

Inhaltsverzeichnis

Sicherheit im eigenen Browserspiel.....	1
Einleitung.....	1
Struktur.....	1
GET und POST.....	2
Manipulation von GET/POST.....	3
Manipulation der SESSION.....	7
Schreiben von Eingabe-Daten in die Datenbank.....	7
XSS.....	9
Links.....	9

Sicherheit im eigenen Browserspiel

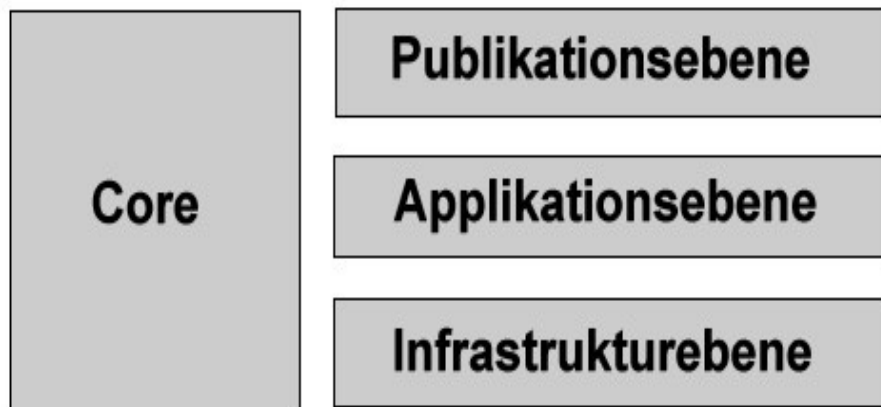
Einleitung

Dieses kleine Tutorial ist vorallem praktisch orientiert und soll mit vielen kleinen Beispielen zeigen wie man sein eigenes Browsergame in PHP/mysql sicher vor Cheater macht.

Struktur



Die Ordnerstruktur eines Browsergame ist sehr wichtig – nicht zuletzt um den Überblick über das Projekt zu wahren. Wichtigster Teil ist der Core (Kern). In diesem Ordner sind alle wichtigen Stripte drin. Infrastrukturebene, Applikationsebene und Publikationsebene.



Infrastrukturebene: Scripte welche bei fast jedem Seitenaufruf benötigt werden und grundlegende Sachen regeln wie beispielsweise Datenbankzugriff, Sicherheit, Standardmethoden und Session-Prüfung

Applikationsebene: Regelt die seitenspezifischen Rechenprozesse. Diese Scripte werden nur von einigen Seiten benötigt und greift dabei auf die Infrastrukturebene zu.

Publikationsebene: Verarbeitet die Daten von der Applikationsebene und generiert den Content einer Seite.

Dieser Ordner wird mit einer .htaccess-Datei vor unbefugtem Zugriff geschützt:

.htaccess

```
Deny from all
```

Das ist alles was zwingend in .htaccess-Datei rein kommt. Einfach leere Datei im Editor öffnen, Text reinschreiben und als Text abspeichern. Dann auf den Server kopieren und dort in .htaccess umbenennen. Damit ist nun sichergestellt das die Core-Scripte nur vom Interface geöffnet werden können. Versucht jetzt jemand externes eine Seite aus dem Core direkt aufzurufen erhält er eine Fehlermeldung.

GET und POST

Noch vor PHP5 war es alltäglich Variablen welche per GET/POST übertragen wurden einfach mit \$myvar anzusprechen. Glücklicherweise sind die Zeiten vorbei! Da die Gefahr wohl jedem Anfänger klar sein sollte gehe ich hierrauf nicht weiter ein. Doch etwas

anderes möchte ich darum umsomehr ins Gewissen rufen: **GET/POST ist nicht sicher und kann jederzeit manipuliert werden**. Auch innerhalb einer Session!! Wie man dies macht lernt ihr hier und jetzt damit ihr die Tricks der Cheater selber testen könnt.

Manipulation von GET/POST

Ein kleines Beispiel vorab: Ein User hat in seinem Account ein Gebäude (adäquat jedes andere Objekt welches einen Datensatz mit einer eindeutigen ID für sich beansprucht) welches er durch einen Formular-Button abreissen kann. Daher sein angezeigtes Formular:

```
<form action="" method="post">
  <fieldset>
    <legend>Gebäude abreissen</legend>
    <input type="hidden" name="geb_id" value="9263" />
    <input type="submit" name="submit" value="Gebäude &nbsp; abreissen" />
  </fieldset>
</form>
```



The image shows a browser window displaying a form. At the top, there is a legend with the text 'Gebäude abreissen'. Below the legend is a submit button with the text 'Gebäude abreissen'.

