# Aufgaben

# Widerholung

## Erläutert die in der Schulung vorgestellten Secure Design-Prinzipien in eigenen Worten.

### Least Privilege Die Anwendungen sollten so modularisiert sein, dass funktionale Verantwortlichkeiten gebündelt sind, und somit nur innerhalb eines z.B. micoservices benötigt und auch nur da angefordert werden.

### Defense in Depth Eine modularisierte Anwendung (nicht – monolitisch!) sollte Aufrufern nicht implizit vertrauen, d.h. ein webfrontend muss sich gegenüber dem Backend Autrhentifizieren, und dieses z.B. gegenüber einer Datenbank oder weiteren aufgerufenen Services, damit man nicht durch eine Schwachstelle direkt Zugriff auf „alles“ hat.

### Fail Security oder Expect the Unexpected: Im Fehlerfall sollte sich das System danach noch ein einem definierten Zugstand befinden und auch keine Informationen über sich preisgeben.

### No Security by Obscurity Man sollte sich nie darauf verlassen, dass „niemand“ schon z.B. den Crypto – Algorithmus herausbekommen wird, er muss vom Design her so sicher sein, dass man nur die Schlüssel sicher aufbewahren muss.

### Detect and Record „Auffällige“ Aktivitäten sollten in speziell gesicherten Audit Logs landen, damit ein potentieller Hacker diese nicht manipulieren kann und man im Fall des Falles dadurch immer noch Spuren und Hinweise über die Methode finden kann, selbst wenn man darüber den Angriff selbst nicht sofort mitbekommt.

### Don’t Trust Man darf an keiner Schnittstelle einer Software oder eines Systems darauf vertrauen, dass sie nur so benutzt wird, wie man es gewünscht hat – die User sind meist sehr viel kreativer als man denkt. Insofern sowohl security-technisch alsauch inhaltlich sind alle Werte zu überprüfen, die auf uns prallen.

### Keep it simple Generell sollte man alles, was man nicht benötigt, auch nicht aktivieren, oder sogar löschen. Jede nicht benötigte Schnittstelle oder Funktionalität kann ein potentielles Security Problem darstellen, wenn sie nicht gewartet wird.

## Welchen Zweck erfüllen Secure Design-Prinzipien im Allgemeinen? Sie sollen die Anwender dieser Prinzipien für grundlegende Bereiche, in denen Sicherheitsprobleme entstehen können, sensibilisieren.

## Diskutiert die folgende Aussage: Eine konsequente Umsetzung des Prinzips Security by Obscurity reicht aus, um die Sicherheit einer Anwendung zu gewährleisten. Kompletter Humbug, und schon sehr oft schief gegangen!

## Beschreibt das Konzept der Distrustful Decomposition. Welche Design-Prinzipien werden umgesetzt? Wie können Prozesse untereinander kommunizieren? Don´t Trust, Defense in Depth, Least Privilege. Am besten mit standartisierten Protokollen, die etablierte Security-Mechanismen schon mitbringen, siehe https uvm.

# Secure Design & Defensive Coding

## Secure Design Prinzipien in der Theorie

1. Ein Angreifer kann dem Frontend Management, ohne zuvor erfolgte Authentifizierung, Dateien schicken. Wenn es sich hierbei um DMN-Dateien handelt, überschreibt das Frontend Management bereits vorhandene DMN-Dateien mit demselben Namen.  
   Welches Secure Design-Prinzip wird in diesem Szenario verletzt? Sollten mehrere Prinzipien verletzt werden, so nennt bitte nur eines. Begründet eure Antwort bitte in maximal 2 Sätzen.  
     
   DON’T TRUST: Das Frontend Management darf nicht darauf vertrauen, dass der User DMN-Dateien ablegen darf, sondern muss prüfen, ob der User auch die Berechtigung dazu hat. Dabei würde auffallen, dass der User nicht angemeldet ist.  
   LEAST PRIVILEGE: Der Frontend Teil der Dateien von unauthentifizierten Usern entgegen nimmt, sollte nicht die Möglichkeit haben, DMN’s zu speichern/zu überschreiben, sondern eben nur „neue“ Schaden-Dateien zu einem angelegten Schaden annehmen.
2. Der AuthFilter erhält einen ungültigen Token und antwortet auf die Anfrage mit einer detaillierten Fehlermeldung. Das Frontend Management leitet diese Fehlermeldung unverändert an den Ursprung der Authentifizierungsanfrage weiter.  
   Welches Secure Design-Prinzip wird in diesem Szenario verletzt? Sollten mehrere Prinzipien verletzt werden, so nennt bitte nur eines. Begründet eure Antwort bitte in maximal 2 Sätzen.  
     
