Softwaresicherheit

Zusammenfassung

Gruppenmitglieder: Julian Hinxlage (7010922)
Edgar Schkrob (7012009)
Johannes Theiner (7010923)

Semester: Wintersemester 2019/2020

letzte Änderung: 12. Januar 2020

Inhaltsverzeichnis

1 statische Code Analyse **2**
1.1 Code Metriken . . . 2
1.2 Visualisierung von Messwerten . . . 3
1.3 Analysen . . . 3
1.4 Lösungen . . . 5

2 Cryptographic Libraries **6**
2.1 Motivation . . . 6
2.2 Bibilotheken . . . 6
2.3 Fazit . . . 7

3 FIDO2 **8**
3.1 Motivation . . . 8
3.2 Begriffe . . . 8
3.3 Unterstützung . . . 9
3.4 Fazit . . . 9

4 Safety **10**
4.1 Software Assurance . . . 10
4.2 Robustheit und Abhängigkeit . . . 10
4.3 Fault Isolation . . . 11
4.4 Schutz von Serveranlagen . . . 11
4.5 Fazit . . . 11

5 Random Numbers **12**
5.1 Was sind Zufallszahlen ? . . . 12
5.2 Physikalische Prozesse . . . 12
5.3 ERNIE . . . 12
5.4 /dev/random . . . 12
5.5 Pseudozufallszahlen . . . 12
5.6 Implementationen . . . 13
5.7 Anwendungsgebiete . . . 14
5.8 Verschlüsselungsverfahren . . . 14

6 Threat Modeling Tool **16**
6.1 MS Security Development Lifecycle (SDL) . . . 16
6.2 Threat Modelling . . . 16
6.3 Microsoft Threat Modeling Tool . . . 17
6.4 Fazit . . . 17

7 RESTful API Authentication (Token based) **18**
7.1 RESTful APIs . . . 18

7.2 Authentifizierungsverfahren . . . 18
7.3 JSON Web Tokens . . . 19
7.4 Autorisierungsprozess . . . 19

8 Key Sharing **20**
8.1 Was ist Key Sharing ? . . . 20
8.2 Anwendungsbereich . . . 20
8.3 Beispiel . . . 20
8.4 Probleme bei diesem Fall . . . 20
8.5 Blakley's secret sharing . . . 21
8.6 Shamir's secret sharing . . . 21

9 Linux Hardening **22**
9.1 Hardware . . . 22
9.2 Nutzer . . . 22
9.3 Software . . . 23
9.4 Netzwerk . . . 23
9.5 Fazit . . . 23

10 Codesigning **24**
10.1 Signatur . . . 24
10.2 Unterschiede bei Betriebssystemen . . . 24

1 statische Code Analyse

Bei der statischen Code Analyse kann manuell oder automatisiert überprüft werden ob ein System den Qualitätsanforderungen entspricht.

1.1 Code Metriken

Metriken geben dem Entwickler mehr Einsicht in den entwickelten Code. So können zu komplexe Stellen im Code und Risiken früh erkannt werden.

1.1.1 klassische Metriken

LOC (lines of code) Die Anzahl an Code Zeilen innerhalb einer Methode besitzt eine starke Korrelation mit dem Aufwand der betrieben werden muss um diesen Code zu erstellen und warten.

Aufgrund verschiedener Programmierstiele und Programmiersprachen ist diese Metrik stark umstritten.

NCSS (non-commenting source statements) zu große Methoden und Klassen sind schwer zu lesen und teuer zu pflegen. Häufig hat eine solche Methode/Klasse zu viele Aufgaben oder Funktionen.

Diese Metrik sollte nach Möglichkeit klein gehalten werden um den Code übersichtlich zu halten.

LOC und NCSS werden häufig in Zusammenhang gebracht um den Anteil der Dokumentation festzustellen.

Halstead-Metriken Mit diesen Metriken können mehrere Kenngrößen berechnet und interpretiert werden. Operatoren und Operanden werden hierbei analysiert und mit Erfahrungswerten verglichen um unter anderem folgende Metriken zu erhalten: Schwierigkeit, benötigte Zeit der Entwicklung, Anzahl gelieferter Fehler.

McCabe-Metrik oder auch Zyklomatische Komplexität gibt an wie viele linear unabhängige Pfade innerhalb des gegeben Quellcodes enthalten sind.

