

Kurs: CY-B-3 Sichere Programmierung

Dozent: Michael Heigl

**Absichern eines terminalbasierten Online-Schachservers (Chess Online)**

Autoren:

Sandro Schamberger (22102471) [33%]

Florian Hagengruber (22101608) [33%]

Christian Joiko (22111097) [33%]

**Inhaltsverzeichnis**

**1. Einleitung**

Die Digitalisierung betrifft momentan fast jeden Bereich unseres Lebens. Sei es nun am Arbeitsplatz, im Auto oder im eigenen zuhause, die Digitalisierung ist mittlerweile omnipräsent. Doch so viele Vorteile der digitale Wandel auch bringt, so wird sie jedoch stets von einem schwerwiegenden Problem begleitet, dass leider noch nicht dieselbe Aufmerksamkeit wie die Digitalisierung erhält wie die Digitalisierung selbst. Bei dieser bösartigen Nebenwirkung handelt es sich um die Cyber Sicherheit oder besser gesagt der Mangel davon. Mit jeder digitalen Neuerung und jedem neuen Gerät bilden wir eine weitere Angriffsfläche, auf der wir von Cyberkriminellen attackiert werden können, weshalb diese Dienste bestmöglich geschützt werden müssen. Viele denken hierbei wahrscheinlich an kritische Infrastruktur wie Krankenhäuser oder Stromversorgung, wobei jedoch alle Aspekte hinreichend abgesichert werden müssen, auch wenn sie noch so unscheinbar sind.

Ein Beispiel hierfür kommt aus dem Bereich der Freizeitgestaltung, die Videospielindustrie. Doch warum ist Informationssicherheit hier so wichtig? Dank der Digitalisierung ist diese Freizeitaktivität mittlerweile weit verbreitet, sodass im Jahr 2021 bereits jeder Zweite in Deutschland ab 16. Jahren gelegentlich Videospiele spielt [1]. Allein 2021 wurden 6,17 Milliarden Euro für Videospielsoftware ausgegeben [1]. Somit entsteht ein sehr großes Risiko für Firmen, da Vorfälle in der IT-Sicherheit einen großen Reputationsschaden mit sich führen können, wodurch der Anteil der Firma an dieser Geldsumme drastisch sinken kann. Doch nicht nur die Anbieter und Produzenten selbst sind von Cyberangriffen betroffen, sondern auch die Konsumenten, welche ihre persönliche Daten, ihre Zahlungsinformationen und ihr investiertes Vermögen in die Hände der Firmen geben. Werden Nutzerdaten nicht ausreichend geschützt, so könnten diese gestohlen werden und zu Schäden beim Nutzer führen.

Doch wie schwer ist es eigentlich, ein Videospiel mit all seinen verknüpften Diensten und Nutzerdaten gegen Cyberkriminelle und Hacker zu schützen? Um diese Frage zu beantworten, wird in diesem Projekt ein Server, mit dem Nutzer gegeneinander Schach spielen können, genauer unter die Lupe genommen. Dabei werden zuerst mögliche Schwachstellen analysiert und aufgrund ihres Gefahrenpotentials für Nutzer und Firma bewertet. Anschließend werden die erkannten Schwachstellen repariert, um eine möglichst sichere Version zu erstellen. Durch den Vergleich der beiden Versionen und die Aufzählung der möglichen Angriffsvektoren wird versucht, diese Frage zu beantworten. [Neulicher IT-Vorfall bei Gaming fehlt!!]

**2. Funktionsweise der schwachen Version**

**2.1 Login und Menü**

Bevor ein Benutzer per „Chess Online“ Schach spielen kann, muss er mit dem Server eine Verbindung aufbauen. Dafür wird das Protokoll „Telnet“ verwendet, womit sich ein Client unter Eingabe der IP-Adresse des Servers und der Portnummer 8080 verbinden kann. Nachdem die Verbindung erfolgreich hergestellt wurde, wird dem Benutzer das Menü angezeigt, wo er aufgefordert wird, sich anzumelden oder sich zu registrieren, falls er noch keinen Account besitzt (siehe Abb. 1).