   FAIL SECURELY: Das Frontend sollte nur die Tatsache „Access Denied“ oder ähnlich weiterleiten (und immer die gleiche bei allen Prüfungen, sei es nun defektes Token oder zu wenig Rechte), damit der Angreifer nicht aufgrund der Fehlermeldung auf Frameworks/Verfahren und/oder Fehlersituationen reagieren kann.
3. Ein Angreifer verändert die Anfrage Request DMN Diagram List, sodass diese zusätzliche Datenbankanfragen enthält. Das Backend Management prüft die Anfrage nicht vollständig, sodass alle Anfragen durchgeführt und ihre Ergebnisse in der Antwort enthalten sind.  
   Welches Secure Design-Prinzip wird in diesem Szenario verletzt? Sollten mehrere Prinzipien verletzt werden, so nennt bitte nur eines. Begründet eure Antwort bitte in maximal 2 Sätzen.  
     
   Don’t Trust: Der User Input sollte per Prepared Statement entsprechend „escaped“ bzw. Separate zur DB geschickt werden, so dass das verknüpfen nicht möglich ist, oder es sollte mit entsprechenden Encodern (z.B. von der OWASP) der Input so aufbereitet werden, dass ein Ausbruch aus einem SQL-Parameter unmöglich ist. Man darf User Input nicht ungeprüft oder unencodiert an andere Sinks schicken.
4. Das Frontend Management führt keine Logs über Anmeldeversuche und Anmeldungen am InsecureDMNManager. Einem Angreifer ist es so möglich, sich mithilfe eines Brute Force-Angriffes einzuloggen und im Namen des gehackten Benutzers Änderungen an den DMN-Diagrammen durchzuführen.  
   Welches Secure Design-Prinzip wird in diesem Szenario verletzt? Sollten mehrere Prinzipien verletzt werden, so nennt bitte nur eines. Begründet eure Antwort bitte in maximal 2 Sätzen.  
     
   Detect and Record: geglückte und misslungene Anmeldeversuche müssen in ein speziell gesichertes (am besten auch zentralisiertes) Audit Log geschrieben werden, welches von einem Hacker auch nicht modifiziert werden kann, selbst wenn er remote code execution kann.

## Defensive Coding

1. Welche Defensive Coding-Richtlinie wird in diesem Beispiel verletzt? Sollten mehrere Richtlinien verletzt werden, so nennt bitte nur eine.
2. Erläutert die Schwachstelle in maximal 4 Sätzen anhand der Stelle(n) an der die Verletzung auftritt.
3. Gebt bitte ein Programm bzw. einen Aufruf an, der den Exploit durchführt. Alternativ könnt ihr in maximal 3 Sätzen beschreiben wie ein Exploit aussieht, der die entstandene Vulnerability ausnutzt.
4. Behebt bitte den Fehler anschließend und gebt den Code im PDF und als extra Datei ab.

CustomerManagement:

1. Input Validation: Es wird dem User Input „customerName“ vertraut.
2. Der Input „customerName“ aus dem Parameter der Funktion getCustomerDetail wird ohne escaping direkt 6 Zeilen tiefer in das SQL Statement eingefügt.
3. Wenn es dem Angreifer gelingt, den Name z.B. „x‘ OR 1=1“ zu setzen, würde aufgrund des fehlenden Escaping eine Liste aller Customers ausgegeben werden.
4. // Statement stmt = conn.createStatement();

// ResultSet result = stmt.executeQuery("SELECT \* FROM Customers WHERE Name = '" + customerName + "'");

PreparedStatement prepareStatement = conn.prepareStatement("SELECT \* FROM Customers WHERE Name = ?;");

prepareStatement.setString(0, customerName);

ResultSet result = prepareStatement.executeQuery();  
Siehe Code

FileReader

1. Exception Handling: Eine FNFException wird einfach durchgereicht.
2. Die Methode getFile sollte bereits prüfen, ob die Datei eine „legale“ Datei ist die gelesen werden darf (über getCanonicalPath prüfen!), und ob sie existiert.
3. Im Moment kann man mit Path Traversal (z.B. ../../../etc/passwd) durch das system laufen, um beliebige Dateien auszulesen..
4. read-Methode: Wenn file == null, dann nur leere Liste zurückgeben.  
   getFile: Wenn name == null, dann null zurückgeben.  
   getFile: Wenn name nicht in file.getcanonicalPath() dann null zurückgeben.  
   getFile: Mögliche IOException abfangen und per Logger Hinweis Loggen (Audit Logging?)  
   Siehe Code.

