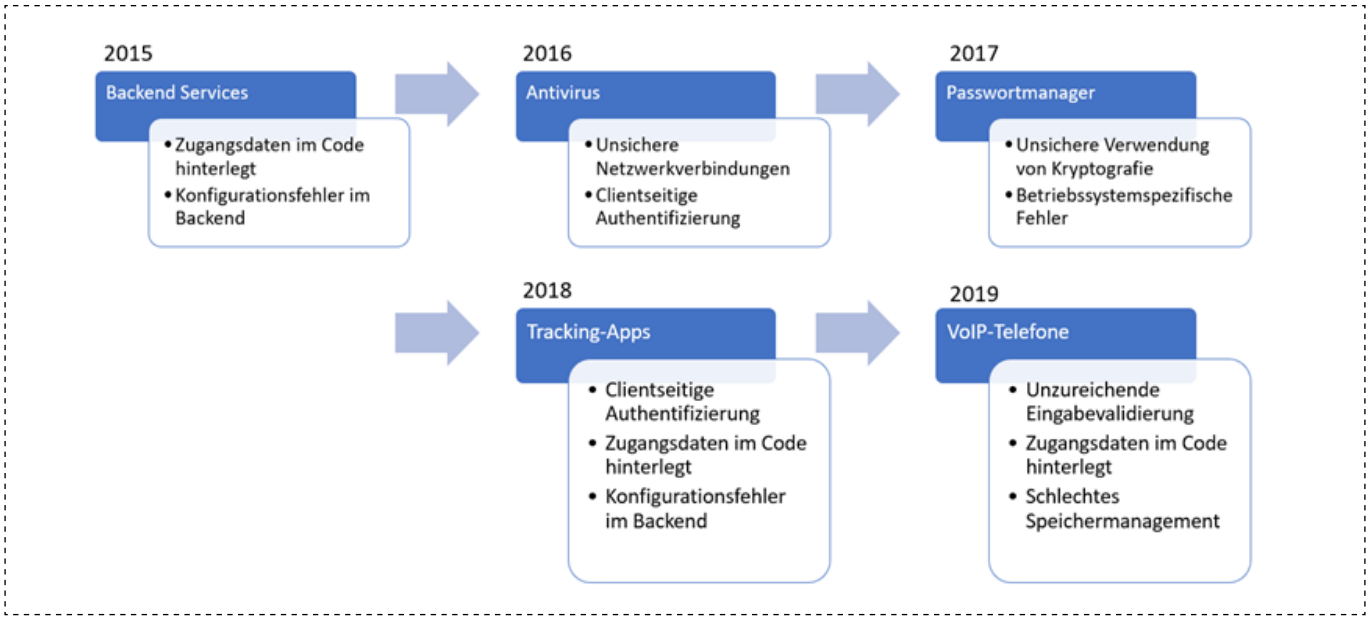


Abbildung 1: Häufigste Sicherheitslücken aus den Projekten

Im Extremfall ist ein Zugriff auf alle dort gespeicherten Daten ohne Authentifizierung möglich. In Kombination mit der vorherigen Schwachstellenkategorie kann auch der in der App hinterlegte Benutzer mit zu vielen Berechtigungen ausgestattet sein, z.B. dem Recht, gespeicherte Dateien zu lesen, obwohl die Anwendung den Dienst nur zum Hochladen von Dateien verwendet.

- **Unsichere Netzwerkverbindungen.** Kommuniziert eine Anwendung mit einem anwendungsspezifischen Backend oder einem Clouddienst, muss diese Datenübertragung adäquat gesichert werden. Schwachstellen schließen die Verwendung von http-Verbindungen ohne TLS, aber auch die fehlerhafte Verwendung von TLS ein, indem z.B. der Aussteller des vom Server präsentierten Zertifikats nicht geprüft wird.

- **Clientseitige Authentifizierung.** Die Schwachstellen in dieser Kategorie sind dadurch charakterisiert, dass sicherheitskritische Funktionen wie die Überprüfung der Benutzerberechtigungen im Client und damit unter der Kontrolle des Angreifers durchgeführt werden. Ein Angreifer kann die Anwendung so manipulieren, dass die Berechtigungsprüfung entfällt, oder die entsprechenden Befehle unabhängig von der Anwendung an den Server senden.