*Abbildung 1: Benutzer wird im Menü zum Login aufgefordert*

Für den Fall, dass der Benutzer noch kein Konto besitzt, kann er den Menüpunkt „Registration“ auswählen, um sich eines anzulegen. Für die Registrierung wird zuerst eine gültige Mailadresse eines Studenten oder Mitarbeiters der Technischen Hochschule Deggendorf benötigt. Die eingegebene E-Mail wird zur Kontrolle auf ihren Aufbau hin überprüft, ob sie eine Mailadresse der THD sein kann. Ist dies der Fall, so wird der Benutzer nach einem Passwort gefragt. Schlussendlich wird dem Nutzer dann mitgeteilt, dass ihm per E-Mail ein Aktivierungscode, bestehend aus fünf Ziffern, gesendet wurde (siehe Abb. 2). Die Mailadresse, das Passwort und der Aktivierungscode werden außerdem zusammen mit einem automatisch generierten Nutzernamen in der Tabelle „Spieler“ der Datenbank abgespeichert (siehe Abb. 3).

*Abbildung 2: Registrierung des Accounts (+ Hackermail abweisung)*

*Abbildung 3: Design der Datenbank*

Sollte der Nutzer bereits ein Konto besitzen, so kann er das Menü „Login“ benutzen, um sich anzumelden. Dafür wird seine Mailadresse und sein Passwort benötigt. Ist dies seine erste Anmeldung, so wird er zudem noch nach seinem Aktivierungscode gefragt (siehe Abb. 4). Wurde dieser richtig eingegeben, so ist der Benutzer eingeloggt und kann nun die Funktionen des Servers benutzen. Zu diesen zählen das Abrufen der Statistik eines anderen Spielers mittels einer Datenbankabfrage mit dem Befehl **--stats**, dem Ausloggen durch den Menüpunkt „Logout“**,** das Laden oder Starten eines Spiels gegen eine KI, dem Beitreten eines Online-Matches und dem Verlassen der Applikation. Im Zuge dieser Arbeit wird hauptsächlich der Vorgang eines Online-Matches betrachtet, da dieser Aspekt die meisten Risiken birgt.

*Abbildung 4: Anmeldung des Nutzers mit Abfrage des Aktivierungscodes*

**2.2 Während eines Online-Matches**

Entschließt sich ein eingeloggter Nutzer, dass er online eine Partie gegen einen anderen Spieler austragen möchte, so betritt er durch Eingabe des dazugehörigen Menüzeichens eine Wartschlange. Betritt eine weiterer Spiele die Warteschlange, so werden die Daten der beiden Kontrahenten in einem Array gespeichert und ein Spiel wird gestartet. Nun können die Nutzer abwechselnd durch die Eingabe einer Zeichenkette, bestehend aus dem Anfangsfeld und dem Zielfeld, ihre Figuren bewegen (e.g. „G1E1“). Neben diesen Aktionen können die Spieler Befehle wie **--stats**, **--surrender** und **--remis** verwenden.

Eine Runde kann aus verschiedenen Gründen enden. Die verschiedenen Möglichkeiten hierfür sind das manuelle Beenden mittels den oben genannten Befehlen **--surrender** und **--remis** (nur wenn sich beide Spieler für ein Remis einigen), das Besiegen des gegnerischen Königs oder das Verlassen der Runde beziehungsweise das Abbrechen der Verbindung. Letzteres führt hierbei zu einer automatischen Niederlage für den Spieler, welcher die Runde verlassen hat.

Am Ende der Runde wird das Ergebnis außerdem in der Datenbank vermerkt. Dazu zählt das Vermerken des Endergebnisses in der Tabelle „Spiele“, sowie das Abändern der Statistiken der Spieler. Dabei wird den beiden Spielern je nach Situation der Wert des Attributs „siege“, „niederlagen“ oder „remis“ um eins erhöht und die ELO-Bewertung wird für beide Spieler neu kalkuliert (siehe Abb. 5) [4 Quelle Wiki].

*Abbildung 5:Berechnung der ELO-Änderung*

**3. Analyse der Schwachstellen**

Im folgenden Kapitel wird der Code der schwachen Version analysiert und die Funktionen des Programms werden auf mögliche Angriffsmöglichkeiten und Schwächen untersucht. Die Schwächen werden dafür in Gruppen zu ihrer jeweiligen Programmfunktion eingeteilt. Dabei werden, nach einer kurzen Einführung zur Schwachstelle, zuerst die betroffenen Codestellen genannt und der Grund für die Schwachstelle aufgezeigt. Danach wird ein möglicher Angriff mit seinen möglichen Konsequenzen zur Veranschaulichung des Problems vorgestellt. Zuletzt wird die jeweilige Klassifizierung der CVE / CWE? (Dann eher Schwächen schreiben) und eine Bepunktung des Schweregrads angegeben. Die Punkteverteilung beinhaltet die allgemeine Punktzahl nach dem CVSS und eine eigene, ans Programm angepasste, Berechnung des Schweregrads. Zur Berechnung der Punktzahl wird der „Axelsec-Rechner“ verwendet.

