IT-Security Tutorübung 01

Dorian Zedler

23. Oktober 2023

Technische Universität München

Inhalt der Tutorübung

Organisatorisches

Kommunikation

1. Folien: `https://itsec.dorian.im`
2. Inhaltliche Fragen
   - E-Mail: `dorian.zedler@tum.de`
   - Zulip: Dorian Zedler [ITSec Tutor]
3. Organisatorische und Fragen von allgemeinem Interesse
   - Zulip: `https://zulip.in.tum.de`
4. Wichtige Sonderfälle
   - Übungsleitung: `franzen@sec.in.tum.de`

Notenbonus

- Wöchentliche Challenges über `https://scoreboard.sec.in.tum.de`
- In Teams mit 2 Studierenden
- Vorstellung einer Hausaufgabe **pro Person**
  - Anmeldung bis Sonntag 18 Uhr auf dem Scoreboard
  - Bei zu vielen Anmeldungen wird gelost
- Notenbonus 0.3 oder 0.7 je nach Anzahl der Punkte

System Setup

System Setup: Scoreboard und SSH

1. SSH-Keypair generieren
   - `ssh-keygen -t ed25519`
2. Public Key kopieren
   - `cat .ssh/id_ed25519.pub`
3. Public Key auf `https://scoreboard.sec.in.tum.de` hochladen
4. Mit dem im Scoreboard angezeigten Befehl auf dem Sandkasten Rechner einloggen
   - `ssh unk-<your_nr>@sandkasten.sec.in.tum.de`
   - Wer schon ein Team hat:
     `ssh team-<team_nr>@sandkasten.sec.in.tum.de`
5. Team auf Scoreboard registrieren!

System Setup: Python

1. Python installieren
2. `requests` Bibliothek installieren:
   `pip install requests`
3. Python Baukasten am Ende des Übungsblattes
4. Probleme? Fragen? $\rightarrow$ Nach dem Tutorium

Aufgaben

Aufgabe 2 - Begrifflichkeiten

a) Grenzen Sie kurz die Begriffe *Safety* und *Security* voneinander ab. Worin unterscheiden sie sich?

- **Safety**
  - Schützen des Systems **vor sich selbst**
  - Störungen **von Innen**
  - z.B. Hardware-Fehler oder Programmierfehler
- **Security**
  - Schützen des Systems **vor anderen**
  - Störungen/Angriffe **von Außen**
  - z.B. Daten-Manipulation, Informationsmissbrauch oder Funktionsstörung

Aufgabe 2 - Begrifflichkeiten

b) Beschreiben Sie die Begriffe *Schwachstelle*, *Bedrohung* und *Angriffsvektor*!

- **Schwachstelle**
  - Sicherheitskontrollen des Systems können umgangen werden
  - z.B. keine guten Passwort-Regeln
- **Bedrohung**
  - Umstand oder Ereignis mit dem **Potenzial**, ein System durch unbefugten Zugriff, Zerstörung, Offenlegung, etc. zu beeinträchtigen.
  - Eine Schwachstelle muss nicht immer eine Bedrohung sein!
  - z.B. DDoS-Angriff
- **Angriffsvektor**
  - **Konkreter Angriffsweg** um die Sicherheit des Systems zu gefährden
  - Nutzt eine oder mehrere Schwachstellen
  - z.B. keine guten Passwort-Regeln (1), kein TFA (2), kein Brute-Force Schutz (3) $\rightarrow$ Passwort herausfinden mit Brute-Force

Aufgabe 3a - Injections

a) Erläutern Sie möglichst allgemein, was eine Injection Vulnerability ist, und nennen Sie Beispiele für Stellen, an welchen eine solche Schwachstelle vorkommen kann!