Sieve:

1. Attack Surface: unnötig viele Methoden public, und liefern das original Array zurück.
2. Methoden, die nicht public sein müssen sollten privat sein, außerdem sollte man nie seine original Werte zurückliefern. Eine Kopie reicht auch.  
   Kopie zurückgeben:  
   **public** **boolean**[] getPrimes() {  
    **return** Arrays.*copyOf*(primes, primes.length);  
    }  
    Ich habe das Teil großflächig umgebaut, um mehrere Probleme zu verhindern (ist sieve() schon gelaufen? uvm).
3. Wenn man eine direkte Referenz auf primes hat, kann man das Array ändern, ohne eine der Methoden aufzurufen.
4. Siehe Code.

Pad:

1. Concurrency
2. Die verwendete ArrayList ist nicht für konkurrierende Inserts optimiert. Man könnte das Problem beheben indem man sie über die Collections-Methode syncronisiert, hier ist aber eine andere Lösung gefordert: Wir nutzen einen static Semaphore für einen Syncronized Block.
3. Programm ist ja gegeben, beide Autoren schreiben wird durcheinander.
4. FIX: **private** **static** **final** Object ***SEMAPHORE*** = **new** Integer((**int**) Math.*round*(Integer.***MAX\_VALUE***\*Math.*random*()));  
     
   …  
   in der   
   public void run() {  
    synchronized (SEMAPHORE) {  
   …  
    }  
   }

# Aufgabe 3: Verschlüsselungsalgorithmen

One-Time Pad (OTP) ist ein Verschlüsselungsverfahren, das 1882 durch den amerikanischen Kryptographen

Frank Miller erstmal beschrieben wurde. Einen kurzen Überblick der Funktionsweise von OTP findet ihr

hier.

a) Beschreibt die Funktionsweise von OTP in eigenen Worten. Verwendet für eure Beschreibung nicht

mehr als 4 Sätze.

b) Handelt es sich bei OTP um ein symmetrisches oder um ein asymmetrisches Verschlüsselungsverfahren?  
c) Welche Probleme bestehen bezüglich des Schlüssels und des Schlüsseltauschs?

Begründet eure Antwort!

1. Bei einem OTP wird vor Nutzung ein zufälliger Schlüssel beiden Parteien bekannt gemacht, welcher sicher aufzubewahren ist. Die Nachricht wird durch Addition (modulo 26) der Buchstaben verschlüsselt, Entschlüsselt wird mit dem gleichen Schlüssel.  
   Besonderheit ist, dass wenn der Schlüssel wirklich zufällig ist, es das sicherste mögliche Verfahren ist was es gibt, solange die Sicherheit des Schlüssels gewährleistet ist, und dass der Schlüssel mindestens so lang wie die Nachricht sein muss.  
   Außerdem darf ein Schlüssel immer nur genau einmal benutzt werden.  
   Insbesondere das Problem der absoluten Zufälligkeit ist schon mehreren Agenten zum Verhängnis geworden.
2. Es ist ein symmetrisches Verfahren, da der gleiche Schlüssel sowohl für Verschlüsslung als auch für Entschlüsselung genutzt wird.
3. Der Schlüssel muss mindestens so lang sein wie die Nachricht, da ansonsten Wiederholungen auftreten.  
   Der Schlüssel muss absolut zufällig sein, da ansonsten ein „Rauschen“ statistisch ermittelbar ist, und deswegen das Verfahren mit statistischen Verfahren angreifbar ist. (Siehe Ziehen mit Bingomaschine aber nachschauen, oder statistisch ermittelbare Zeichenfolgen beim „auf die Tastatur hämmern“).  
   Der Schlüssel muss sicher übertragen werden!  
   Der Schlüssel darf nicht wiederverwendet werden!