**3.1 Datenbank**

Da „Chess Online“ zur Verwaltung der Nutzerdaten, der Spielehistorie und der Speicherstände eine Datenbank verwendet, sollte bei jedem Programmierer sofort die Gefahr einer SQL-Injection in den Sinn kommen. Die SQL-Injection ist nicht ohne Grund sowohl im Jahr 2020 als auch im Jahr 2021 auf Platz 6 der „CWE Top 25 Most Dangerous Software Weaknesses“ [3]. Die Möglichkeit einer SQL-Injection kann somit katastrophale Folgen haben, wie nachfolgend aufgezeigt wird.

Bei unserem Projekt werden die meisten Datenbankbefehle unabhängig von Eingaben des Nutzers ausgeführt. Jedoch werden für manche Abfragen Nutzereingaben zwingend benötigt, genauer gesagt bei den Datenbankabfragen, welche während der Anmeldung und der Registrierung benötigt werden. Die Ursache für die SQL-Injection ist eine Kombination aus dem Verwenden von schwachen Softwarekonstruktionen, obwohl sicherere vorhanden wären und dem Fehlen der Input Validation. Ersteres bezieht sich hierbei auf die Verwendung der Funktion **executescript()** und dem **%s-Platzhalter**.

Sollte ein Angreifer die Datenbank per SQL-Injection angreifen wollen, so muss er lediglich die richtige Anzahl der Parameter für das SQL-Statement herausfinden, sodass dieses ausgeführt wird und nicht vor der Ausführung abbricht. Da dafür jedoch ein wenig Trial-and-Error genügt, kann der Angreifer schnell, einfach und, im Falle der Registrierung, ohne Account und funktionierende Mailadresse per SQL-Injection die Datenbank in vollem Umfang nach seinen Wünschen manipulieren oder sich bei anderen Accounts ohne Passwort anmelden. Einzig die Validierung der Mailadresse erschwert dem Angreifer das Herausfinden einer korrekten Eingabe, da seine Injection die Überprüfung des Domänenteils überstehen muss und dann bereits beim Überprüfen, ob diese Mailadresse bereits registriert ist, ausgeführt werden muss. Jedoch ist dies nicht allzu schwer und kann durch Umsteigen auf die anderen drei Eingabefelder sogar vollständig umgangen werden. Nachfolgend wird eine mögliche Eingabe für beide Vorgänge mit ihren Folgen für die Anwendung aufgezeigt, wobei ein Stern (\*) für eine beliebige Texteingabe steht:

|  |  |  |
| --- | --- | --- |
| **Registrierung** | | |
| Feld | Eingabe | Konsequenzen |
| E-Mail | \*@th-deg.de | Löscht die Tabelle Spieler mit allen Accountdaten |
| Passwort | \*‘, ‘\*‘, ‘\*‘); DROP TABLE Spieler;-- |
| E-Mail | \*' OR 1 = 1; UPDATE Spieler SET siege = 1000 WHERE nutzername = 'c.joiko';--@th-deg.de | Setzt die Anzahl der Siege des Spielers mit dem Nutzernamen „c.joiko“ auf 1000 |
| Passwort | *Eingabe egal* |
|  | | |
| **Anmeldung** | | |
| Feld | Eingabe | Konsequenzen |
| E-Mail | *Eingabe egal* | Einloggen mit ersten registrierten Account |
| Passwort | \* ‘ OR True;-- |

[Punkte!!! CVSS oder CWSS???]

**3.2 Passwort und Aktivierungscode**

Das Passwort ist wahrscheinlich eines der ältesten und am weitest verbreiteten Authentifizierungsverfahren. Sie sind einfach als Schutz zu implementieren und können trotz ihrer Simplizität dennoch großen Schutz bitten, vorausgesetzt sie werden auch richtig verwendet. Noch immer verwenden fast 50% der Deutschen meistens Passwörter mit einer Länge von maximal 10 Zeichen [Quelle]. In Kombination mit zu einfach gewählten Passwörtern wie „hallo123“ oder dem Notieren an unsicheren Orten verlieren sie relativ schnell jegliche Bedeutung beim Schutz der eigenen Daten, was nachfolgend auch bei „Chess Online“ aufgezeigt wird.