- **Unsichere Verwendung von Kryptografie.** Anwendungen verwenden kryptografische Funktionen, um Daten zu verschlüsseln oder zu signieren. Werden die Funktionen falsch genutzt, z.B. indem Schlüssel im Anwendungscode hinterlegt sind oder aus nicht zufälligen Daten abgeleitet werden oder veraltete Algorithmen zum Einsatz kommen, werden die entsprechenden Schutzziele nicht erreicht. Auch die Verwechslung von Verfahren (z.B. Einsatz einer Verschlüsselung, wenn eigentlich ein Integritätsschutz gewährleistet werden soll) ist eine häufige Sicherheitslücke.

- **Betriebssystemspezifische Fehler.** In diese Kategorie fallen alle Sicherheitslücken, die auf der falschen Verwendung plattformspezifischer APIs und Konzepte beruhen. Auf mobilen Plattformen zählen dazu die Mechanismen zur Isolation von Apps oder zur Speicherung kryptografischer Schlüssel in sicheren Hardwaremodulen.

- **Unzureichende Eingabevalidierung.** Diese Kategorie umfasst alle Schwachstellen, die aufgrund fehlender oder unvollständiger Validierung von Eingabedaten aus nicht vertrauenswürdiger Quelle vor der Verarbeitung entstehen. SQL Injections oder Cross Site Scripting sind häufige Vertreter dieser Kategorie. Solche Sicherheitslücken treten meist in anwendungsspezifischen Backend-Diensten, aber auch in Datenbankkomponenten von Android-Apps auf.

- **Schlechtes Speichermanagement.** Nativer Code greift direkt auf den Arbeitsspeicher zu. Fehlerhafte Prüfungen der allokierten Speicherbereiche und ihrer Grenzen können Angreifern ermöglichen, Speicherbereiche zu überschreiben, die normalerweise der Anwendung vorbehalten sein sollten. Alternativ können Lesezugriffe außerhalb der eigentlich vorgesehenen Speicherbereiche einem Angreifer sensible Informationen bereitstellen.

Allen diesen Kategorien von Sicherheitslücken, die wir in den vergangenen vier Jahren in verschiedensten Anwendungen identifiziert haben, ist gemein, dass sie bekannt und in Standards dokumentiert sind. Die Common Weakness Enumeration (CWE) listet beispielsweise die folgenden Kategorien auf (übersetzt aus dem Englischen):

- Nr. 3: Unzureichende Eingabevalidierung
- Nr. 18: Unzureichend geschützte Zugangsdaten
- Nr. 20: Im Code hinterlegte Zugangsdaten
- Nr. 22: Unzureichendes Berechtigungsmanagement
- Nr. 24: Fehlende Authentifizierung für kritische Funktionen
- Nr. 25: Fehlende Autorisierung

Somit ergibt sich das Bild, dass zahlreiche sicherheitskritische Anwendungen elementare Sicherheitslücken enthalten, die bei einer Sicherheitsprüfung nach standardisierten Katalogen hätten gefunden werden müssen. Auch ist über die vier Jahre Sicherheitsanalysen hinweg nicht erkennbar, dass die Anzahl oder der Schweregrad der erkannten Sicherheitslücken abgenommen hätte. Aufgrund des begrenzten Umfangs der TeamSIK-Projekte lässt sich keine repräsentative Aussage zu Software im Allgemeinen treffen. Da es sich bei vielen Lücken jedoch um vermeidbare Fehler handelt, stellt sich dennoch die Frage, wie zumindest elementare Lücken in Zukunft vermieden werden können.