Und im Hintergrund haben wir noch das Script, welches anschliessend den Gebäude-Datensatz löschen soll:

```
<?
$id = $_POST['id'];
mysql_query("DELETE FROM `table` WHERE `id` = ".$id." ")or die( mysql_error() );
?>
```

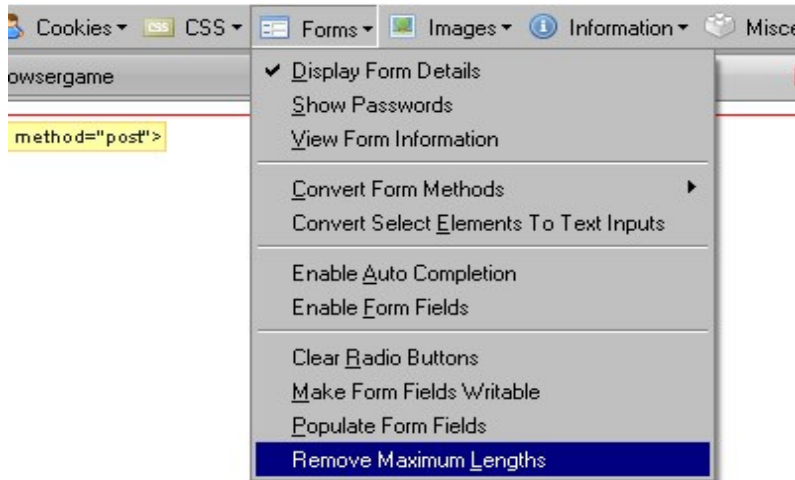
Nun, das Script oben erfüllt zwar seinen Zweck und würde auch den Datensatz löschen, doch lässt sich das Script noch zugleich missbrauchen!!

Folgendes ist von Nöten:

<http://www.firefox2.de/> (den besten Browser, grade wegen seiner AddOns)
<https://addons.mozilla.org/firefox/60/> (das WebDeveloper-Tool englisch)
http://www.erweiterungen.de/detail/Web_Developer/ (oder das WebDeveloper-Tool

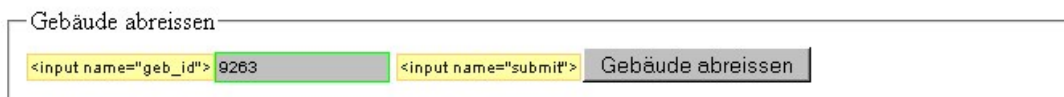
deutsch)

Sobald ihr das WebDeveloper-Tool im Firefox installiert und geladen habt fahren wir fort:



— Gebäude abreißen —

Der schwerwiegendste Fehler der hier gemacht worden ist: Mögliche mySQL-Injection. Eine mySQL-Injection ist eine Sicherheitslücke die es einem potenziellen Angreifer erlaubt falschen Code in die Funktion `mysql_query()` zu schleusen und dadurch Schaden anzurichten. Hauptangriffspunkt ist das versteckte Formularelement. Um den Angriff selber durchzuführen klicken wir in der WebDeveloper-Toolbar auf "Forms"->"Display Form Details". Nun sind alle Formularelemente sichtbar und modifizierbar. Selbstredend auch das versteckte Formularlement:



In einem gelben Input-Feld steht dort die Zahl 9263. Ersetzen wir die Zahl mit `"1 || `id`!=1"` (ohne Anführungszeichen) und schicken das Formular ab. Anstelle der Query zum löschen eines Datensatzes erhalten wir folgende Query:

```
mysql_query("DELETE FROM `table` WHERE `id`=1 || `id`!=1 ")or die(mysql_error());
```

Die Query löscht nun alle Datensätze mit id gleich 1 oder id nicht gleich 1 – daher alle. Ihr habt nen Monat in eurem Browsergame gespielt und diese Angriffsmöglichkeit nicht gekannt und nun löscht euch jemand alle im Spiel gebauten Gebäude, dumm gelaufen.