1.1.2 Objektorientierte Metriken

Komponentenmetriken Bei dieser Metrik werden einzelne Bestandteile separat bewertet. Im folgende sind einige Metriken aufgelistet.

- Anzahl Klassenvariablen
- Anzahl Objektvariablen
- Anzahl Attribute
- Vererbungstiefe
- Anzahl überschriebener Methoden

Strukturmetriken Hier wird die Klassenvererbung als ganzes auf Interaktions- und Vererbungsmuster analysiert.

- Anzahl zugreifender Klassen
- Anzahl zugegriffener Klassen

1.2 Visualisierung von Messwerten

Mit Pareto- und Streudiagrammen können zwei Metriken auf Verbindungen untersucht werden, während in einem Kiviat-Diagramm das Zusammenspiel von 5 Metriken beobachtet werden kann.

1.3 Analysen

1.3.1 Konformitätsanalyse

Bei dieser Analyse wird der Quellcode auf Einhaltung von gegebenen Regeln überprüft.

Syntax Analyse Die Syntax einer Programmiersprache definiert die verwendbaren Konstrukte. Diese Analyse überprüft somit ob alle verwendeten Konstrukte gültig sind.

Lex Scanner Mit dem Programm Lex können Programme aufgrund eines Lexikons auf Korrektheit überprüft werden.

Yacc Parser Dieses Programm erzeugt aus der Syntaxdefinition einer Programmiersprache automatisch einen Parser der Programme auf korrekte Syntax überprüfen kann.

Semantik-Analyse Semantik beschreibt die inhaltliche Bedeutung des Quellcodes. Fehleranfällige oder fragwürdige Konstrukte können mit Analysen aus dieser Kategorie identifiziert werden.

lint Ein Linter überprüft Code auf Programmierfehler, Bugs, schlechten Stiel und fragwürdigen Code.

1.3.2 Exploit Analyse

Syntaktische Analyse Bei dieser Analyse wird der Code auf Schlüsselwörter durchsucht. Dies können etwa sein:

- Verwendung anfälliger Funktionen
- Verwendung ungültiger Datentypen

Semantische Analyse Hierbei wird der Fluss des Code analysiert. Dabei können unter anderem folgende Probleme identifiziert werden:

- Übernahme von Benutzereingaben ohne Validierung
- Verwendung uninitialisierter Variablen

1.3.3 Anomalieanalyse

Eine Anomalie ist ein Zustand der von dem aufgrund von Spezifikation und Normen erwartetem abweicht.

Kontrollflussanomalie Sie bezeichnet Unstimmigkeiten im Ablauf von Programmteilen wie etwa mehrere Rückgabestellen aus einer Methode oder Sprünge aus Schleifen hinaus. Dies entspricht nicht der strukturierten Programmierung muss aber nicht fehlerhaft sein.

Datenflussanomalie Beispiele für diese Anomalie ist das Verwenden einer Variable ohne diese zuvor intialisiert zu haben oder mögliche Werte die nicht abgedeckt sind.

1.4 Lösungen

1.4.1 Manuelle Software-Prüfung

Walkthroughs Während eines Walkthroughs gehen mehrere Entwickler gemeinsam durch Code und stellen Fragen um Probleme zu finden.

Reviews Hierbei wird bevor neuer Code produktiv eingesetzt wird dieser von anderen Entwicklern auf Qualität, Fehler und Sicherheitslücken überprüft.

Inspektion Abweichungen von zuvor erstellten Spezifikationen werden bei dieser Methode festgehalten und wird verschiedene Gruppen unterteilt.

1.4.2 Tools

Viele Tools nutzen einen von zwei Ansätzen. Entweder wird der Code auf Einhaltung gegebener Standards überprüft was vorbeugend wirkt und Fehlerwahrscheinlichkeit und Wartungsaufwand senkt. Oder der Code wird auf tatsächliche und mutmaßliche Fehler untersucht.

Daraus folgendend sollten mehrere Tools verwendet werden da ein einzelnes nicht das volle Spektrum abdeckt.

2 Cryptographic Libraries

2.1 Motivation

Häufig liegt die Schwäche eines Systems nicht in den Grundlagen oder dem Konzept, sondern in der konkreten Implementierung. Um Fehler zu vermeiden sollte daher eine Bibliothek verwendet werden.