Das erste Problem entsteht bereits bei der Erstellung des Passwortes bei der Registrierung eines neuen Accounts. Das Passwort wird bei der Eingabe am Terminal angezeigt und kann somit bereits durch einen Blick auf den Bildschirm erlangt werden. Zudem wird der Nutzer nicht um eine Bestätigung des Passwortes gebeten, was in Kombination mit der fehlenden Möglichkeit zum Ändern des Passwortes bei einem Tippfehler schnell zu einem Problem führt. Doch die eingeschränkte Effektivität der Passwörter ist nicht nur Schuld des Benutzers, sondern auch des Programmentwicklers. Die Passwörter werden kein einziges mal verschlüsselt und liegen permanent als Plaintext vor. Zudem wurde von Programmierern ein Passwort für einen Account im Quelltext vergessen.

Neben den Passwörtern weisen auch die Aktivierungscodes zum Freischalten eines neuen Accounts einen verheerenden Mangel auf. Die 5-stelligen Codes sind sehr einfach per Brute-Force-Methode zu knacken, was aufgrund der fehlenden Limitierung der Eingabeversuche zu einer kompletten Verlust der Authentifizierungsfunktion des Aktivierungscodes führt.

[Aktivierungscode bruteforcen]

**4. Beheben der Schwächen**

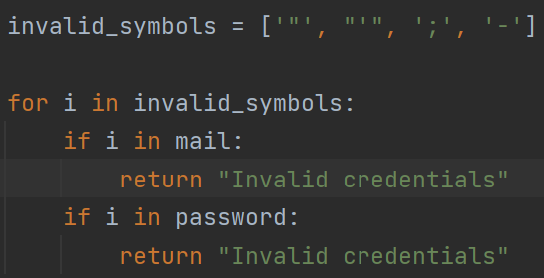
**4.1 Datenbank**

Um die in 3.1 angesprochene SQL-Injection zu beheben, sollten zuerst die verwendeten unsicheren Softwarekonstruktionen durch sichere Alternativen ersetzt werden. Die erste Änderung ist das Ersetzen der Funktion **executescript ()** durch **execute()**. Der Unterschied zwischen diesen beiden Funktionen ist, dass **execute()** nur einen SQL-Befehl ausführen kann, während **executescript()** mehrere Statements verarbeitet. Da unsere Funktionen zur Manipulation der Datenbank sowieso immer nur ein Statement ausführen müssen, ist dies ein sehr einfacher Schutz vor angehängten, unerwünschten SQL-Statements des Angreifers. Nach Anwendung des Fixes sind die beiden Injections, welche zuvor während der Registrierung möglich waren, nicht mehr möglich (siehe Abb. 6).



*Abbildung 6: Fehlermeldung nach Ausführung der ersten SQL-Injection*

Leider schützt diese Verbesserung noch nicht vor dem Login ohne Passwort, weshalb eine weitere Verbesserung vorgenommen werden muss. Diese Verbesserung betrifft die Platzhalter in den SQL-Statements, welche später durch die Eingaben des Nutzers ersetzt werden. Die schwache Version verwendet momentan **%s-Platzhalter**, welche laut der Dokumentation des sqlite3-Moduls in Python unsicher und anfällig für SQL-Injections sind. Als Verbesserung wird hierfür die Verwendung der moduleigenen **?-Platzhalter** vorgeschlagen, welche nach Einbindung die dritte SQL-Injection verhindern sollte [4]. Da dies jedoch nicht der Fall ist, wird zusätzlich der Input des Users auf maliziöse Absichten überprüft, bevor er verarbeitet wird. Um dies zu erreichen, wird einfach ein Array mit verbotenen Zeichen eingefügt und über dieses iteriert, ob in der eingegebenen Mailadresse oder dem Passwort ein verbotenes Zeichen vorhanden ist. Um die SQL-Injection zu verhindern, ohne den zulässigen Zeichensatz für Passwörter unnötig einzuschränken wird sich im Projekt auf vier Sonderzeichen beschränkt (siehe Abb. 7).



*Abbildung 7: Einbindung und Überprüfung der invaliden Zeichen*

Gegen die ungesicherte und unverschlüsselte Ablage der Datenbank sollte ebenfalls vorgegangen werden. Dazu könnte beispielsweise **SQLCipher** verwendet werden [Quelle?]. Zur weiteren Absicherung der Datenbank sollte ein regelmäßiges Backup der Datenbank gemacht werden, damit im Falle der ersten beiden SQL-Injections der Schaden zumindest wieder behebbar wäre und die Integrität nicht vollständig verloren geht. Diese beiden Fixes wurden jedoch aus Zeitgründen und der höheren Relevanz und Gefährlichkeit der SQL-Injection nicht ins Programm eingebaut.