4 Automatische Sicherheitstests

Jeder Schritt eines Entwicklungsprozesses sollte durch Maßnahmen für IT-Sicherheit flankiert werden. Parallel zu den funktionalen Anforderungen sollten z.B. auch mögliche Bedrohungen analysiert und Sicherheitsanforderungen abgeleitet werden. Die später erstellte Anwendungsarchitektur muss diesen Anforderungen genügen bzw. ermöglichen und im besten Fall erleichtern, eine Implementierung zu schaffen, die die Anforderungen erfüllt.
Die zuvor gezeigten Kategorien von Sicherheitslücken betreffen einerseits die Architektur (z.B. clientseitige Authentifizierung), andererseits die konkrete Implementierung (z.B. unzureichende Eingabevalidierung). In diesem Whitepaper konzentrieren wir uns auf Implementierungsschwächen. Dabei ist zu beachten, dass sich anhand der Implementierung auch teilweise konzeptionelle Fehler erkennen lassen. So können im Code hinterlegte Zugangsdaten für eine Backend-Datenbank bspw. auf eine clientseitige Implementierung des gesamten Datenbankzugriffs (und damit potenziell auch der Authentifizierung) hindeuten. Der Analyst kann die hinterlegten Zugangsdaten testen und so z.B. eine Fehlkonfiguration des Backends erkennen, wenn dem extrahierten Benutzerkonto zu viele Berechtigungen zugeteilt wurden. Eine solche a-posteriori-Analyse ersetzt selbstverständlich keinen adäquaten Entwicklungsprozess. Werden Sicherheitslücken erst im Code erkannt, ist der Aufwand zur Behebung etwaiger konzeptioneller Probleme deutlich größer als bei einer Anpassung in der Design- und Modellierungsphase. Somit dient die Prüfung der Implementierung einerseits als Test auf Umsetzungsschwächen, andererseits als „letzte Chance“ zur Erkennung mancher Arten von Designproblemen, die in vorherigen Schritten nicht identifiziert wurden. Eine solche Prüfung anhand des Codes kann auch geboten sein, wenn bestehender Code ohne vorhandene Dokumentation oder Modellierung weiterentwickelt werden soll und ein erster Überblick über den Sicherheitszustand erforderlich ist.

Moderne Anwendungen sind oftmals zu groß, um sie mit einen manuellen Code Review vollständig, Zeile für Zeile, zu begutachten. Zudem erfordert eine solche manuelle Prüfung ein großes Maß an Expertenwissen nicht nur über IT-Sicherheit im Allgemeinen, sondern auch über die verwendete Programmiersprache und Plattform. Da Anwendungen gerade im Mobilbereich häufig aktualisiert werden, und eine Überprüfung bei jeder Änderung wiederholt werden müsste, ist ein solcher Aufwand nicht zu leisten. Abhilfe schaffen automatisierte Schwachstellenscans.

Automatisierte Codeanalysen lassen sich in statische und dynamische Ansätze unterteilen. Während die statische Codeanalyse den (binären) Code einer Anwendung betrachtet, führt die dynamische Analyse die Anwendung aus und beobachtet sie zur Laufzeit. Beide Ansätze haben ihre jeweiligen Vor- und Nachteile. Die statische Analyse kann den gesamten Code betrachten und ist nicht darauf angewiesen, dass eine bestimmte Funktion tatsächlich ausgeführt wird. Auf der anderen Seite können dynamische Analysen nur das tatsächlich erreichte Verhalten begutachten, d.h. sie müssen möglichst alle Funktionen in allen Konfigurationen auslösen, was in der Praxis nicht vollständig möglich ist. Dafür bieten dynamisch erkannte Schwachstellen eine erhöhte Konfidenz für die Existenz einer Schwachstelle. Da das Verhalten tatsächlich zur Laufzeit beobachtet wurde, wird die Wahrscheinlichkeit einer Falschmeldung deutlich reduziert.

Präzision bei Werkzeugen für Codeanalyse

Bei einer Codeanalyse ist es essenziell, die spezifische Semantik der jeweiligen Plattform zu betrachten. Obwohl Java als Programmiersprache sowohl für Desktopanwendungen als auch für Android-Apps genutzt wird, unterscheiden sich die Eigenschaften der beiden Plattformen, die verfügbaren Frameworks und APIs und damit auch die Anforderungen an den Code. Während eine Android-App aus Komponenten wie Activities und Services besteht, die über Intents Daten austauschen können, stehen bei einer Java-Webanwendung (z.B. implementiert über Spring Boot) Servlets und Beans im Vordergrund, die http-Anfragen bearbeiten. Entsprechend sind auch die Konfigurationen für den Zugriffsschutz sehr unterschiedlich, genauso wie die impliziten und expliziten Datenflüsse zwischen den einzelnen Komponenten. Versendet eine Android-App beispielsweise einen Intent ohne Angabe der Zielkomponente (impliziter Intent), wird das Ziel vom Betriebssystem identifiziert, wodurch auch nicht geplante, potenziell unerwünschte Anwendungen, den Intent erhalten können. Dies ist nicht ersichtlich, wenn die Analyse die Android-App rein als Java-Code analog zu einem Desktopprogramm betrachtet. Eine Codeanalyse kann nur dann präzise sein und möglichst viele Sicherheitslücken identifizieren, wenn sie die Semantik der jeweiligen Plattform möglichst exakt unterstützt und die Semantik der Anwendung im Kontext der Plattform rekonstruiert.