2.1.1 Anforderungen

Aus dem BSI[1] Projekt „Sichere Implementierung einer allgemeinen Kryptobibliothek“ resultieren folgende Anforderungen aus Sicht des Programmierers:

- Ressourcen schonend
- Sicherheit bei sauber Implementation gewährleistet
- leicht benutzbar
- (selbsterklärende) Dokumentation
- immer auf dem neuesten Stand
- keine Eigenkreation

2.2 Bibilotheken

2.2.1 Bouncy Castle

Eine Bibliothek die seit dem Jahr 2000 für Java und C# zur Verfügung steht ist Bouncy Castle. Diese Bibliothek bietet eine Fülle an Funktionen welche:

- nicht übersichtlich organisiert sind
- schlecht dokumentiert sind
- und Implementierungen vom Programmierer fordern

Weiterhin sind viele der enthaltenen Methoden veraltet oder bereits gebrochen was kryptologisches Wissen vom Programmierer in der Methoden fordert.

2.2.2 JCA - Java Cryptography Architecture

Diese in den JDK enthaltene Bibliothek teilt sich in verschiedene Bereiche (u.A Verschlüsselung, Schlüsselerzeugung und Zertifikatsvalidierung). Die konkrete Implementierung der

[1] Bundesamt für Sicherheit in der Informationstechnik

Verschlüsselung ist vom ausgewählten Provider definiert.

2.2.3 JCE - Java Cryptograhy Extension

Die stärksten Verschlüsselungstechnologien waren für Unternehmen außerhalb der USA bis 2017 nicht verfügbar und somit separat von der JCA im JCE gebündelt. Inzwischen ist diese Standard-Erweiterung Teil der JCA.

2.2.4 Botan 2.X

Diese in C++ implementierte Bibliothek ist auf möglichst einfache Bedienung fokussiert und entstand in Zusammenarbeit mit dem BSI. Botan 2.X unterstützt unter anderem folgende Funktionen:

- Transport Layer Security (TLS)
- Public Key Infrastructure (PKI)
- Public Key Cryptography (PKC)
- Verschlüsselungen
- Hash-Funktionen
- Message Authentication Codes (MAC)
- Checksummen

2.2.5 Google Tink

Diese von Google für den interne Gebrauch entwickelte Bibliothek ist in den Sprachen Java, C++, Objective-C und Go implementiert. Der Fokus liegt hier möglichst einfacher Bedienung. Seit August 2018 ist sie der Öffentlichkeit auf Github zugänglich.

2.3 Fazit

Eine Bibliothek sollte nach folgenden Kriterien ausgewählt werden:

- kontinuierliche Updates
- bestenfalls Support
- gut dokumentiert
- seriöse Quelle der Implementierung
- keine Eigenkreationen
- verwendet allgemeine Standards

3 FIDO2

FIDO2 ist ein Standard der FIDO-Allianz und dem W3C. FIDO steht führ Fast Identity Online. FIDO2 basiert auf WebAuthn vom W3C und CTAP2 von der FIDO-Allianz.

3.1 Motivation

Passwörter sind ein Hauptgrund für 80 Prozent aller Datenlecks. Das Ziel von FIDO2 ist es, die manuelle Passworteingabe zu ersetzten und somit die Schwachstelle der Passwörter zu umgehen, da sich die Betreiber nicht mehr um die Speicherung und Absicherung von Passwörtern kümmern müssen. Dabei soll das Einloggen sicherer gemacht werden, ohne den Komfort einbüßen zu müssen.

3.2 Begriffe

FIDO-Allianz Die FIDO-Allianz wurde im Februar 2013 gegründet, um das globale Passwortproblem zu lösen und eine sichere und komfortable Alternative zu entwickeln. Die Gründungsmitglieder sind Agnitio, Lenovo, NokNokLabs, PayPal und ValiditySensors. Seit der Gründung der Allianz haben sich über 150 Firmen angeschlossen. Dazu zählen Firmen wie Bank of America, Microsoft, Mastercard und Google. FIDO hat Standards wie UAF und U2F hervorgebracht.

W3C Das W3C ist ein Gremium zur Standardisierung von Techniken im Internet und wurde von Tim Berners Lee am 01.10.1994 am MIT gegründet. Es hat die standardisierten Technologien HTML, XHTML und CSS hervorgebracht.

UAF UAF ist ein Standard der FIDO-Allianz für eine Authentifizierung ohne Angaben von Benutzername und Passwort. In UAF werden biometrische Merkmale wie der Fingerabdruck, das Gesicht, die Regenbogenhaut, oder die Stimme verwendet.