- Eingabe des Benutzers wird ungeprüft an einen Interpreter weitergegeben
- Interpreter weißnicht, was vom Benutzer kommt und was nicht
  $\rightarrow$ Interpreter führt injizierten Code aus
- Bekannte Beispiele: SQL, LDAP, Shell
- z.B.

```
1  import os
2  ip = input("ip: ")
3  os.system(f"ping -c 1 {ip}")
```

Aufgabe 3 - SQL Basics

- **SQL** (Structured Query Language) ist eine Sprache, die uns erlaubt, CRUD (und weitere Operationen) auf relationalen Datenbanken auszuführen
- Relationale Datenbanken speichern Daten in Tabellen
- Um die Tabelleninhalte auszulesen, verwendet man den SELECT-Befehl
- Um Ergebnisse zu filtern, verwendet man die WHERE-Klausel
- Beispiel:

```
1  SELECT * FROM studenten
2  WHERE semester > 3
```

- Es gibt viele weitere Befehle - siehe Cheatsheet
- Ausführlichere Erklärung:
  `https://wiki.selfhtml.org/wiki/Datenbank/SQL-Grundlagen`

Aufgabe 3b - SQL Injection

b) Schauen Sie sich das folgende Beispiel zu SQL-Injections an. Gehen Sie davon aus, dass es aus einem fiktiven Nutzerverwaltungsprogramm stammt. Wie geht der Angreifer vor und was ist im gegebenen Beispiel sein Ziel?

1) SQL-Query zur Suche von Nutzern:

```
1  query = f"SELECT * FROM users WHERE vorname='{name}' AND id
   ↪  > 1000"
```

2) Angriff: `name = "' OR 1=1 --"`

ID	Vorname	Nachname	E-Mail	pwhash
1	Fabian	Franzen	franzen@sec.in.tum.de	$2b$12$5xuB/x
2	Julian	Kirsch	kirschju@sec.in.tum.de	$2b$12$V4q6AM
3	Claudia	Eckert	claudia.eckert@in.tum.de	$2b$12$MqIWEY
...	...	...	...	...
1001	Max	Muster	max.muster@gmail.com	$2b$12$lkjaAS
...	...	...	...	...

Aufgabe 3b - SQL Injection

b) Schauen Sie sich das folgende Beispiel zu SQL-Injections an. Gehen Sie davon aus, dass es aus einem fiktiven Nutzerverwaltungsprogramm stammt. Wie geht der Angreifer vor und was ist im gegebenen Beispiel sein Ziel?

- Der Inhalt von `name` wird direkt in die SQL-Abfrage eingefügt:

```
1  SELECT * FROM users WHERE vorname='' OR 1=1 --' AND id > 1000
```

- Die Teilbedingung `AND id > 1000` wird ausgehebelt
- Man bekommt Daten, die eigentlich geschützt sein sollten

Aufgabe 3c - sqlite_master

c) In der Datenbanklösung SQLite3 existiert eine Tabelle `sqlite_master`, welche Informationen über die existierenden Tabellen der Datenbank enthält. Dies kann nützlich sein, wenn Ihnen nicht bekannt ist, welche Datenbanktabellen eine Anwendung angelegt hat (z.B. weil Ihnen dessen Implementierung unbekannt ist). Wie würden Sie die SQL-Injection aus der vorherigen Aufgabe verändern, um diese Informationen auszulesen?

- SQL-Injection mit `UNION` den Inhalt von `sqlite_master` zusätzlich ausgeben
- Query:

```
query = f"select name from studenten where semester >
↪ {semester}"
```

- Payload: `semester = "10 union select name from sqlite_master"`
- Ergebnis:

```
select name from studenten where semester > 10 union select
↪ name from sqlite_master
```

- Ausgabe enthält Namen aller Tabellen

Aufgabe 3d - Blind SQL-Injection

d) In einigen Fällen werden die Ergebnisse einer verwundbaren Datenbankabfrage überhaupt nicht ausgegeben, sondern man erhält von der verwundbaren Webseite nur eine *richtig* oder *falsch* Meldung. Entwickeln Sie eine Methode, um trotzdem Informationen aus einer Webseite zu extrahieren.

- Ergebnis nur Wahr oder Falsch (z.B. erfolgreicher oder gescheiterter Login)
- Der gesuchte Wert wird zeichenweise per Brute-Force bestimmt.

Aufgabe 3d - Blind SQL-Injection - Beispiel

- Query:

```
1   query = f"SELECT * FROM users WHERE id='{id}'"
```

- Payload:

Payload	Ergebnis
`"1' AND 'A'=SUBSTR(vorname, 1, 1) --"`	Falsch
`"1' AND 'B'=SUBSTR(vorname, 1, 1) --"`	Falsch
⋮	⋮
`"1' AND 'F'=SUBSTR(vorname, 1, 1) --"`	Wahr
`"1' AND 'FA'=SUBSTR(vorname, 1, 2) --"`	Wahr
⋮	⋮