Die Verwendung von Drittanbieterkomponenten stellt eine weitere Herausforderung für Codeanalysesysteme dar. Solche Komponenten sind nicht immer im Quellcode verfügbar. Werden Anwendungen von externen Firmen im Auftrag entwickelt, wird ebenfalls oftmals kein Quellcode übergeben, speziell wenn der Auftragnehmer für verschiedene Kunden ähnliche Anwendungen erstellt oder seine Produkte auf denselben proprietären Frameworks basieren. Eine Herausgabe solcher zentralen Komponenten wäre im Sinne des Schutzes geistigen Eigentums und wirtschaftlicher Interessen für den Auftragnehmer nicht durchführbar bzw. mit erheblichen Kosten für den Auftraggeber verbunden. In allen genannten Fällen besteht für den Auftraggeber bzw. Entwickler die Notwendigkeit, Anwendungen zu analysieren, für die zumindest in Teilen kein Quelltext vorliegt. Während eine dynamische Analyse ohne Änderungen auch solche Anwendungen ausführen kann, ist eine statische Analyse nur dann einsetzbar, wenn sie den Binärcode analysiert. Dies schränkt die Auswahl der verfügbaren Werkzeuge ein. Gleichzeitig bietet ein rein dynamischer Ansatz aufgrund der begrenzten Codeabdeckung bei komplexer Software kein vollständiges Bild des Sicherheitsstatus' der Anwendung.

Um sinnvoll in Entwicklungsprozesse integrierbar zu sein, müssen Codeanalysewerkzeuge hinreichend prä zise sein. Jede Meldung des Werkzeugs muss vom Entwickler bzw. einem Sicherheitsexperten überprüft wer-

den. Liefert der Scanner zu viele Falschmeldungen, bindet die Prüfung dieser Vielzahl an Meldungen Kapazitäten, die für die normale Entwicklungsarbeit nicht mehr zur Verfügung stehen. Zudem reduzieren Falschmeldungen die Akzeptanz des Scanners beim Entwicklungsteam und das Vertrauen in zukünftige Meldungen.

Zusammenfassend können Codeanalysetechniken dazu beitragen, Software bei jeder Änderung am Code automatisiert auf Sicherheitslücken zu prüfen. Allerdings stellen Entwicklungsprojekte diverse Anforderungen an den Codescanner, speziell bzgl. nicht vorhandenem Quellcode und Falschmeldungen.

5 Der Schwachstellenscanner VUSC

Das Fraunhofer SIT entwickelt den statischen Schwachstellenscanner VUSC (für engl. VUlnerability SCanner). VUSC scannt den Binärcode von Anwendungen auf Sicherheitslücken. Dadurch ist nur die Anwendung in kompilierter Form erforderlich, wie sie beim Kunden ausgeführt wird, weitere Zulieferungen durch den Entwickler oder Auftragnehmer sind nicht erforderlich. Damit ist VUSC auch für die Prüfung von eingekauften oder im Auftrag entwickelten Anwendungen einsetzbar. Auch Drittanbieterkomponenten, für die kein Quelltext zur Verfügung steht, sind damit kein Problem für VUSC.

Derzeit unterstützt VUSC die Analyse von mobilen Apps für Android und iOS sowie von Desktopanwendungen für Java, unabhängig davon, ob die Anwendung in Java oder einer anderen kompatiblen Sprache wie Scala entwickelt wurde. Zudem werden Java-Web-Start-Anwendungen und Backend-Dienste für Java Enterprise und Spring Boot unterstützt. In Übereinstimmung mit den im vorherigen Kapitel entwickelten Anforderungen bietet VUSC für alle diese Plattformen spezifische und präzise Modelle, welche die Semantik der jeweiligen Plattform abbilden.