U2F U2F ist ein Standard der FIDO-Allianz für ein Zwei-Faktor-Authentisierung, basierend auf dem asymmetrischen Challenge-Response Verfahren.

WebAuthn WebAuthn ist ein Standard für die passwortfreie Authentifizierung von Nutzern in Webanwendungen. Dabei werden kryptographische Operationen von einem Authentifikator ausgeführt.

CTAP2 CTAP steht für Client-To-Authenticator-Protocol. Es ermöglicht die Verwendung externer Authentifikatoren über USB, NFC oder Bluetooth.

3.3 Unterstützung

Welche Schnittstellen benötigt ein FIDO2 Authentificator

- Lightning
- USB-A, USB-C
- Bluetooth, NFC

Welche Anwendungen unterstützen FIDO2

- Microsoft.com (outlook.com, Office 365, OneDrive)
- Allerdings mit Eingabe eines Pins

Als 2.Faktor bei:

- Github, GitLab, Code Enigma
- Amazon Facebook, Twitter
- DropBox, BoxCrypto
- Moderne Webbrowser

3.4 Fazit

Vorteile

- Kleinere Angriffsfläche
- Ein Sicherheitstoken, keine unterschiedlichen Passwörter
- Einfaches Konzept
- Sicherheit kann beliebig hoch skaliert werden
- Kein physischer Token notwendig
- Keine Kopien

Nachteile

- wenige Anwendungen, die FIDO unterstützen
- Kosten für Sicherheitstokens
- Verschlüsselung nicht nach Best Practice, aus Kompatibilitätsgründen

4 Safety

Mit Safety wird die Betriebssicherheit eines Systems beschrieben. Ein System kann so vor ungewolltem Schaden geschützt werden. Weiterhin sind Zuverlässigkeit, Ablauf- und Ausfallsicherheit wichtig.

4.1 Software Assurance

Um sicher zu stellen das eine Anwendung sicher und verlässlich ist können während der Entwicklung einige Schritte verfolgt werden.

- **Konzeptphase:** Wie soll der Nutzer mit dem System interagieren ?
- **Anforderungsphase:** Wie verlässlich oder verfügbar soll das System sein ?
- **Programmierphase:** keine Schwachstellen implementiert
- **Evaluations und Testphase:** ist das Programm konsistent ?
- **Releasephase:** Support und Wartung

4.1.1 Fuzzing

Bei Fuzzing wird ein Programm mit ungültigen oder zufälligen Eingaben getestet. Somit kann automatisch geprüft werden wie Robust eine gegebene Software ist. Die Gesamtsicherheit kann allerdings nicht gewährleistet werden.

Beim Fuzzing durchläuft ein Programm folgende Schritte:

- Testdaten erstellen und Struktur verändern
- Übergabe an Programm
- Programm überwachen und finden der Absturzursache.

4.2 Robustheit und Abhängigkeit

4.2.1 Robustheit

Robustheit häufig auch als Fehlertoleranz bezeichnet, beschreibt die Eigenschaft eines Verfahrens/Systems auch unter ungünstigen Bedingungen noch zuverlässig zu funktionieren. Die komplette Robustheit eines Systems ist nicht immer möglich, in einem solchen Fall schaffen aussagekräftige Fehlermeldungen und kontrolliertes beenden des Systems Abhilfe. Wichtig ist es sicherzustellen das keine Daten oder Hardware durch das System in diesem Zustand beschädigt werden.

4.2.2 Abhängigkeit

Ist das System von Bibliotheken abhängig muss sichergestellt werden das durch Veränderung/entfernen dieser Bibliotheken keine undefinierten Zustände auftreten. In einem solchen Fall sollten Fehlermeldungen mit Lösungsvorschlägen ausgelöst werden.

4.3 Fault Isolation

Um das Betriebssystem vor schlecht programmierten Treibern zu schützen werden diese nach bestimmten Verfahren isoliert (Sandbox, Virtuelle Maschine, Hardware, Programmiersprachen)

4.4 Schutz von Serveranlagen

Um den Verlust wichtiger Daten und Ansehen zu vermeiden sollten Serverräume vor Brand, Wasserschaden, Stromausfall und unbefugtem Zugriff geschützt werden.

4.4.1 Backups

Zusätzlich sollten für den Fall eines Ausfalls Backups nach dem Tao of Backup[2] durchgeführt werden.

4.5 Fazit

Safety schützt den Nutzer und das System vor Unfällen und einigen unangenehmen Überraschungen.

[2] `http://taobackup.com/`

5 Random Numbers

5.1 Was sind Zufallszahlen ?

Echte Zufallszahlen werden durch physikalische Prozesse erzeugt. Bei echten Zufallsbitfolgen sind die Werte 0 und 1 gleich wahrscheinlich. Es ist unmöglich zu bestimmen, welchen Wert das nächstfolgenden Bit hat.

5.2 Physikalische Prozesse

Beispiele: Münzwurf, Würfeln, Rauschen von elektronischen Bauelementen, radioaktiven Zerfallsprozessen

5.3 ERNIE

ERNIE steht für Electroning Random Number Indicator Equipment und ist ein Zufallsgenerator, der entwickelt wurde, um monatliche Lotterieziehungen durchzuführen. ERNIE1 wurde 1957 vorgestellet und bestimmte die Zufallszahlen, anhand des Rauschen mehrere Neonröhren. Später wurde ERNIE1 durch ERNIE2 und danach durch ERNIE3 ersetzt. Diese neueren Versionen brachten Optimierungen, in Bezug zur Laufzeit und Größe mit sich. ERNIE4, welcher August 2004 in Betrieb genommen wurde, generiert Zufallszahlen, mittels Wärmerauschen in Transistoren. Dadurch ist ERNIE4 500-mal schneller als das Original. Das aktuelle Modell ERNIE 5 wurde im März 2019 auf den Markt gebracht und bedient sich als erstes Modell nicht mehr an Rauschen, sondern arbeitet auf Quantenebene und generiert Zufallszahlen in Abhängigkeit von Licht.

5.4 /dev/random

/dev/random ist unter Unix-Betriebssystemen eine Funktion, die zur Generierung von Zufallszahlen Werte aus den Speicherbereichen von Gerätetreibern benutzt.

5.5 Pseudozufallszahlen

Pseudozufallszahlen werden durch einen deterministischen Algorithmus berechnet, sie sind daher eigentlich keien echten Zufallszahlen.

5.5.1 /dev/urandom

/dev/urandom funktioniert wie /dev/random, ohne den Lesepool zu Erschöpfen.

5.5.2 Middle-Square Methode

Die Middle-Square Methode generiert Zufallszahle mithilfe eines x-stelligen Seeds, der quadriert wird. Die mittleren Stellen des Ergebnisses sind die Pseudozufallszahl und der Seed für die folgende Pseudozufallszahl. Das Ergbnis muss eine gerade Anzahl an Stellen haben. Ist das nicht der Fall wird die Zahl mit Nullen aufgefüllt. Die Middle-Square Methode gilt als rechenintensiv und schwach, da sich die Sequenzen bei bestimmten Zahlen wiederholen.

5.5.3 Algorithmus K

Der Algorithmus K benutzt auch einen Seed. Der Algorithmus wird in Abhängigkeit des Seed unterschiedlich Häufig und in verschieden Reihenfolgen abgearbeitet. Der Algorithmus wurde mit der Intention entwickelt, dass er sich bei Betrachtung des Maschinencodes nicht ohne Kommentare rekonstruieren lässt. Allerding kommet es auch hier zu sich wiederholenden Sequenzen.

5.5.4 Nutzereingaben

Erzeugt Zufallszahlen durch vom Nutzer nicht kontrollierbare Eingaben. Beispiel: Tastaturanschlägen, pixelgenaue Bewegung der Maus, Lage des Gyroskops bei Smartphones

5.6 Implementationen

5.6.1 C/C++

- stdlib/rand
- sodium/randombytes

5.6.2 Java

- util/Random
- util/SecureRandom
- Math.random

5.6.3 JavaScript

- crypto.getRandomValues

5.6.4 PHP

- random_bytes

5.6.5 Python

- random
- secrets

5.7 Anwendungsgebiete

- Simulationen
- Stichproben
- Fehlersuche in Computerprogrammen
- Entscheidungsfindung
- Unterhaltung
- Kryptographie

5.8 Verschlüsselungsverfahren

5.8.1 One Time Pad

- Schlüssellänge mindestens so lange wie Nachricht
- Nachteile: Schlüssel muss vor erster Verwendung ausgetauscht werden.