Abbildung 2 zeigt den Ablauf einer Codeanalyse in VUSC. Zuerst wird der Binärcode (z.B. als APK-Datei bei einer Android-App oder einer WAR-Datei für eine Java-Webanwendung) eingelesen und in eine Zwischendarstellung transformiert. Diese erste Form der Darstellung bietet ein niedriges Abstraktionsniveau. Bei Java-basierten Anwendung werden Klassen und Methoden dargestellt, bei iOS-Apps Instruktionen in Assemblercode. Im nächsten Schritt verwendet VUSC die Plattformdefinitionen, um semantisch höhere Modelle zu erzeugen. In Android-Apps werden Komponenten wie Activities und Services rekonstruiert, während in Webanwendungen Servlets und Beans erstellt werden. VUSC rekonstruiert die Domäne des ursprünglichen Entwicklers, wodurch spätere Schritte des Codescans auf denselben Konzepten aufbauen können, mit denen auch der Entwickler (und Softwarearchitekt) gearbeitet hat. Eine solche Erhöhung des Abstraktionsniveaus erleichtert nicht nur die Definition von Sicherheitsanalysen, sondern ermöglicht es auch, Analysen sehr viel präziser zu formulieren und so Falschmeldungen zu vermeiden.

Im nächsten Schritt berechnet VUSC Datenflüsse. In diesem Schritt wird ermittelt, ob sensitive Daten an nicht vertrauenswürdige Senken weitergegeben werden (Datenschutzverstöße), oder ob nicht vertrauenswürdige Eingabedaten für Operationen verwendet werden (Injection-Angriffe). Die Datenflussanalyse basiert auf dem Systemmodell. Flüsse werden nicht von bspw. einer bestimmten Instruktion im Assemblercode zu einer anderen Instruktion gesucht, sondern z.B. von einem Passwortfeld in einer Benutzerschnittstelle zu einer Netzwerkverbindung. Dieser semantisch reichhaltige Ansatz ermöglicht die präzise Definition von Schwachstellenmustern, wodurch Falschmeldungen vermieden wer
den. Zudem ermöglicht das Systemmodell den Zugriff auf Detaildaten, z.B. die URL der Netzwerkverbindung. So kann auch zwischen sicherer (TLS) und unsicherer Datenübertragung unterschieden werden. Die einzelnen

Sicherheitsprüfungen basieren auf dem Systemmodell und den Datenflüssen. Neben den bereits genannten Beispielen können so auch plattformspezifische Fehler, z.B. die unsichere Verwendung von Intents zur Kommunikation zwischen den einzelnen Komponenten, erkannt werden. Hier geht VUSC über eine reine Betrachtung von z.B. Java-Code hinaus und interpretiert eine Android-App im Kontext von Android und eine Webanwendung im Kontext eines Java-basierten Anwendungsservers. Somit ermöglicht VUSC eine hohe Präzision und kann darüber hinaus zahlreiche kontextabhängige Detailinformationen liefern (welche Datei, welcher Server, welcher kryptografische Algorithmus, …), die dem Analysten oder Entwickler die Einschätzung bzw. Behebung der Schwachstelle erleichtern.

Abbildung 2: Ablaufschema der VUSC-Analyse

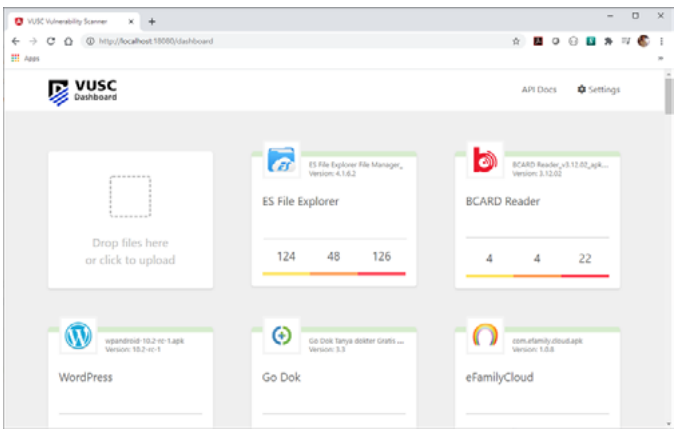


Abbildung 3: VUSC Dashboard

Abbildung 3 zeigt das Dashboard von VUSC. Der Benutzer zieht die binäre Anwendungsdatei (APK, IPA, JAR, WAR, EAR, usw.) per Drag&Drop auf die freie Kachel links oben. VUSC erzeugt anschließend eine neue Kachel für die Anwendung. In der Übersicht jeder Kachel befindet sich die Anzahl der Schwachstellen sortiert nach Schweregrad (niedrig, mittel hoch). Mit einem Klick auf die Kachel gelangt der Benutzer zur Detailansicht der jeweiligen Anwendung, dargestellt in Abbildung 4.