Vorgehen

- Alphabet: 0,1
- Klartext: x, Chiffriertext: y
- Klartext-Bit b wird mit hilfe eines Zufallsbits z zu Chiffriertextbit c transformiert
- $b \rightarrow c|b \oplus z$

Da XOR selbstinvers ist, kann durch dieselbe Methode das Chiffriertext-Bit c wieder entschlüsselt werden.

5.8.2 Stromverschlüsselung

- Daten werden am Stück verschlüsselt
- Verfahren arbeitet rechtzeitig
- Schlüsselstrom rekonstruieren
- Kein Geschwindigkeitsvorteil mehr

5.8.3 Blockverschlüsselung

- Nachricht in Blöcke fester Länge
- jeder Block wird separat Ver -und Entschlüsselt
- deterministisches Verschlüsselungsverfahren

5.8.4 SSL / TLS Nonce

- Wird bei HTTPS verwendet
- Wichtig für Handshake Phase
- Server/Client geben jeweils einen Nonce vor
- verhindert Replay-Angriffe auf den Server

6 Threat Modeling Tool

6.1 MS Security Development Lifecycle (SDL)

Grundsätze:

- Secure by Design
- Secure by Deployment
- Communications
- Secure by Default

Gründe für SDL:

- unzureichende Software-Sicherheit
- mangelnde Sicherheit, die von den Benutzern implementiert wurde
- Sozial-Engineering
- Vulnerabilität in der Software
- Hacker

Schritte des SDL:

- Provide Training
- Define Security Requirements
- Design
- Implementation
- Verification
- Release/Response

6.2 Threat Modelling

Thread Modelling ist ein Prozess, bei dem Risiken, Gefährdungen und Schwachstellen identifiziert und gemindert werden sollen.

5 Schritte des Thread Modelling:

- Definition
- Diagramm
- Identifikation
- Milderung
- Überprüfung

6.3 Microsoft Threat Modeling Tool

Es können Modelle erstellt werden. Ein Modell kann Diagramme enthalten. Es gibt z.b. das STRIDE-Modell, bei dem die Anfangsbuchstaben für folgende Bedrohungen stehen:

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of Service
- Elevation of privelege

In diesem Modell werden Bedrohungen nach Kategorien klassifiziert. Danach kann eine Sicherheitsstrategie der Bedrohung erstellt werden.

Anhand des erstellten Diagramms kann ein Report erstellt werden. Der Report enthält identifizierte Bedrohungen und bietet Gegenmaßnahmen zur Vermeidung an.

6.4 Fazit

Thread Modelling ist für die Sicherheit wichtig. Das MS Thread Modeling Tool bietet eine Bedrohungsanalyse. Potenzielle Gefährdungen können erkannt und im Entwicklungsprozess vermieden werden. Es wird kaum Sicherheitswissen vorausgesetzt.

7 RESTful API Authentication (Token based)

7.1 RESTful APIs

Eine API ist ein Application programming interface. REST API steht für Representational State Transfer Application Programming Interface. Eine REST API ist ein zustandsloses Interface bei den Nachrichten über ein Netzwerk ausgetauscht werden. Es wird das HTTP Protokoll für die Kommunikation verwendet.

Vorteile:

- schnell
- Skalierbar
- flexibel
- unabhängig
- leicht integrierbar

Sicherheitsbedrohungen:

- Client hat volle Kontrolle auf die Ressourcen
- Injection Attacks
- MITM
- DoS

7.2 Authentifizierungsverfahren

Es gibt verschiedene Authentifizierungsverfahren:

- Nachweis durch Wissen (Passwort, PIN)
- Nachweis durch Besitz (Tokens)
- Nachweis durch Biometrie (Fingerabdruck, Augen)

Bei der Session based Authentification speichert der Server den Status der Session. Die Session wird über eine Session ID, die in den Cookies gespeichert wird, erkannt.

Bei der Token based Authentifikation werden die Daten der Session Clientseitig gespeichert. Der Zustand der Session wird verschlüsselt und in den Cookies gespeichert.

Bei der Zwei-Faktor-Authentifikation wird eine zweite Authentifizierung nach der eigentlichen Authentifizierung durchgeführt. Zum Beispiel wird nach Eingabe eines Passworts noch ein Hardwaretoken benötigt.

7.3 JSON Web Tokens

Ein JSON Web Token enthält Informationen über die Session. Der Token enthält personenbezogene Daten, wie z. B. Benutzername oder E-Mail-Adresse. Der Token wird anschließend vom Server signiert. Durch Die Signierung ist es nicht möglich die Informationen in einem Web Token zu manipulieren. Der Server würde eine Manipulation erkennen, da die Signatur dann ungültig wäre.