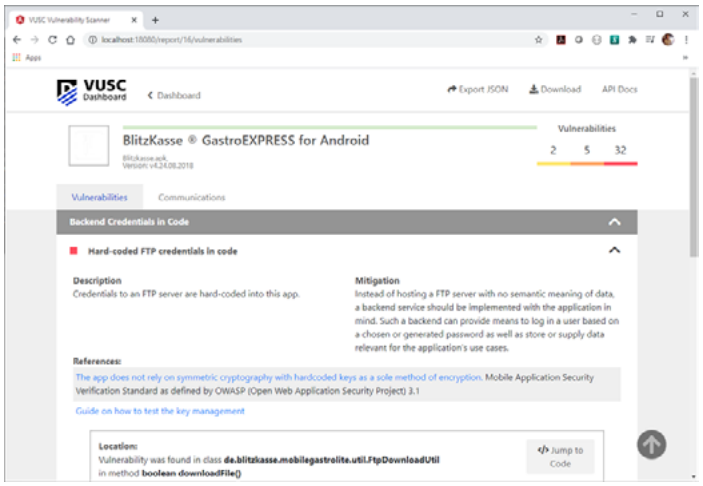


Abbildung 4: Detailansicht zu einer Anwendung

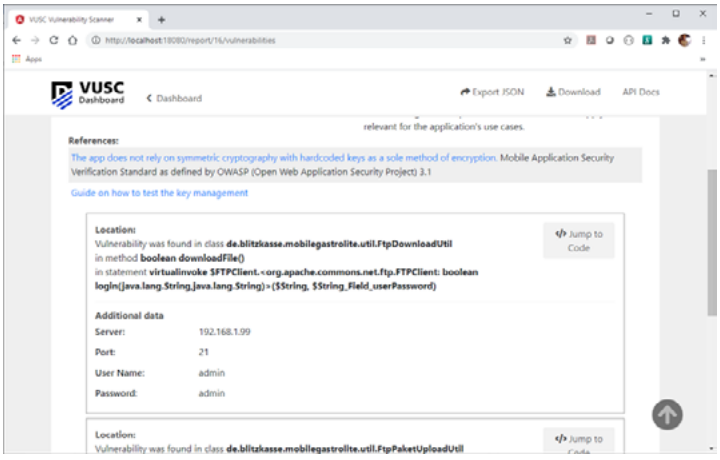


Abbildung 5: Zusatzdaten zu einer Schwachstelle

Alle Schwachstellen sind in Kategorien einsortiert. Im Beispiel werden Lücken angezeigt, die sich durch in der Anwendung hinterlegte Zugangsdaten ergeben, z.B. als Benutzername und Passwort für einen FTP-Dateiaustauschdienst. Zu jeder Schwachstelle werden eine Be schreibung des jeweiligen Problems, ein Vorschlag zur Behebung des Problems und Referenzen auf weiterführende Informationen und öffentliche Kataloge wie z.B. OWASP angezeigt. Außerdem wird angezeigt, an welcher Codestelle die Schwachstelle gefunden wurde. Bei Java- und Android-Anwendungen werden der Name der Klasse und Methode angezeigt, bei iOS-Anwendungen der Offset innerhalb der Binärdatei.
Zudem werden Zusatzinformationen zur Schwachstelle angezeigt, wie in Abbildung 5 dargestellt. Im Falle der hinterlegten Zugangsdaten für den FTP-Server zeigt VUSC an, um welchen Server es sich handelt und welcher Benutzername und welches Passwort hinterlegt sind. Hierdurch kann der Analyst direkt erkennen, welches Benutzerkonto er auf dem Server sperren muss, da diese Zugangsdaten nicht mehr als sicher gelten können. Ebenso können die Kontextinformationen helfen, die Kritikalität des Sicherheitsproblems einzuschätzen, wenn dem Entwickler z.B. bekannt ist, welche Daten (auch von anderen Anwendungen) auf diesem FTP-Server gespeichert werden und nun potenziell von Angreifern ausgelesen werden können.

Abbildung 6: DFarm-Rack

VUSC ist als Produkt des Fraunhofer SIT verfügbar und wird lokal beim Kunden ausgeführt. Hierdurch verbleiben die Informationen über Schwachstellen beim Kunden und können nicht von einem Cloudanbieter eingesehen werden.