7.3.1 Struktur

Ein JSON Web Token enthält:

- Header
- Payload
- Signature

Der Header enthält information über die verwendeten Verschlüsselungsalgorithmen die verwendet werden. Der Payload ist im JSON-Format. Der Payload und Header wird dannach base64 kodiert. Der Server erstellt mit sein Private Key eine Signatur der Daten im Token. Die Signatur wird am Payload angehängt.

7.4 Autorisierungsprozess

Wenn man sich beispielsweise über Facebook bei Spotify anmelden möchte, erhält Spotify einen Zugriffstoken von Facebook. Mit dem Token können Daten des Benutzers anrufen. Beispielsweise kann der Namen, die E-Mail-Adresse und das Geburtsdatum des Benutzers abrufen werden.

OAuth 2.0 ist ein Framework zur Verwaltung und Erstellung dieser Token.

Flows von OAuth 2.0:

- Authorization Code
- Implicit
- Resource Owner Password Credentials
- Client Credentials

8 Key Sharing

8.1 Was ist Key Sharing ?

Key Sharing oder auch Secret Sharing ist eine Methode bei der ein Secret auf eine Gruppe von Teilnehmern verteilt wird. Jeder Teilnehmer erhält ein Share und es wird eine vorgegebene Anzahl von Shares benötigt, um das Secret wiederherzustellen.

8.2 Anwendungsbereich

Ziele:

- Konzentration vermeiden
- Risiken verbreiten
- Eingriffe tolerieren

Anwendung gibt es in der Informationssicherheit. Es kann Vertraulichkeit der Daten sichergestellt werden.

8.3 Beispiel

- Secret: x
- Shares: x_1 bis x_n
- x_1 bis x_{n-1} zufällig wählen
- x_n erzeugen: $x_n = x - \sum_{i=1}^{n-1} x_i$
- Secret erzeugen: $x = \sum_{i=1}^{n} x_i$

8.4 Probleme bei diesem Fall

Es müssen alle Shares von allen Teilnehmern gesammelt werden, um das Secret wiederherzustellen. Bei Verlust eines Share sind die Daten verloren, wenn diese mit dem Secret verschlüsselt wurden.

8.4.1 Threshold Secret Sharing

Bei dem Threshold Secret Sharing kann das Secret mit einer Mindestanzahl von Shares erzeugt werden. Dabei ist die Anzahl der benötigten Shares der Schwellenwert oder Threshold. z. B. Mindestens die Hälfte der Shares werden nötig um das Secret herzustellen.

8.5 Blakley's secret sharing

Idee: Zwei nicht parallele Linien in derselben Ebene schneiden sich an genau einem Punkt und drei nicht parallele Ebenen kreuzen sich an genau einem Punkt.

Jeder Share ist eine Ebene. Das Secret ist der Schnittpunkt $[x, y, z]$ Es werden 3 Ebenen benötigt, um einen Schnittpunkt zu berechnen. Wenn es mehr als drei Ebenen gibt, kann der Schnittpunkt aus drei beliebigen Ebenen daraus errechnet werden.

8.6 Shamir's secret sharing

Secret Sharing von Shamir basiert auf der Grundidee des Polynoms.

Im Beispiel:

- Secret x = 3
- threshold r = 3
- polynom festlegen mit grad = r - 1 = 2
- allg. Form des Polynomes: $y = ax^2 + bx + c$
- a und b zufällig wählen
- c ist das Secret
- unser Polynom ist also: $y = 2x^2 + x + 3$
- jedes Share ist ein punkt auf dem Polynom
- das Polynom kann mit 3 punkten genau ermittelt werden.
- der c Koeffizient ist nach der Rekonstruktion das Secret.

9 Linux Hardening

Die Sicherheit in einem System ist nur gegeben wenn:

- die Hardware sicher ist
- der Nutzer geschult ist
- die Software sicher ist

9.1 Hardware

Wichtig ist bei Hardware vor allem die Auswahl und Konfiguration der Komponenten (vertrauenswürdig, sicher, stabil). Zusätzlich sollten die Komponenten regelmäßig auf volle Funktionalität überprüft werden.

Um die Hardware vor Umwelteinflüssen und Zugriff durch nicht privilegierte Personen zu schützen ist auch die Wahl des Standorts wichtig.