In unserer Forschung am Institut arbeiten wir an der Kombination statischer und dynamischer Analysen, um die von VUSC identifizierten Sicherheitslücken zukünftig automatisiert verifizieren zu können. Hierdurch soll eine noch bessere Priorisierung der gefundenen Sicherheitslücken erreicht werden. Als Prototyp und Open-Source-Projekt ist bereits unsere Plattform DFarm verfügbar, mittels welcher dynamische Analysen auf eine Vielzahl von Geräten verteilt werden können. Hierdurch wird die Skalierung dynamischer Analysen über hunderte Apps und Geräte verbessert. Abbildung 6 zeigt einen Aufbau für reale Geräte bei einer Massenanalyse in Fällen, in denen keine Verwendung eines Emulators möglich ist. Eine Integration in unser kommerzielles Produkt VUSC erfolgt in Kürze.

6 Zusammenfassung

In diesem Whitepaper wurden die häufigsten Arten von Sicherheitslücken aus den Projekten der TeamSIK-Offensive-Security-Gruppe am Fraunhofer SIT analysiert. Diese Schwachstellen entsprechen bekannten Programmierfehlern, z.B. aus der CWE-Liste. Auch wenn die Auswahl der Schwachstellen nicht repräsentativ für Softwaresicherheit im Allgemeinen ist, wurde sichtbar, dass dieselben Arten von Schwachstellen über verschiedene Arten von Anwendungen und unterschiedliche Jahre auftreten. Somit ist keine Abnahme in Anzahl oder Kritikalität der Schwachstellen erkennbar – zumindest bzgl. der untersuchten Anwendungen aus verschiedenen Domänen (Passwortmanager, Antivirus, Smart Home, etc.).

Des Weiteren wurde erläutert, wie Werkzeuge zur statischen und dynamischen Codeanalyse während der Entwicklung eingesetzt werden können, um solche wiederkehrenden Arten von Sicherheitslücken zu vermeiden bzw. vor Veröffentlichung der Software zu beheben.

Beim Einsatz dieser Werkzeuge treten jedoch diverse Herausforderungen auf, insbesondere bzgl. Präzision und dem Umgang mit Anwendungen, für die ganz oder partiell kein Quellcode bereitsteht.

Das Fraunhofer SIT bietet mit VUSC einen eigenen Codescanner an, der mit neuartigen Analysetechniken binäre Anwendungen für diverse Plattformen auf Schwachstellen untersuchen kann. Durch die semantisch reichen Modelle für alle unterstützten Plattformen können Analysen in der Semantik der ursprünglichen Anwendungsdomäne (z.B. Android oder Java Enterprise) formuliert werden, was eine sehr präzise Beschreibung zu der zu erkennenden Schwachstellen ermöglicht und so Falschmeldungen vermeidet.

Zudem extrahiert VUSC für jede Schwachstelle Detailinformationen, wie bspw. die verwendete kryptografische Funktion, den Pfad und Namen der betroffenen Datei oder die URL des entfernten Servers.

Secure Software Engineering am Fraunhofer SIT

Das Fraunhofer SIT unterstützt Firmen bei der sichere Entwicklung von Software über den gesamten Lebenszyklus der Produkte hinweg. Gemeinsam mit unseren Kunden führen wir systematische Bedrohungsmodellierungen und Risikoabschätzungen durch, analysieren und optimieren System- und Softwarearchitekturen hinsichtlich IT-Sicherheit, empfehlen geeignete Maßnahmen zur Qualitätssicherung während der Entwicklungsphase, und führen Penetrationstests und Code Reviews durch. Hierfür verwenden wir nicht nur standardisierte Engineering-Methoden, sondern entwickeln diese stetig
weiter.

VUSC - der Codescanner ist eine Eigenentwicklung des Fraunhofer SIT, die innerhalb von Minuten Software auf Sicherheit hin prüfen kann. Im Gegensatz zu anderen Softwarescannern benötigt VUSC keinen Quellcode. Weiterer Pluspunkt: VUSC arbeitet on premises und nicht in der Cloud, dadurch haben Nutzer jederzeit volle Kontrolle über ihre Daten.VUSC findet nicht nur Sicherheitslücken, sondern gibt zu jeder Schwachstelle eine allgemein verständliche Beschreibung des Problems. Darüber hinaus erstellt VUSC automatisch eine Klassifizierung der Schwachstellen in hohes, mittleres oder niedriges Risiko.