9.2 Nutzer

Damit der Nutzer korrekt mit dem System umgeht sollte er in der Bedienung geschult werden.

Umgang mit E-Mails Der Nutzer sollte Phishing und Spoofing Angriffe erkennen können.

Passwörter Einhaltung der NIST[3] Passwortrichtlinien.

Viren und andere Schadsoftware Ausschließlich Verwendung von vertrauenswürdiger Software und entsprechender Bezugsquellen.

Surfen im Internet Dem Nutzer sollte es möglich sein gefälschte Webseiten und Links zu erkennen.

Sicheres Austauschen von Dateien Wie authentifiziere und verschlüssle ich Dateien und Nachrichten.

[3] `https://pages.nist.gov/800-63-3/sp800-63-3.html`

9.3 Software

Das Betriebssystem sollte dem Einsatzzweck entsprechend ausgewählt werden.[4]

9.3.1 allgemein

Zusätzlich sollten bei der Einrichtung best practices eingehalten werden, wie etwa:

- root Login sperren
- starke Policy für Passwörter
- Rechte einschränken
- regelmäßiges Einspielen von Updates
- überprüfen auf vorhandene Sicherheitslücken in Datenbanken

9.4 Netzwerk

Da das Netzwerk die größte Angriffsfläche ohne physikalischen Zugriff ist sollte hier besonders geschützt werden, etwa mit:

- ändern von Standardports(etwa für SSH)
- Verschlüsselung
- Firewalls und Filter
- Intrusion Detection Systemen
- Intrusion Prevention Systemen
- Backups

9.5 Fazit

Sicherheit ist Zusammenspiel aus User Sensibilisierung, Hardware und Software. Hardware und Software sollte auf den Use Case angepasst werden.

[4] `https://distrochooser.de`

10 Codesigning

10.1 Signatur

Signaturen entstehen dadurch, dass Daten gehasht und anschließend der Hash-Wert mit einem asymmetrischen Kryptographie-Verfahren verschlüsselt wird. Der dabei zum Einsatz kommende private Schlüssel kann einer bestimmten Person zugeordnet sein, sodass die Signatur nicht nur Datenänderungen erkennen lässt, sondern auch noch Auskunft liefert, wer den Datensatz „unterschrieben“ hat.

10.1.1 Allgemeiner Ablauf

Alice hasht ihre Daten und verschlüsselt sie mit ihrem Private Key. Anschließend sendet Alice die originalen Daten, die Hashfunktion, den Public Key und das Verfahren an Bob. Bob hasht die originalen Daten und entschlüsselt mit dem Public Key die verschlüsselten Daten. Wenn die gehashten Daten und die entschlüsselten Daten übereinstimmen ist gewährleistet, dass die Daten nicht verändert wurden.

10.2 Unterschiede bei Betriebssystemen

iOS und XOS Signaturen werden vom Benutzer selbst erstellt. Apple stellt eigene Zertifikate für dessen Applikationen aus. Zertifikaten werden auf iOS und XOS üblicherweise über XCode verwaltet.

Probleme Mehrere Sicherheitszertifikate sind abgelaufen. Dadurch können einige Apps nicht mehr installiert oder geupdatet werden. Außerdem veröffentlicht Apple ältere Softwarepakete neu. Daher werden Apps auf Applegeräten oft gecrackt, weshalb sich Apple das Recht vorbehält, eine App jederzeit vom Markt zu nehmen.

Windows Signaturen werden mit dem Befehlszeilentool SignTool erstellt. Zertifikate werden meist von Drittanbietern ausgestellt.

Android Apps müssen signiert sein um sie Installieren zu können. Unsignierte Apps werden von Google Play und dem Paketmanager blockiert. Google Play sucht mit Hilfe von Signaturprüfungen und Verhaltenserkennung regelmäßig nach Schadsoftware. Die Signaturen der Apps sind ein Bestandteil der Application-Sandbox

Application Sandbox Die Application Sandbox ist ein System, das die Zugriffe auf Apps reguliert. Dies ist eine Grundlage für die Datensicherheit und den Datenschutz auf Android-Systemen. Der Zugriffschutz wird mit jedem Versionsupdate gehärtet. Die Sandbox verleiht jeden Apps User-IDs, welche aus dem Public Key der App-Signatur definiert wird. Die Apps haben keinen Zugriff auf Daten, denen Andere User-IDs zugeordnet sind.