Wie begegnen wir diesem Problem? Nun eigentlich ganz einfach, wir wissen das per POST eine Zahl ankommen sollte. Also machen wir einfach:

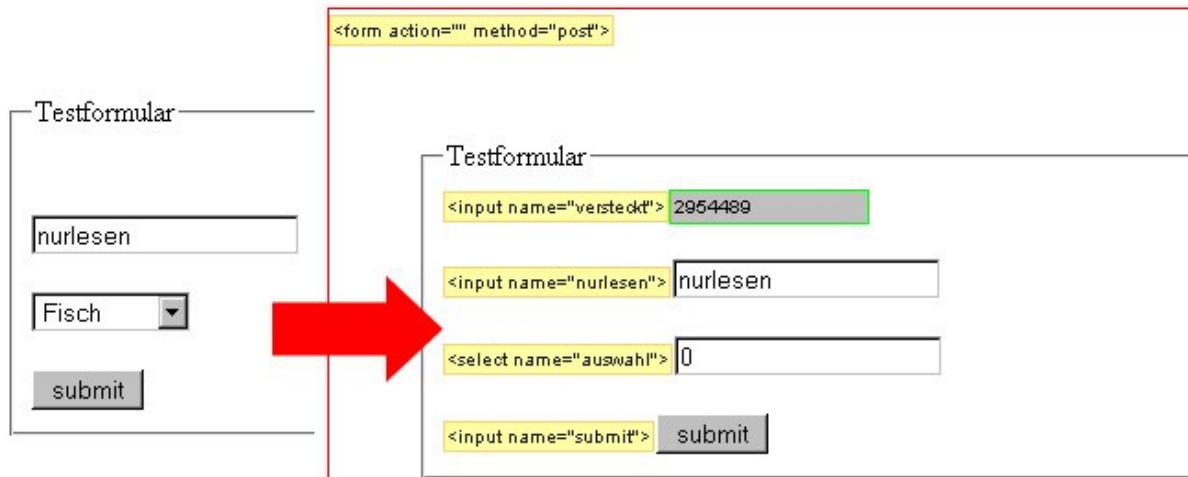
```
$id = (int) $_POST['id'];
```

(int) verwandelt alle strings und floats eben in Ganzzahlen. Doch so richtig sicher ist dies noch nicht, wir wissen ja och nicht ob der User überhaupt die Erlaubnis hat eben gerade DIESES Gebäude abzureissen. Er könnte genausogut die ID eines Gebäudes eines andern Spielers übergeben haben. Der userspezifische Datensatz muss auch die user_id tragen und ist damit mit dem User assoziiert:

```
mysql_query("DELETE FROM `table` WHERE `id`=".$id."&& `user_id`=".$_SESSION['user_id'].")or die( mysql_error() );
```

Damit kann der User nun auch wirklich nur noch Gebäude löschen welche ihm selbst gehören. Mit einem einfachen mysql_affected_rows() können wir nun prüfen ob die Aktion erfolgreich war. War sie es nicht dann hat der User eine falsche Nummer übergeben. Und jetzt der Vorteil: In diesem Fallback können wir auch gleich eine Funktion aufrufen welche den Account des Users bannt und ihn aus dem Spiel schmeisst. Potenzielle Cheater werden dadurch ausgebremst und ihnen wir nicht die Möglichkeit gegeben das Spiel ausgiebig nach Schwachstellen zu durchsuchen.

Das gleiche Prinzip funktioniert übrigens auch bei select-Dropdowns, gesperrten input-Feldern und dergleichen. Alles lässt sich bequem manipulieren:



[Collapse All](#) [Expand All](#)

file:///D:/test/test.html

1 form

Form

ID	NAME	METHOD
		post

Elements

INDEX	ID	NAME	TYPE	VALUE
0			fieldset	
1		versteckt	text	2954489
2		nurlesen	text	nurlesen
3		auswahl	text	0
4		submit	submit	submit

Bevor wir das Thema GET/POST abhaken können gibt's noch eine kleine Problematik aufzuzeigen: Browserfenster. Nicht dass das jetzt etwas schlechtes wäre, im Gegenteil. Doch unter manchen Bedingungen kann sich hier auch eine Cheat-Möglichkeit verstecken. Folgende Situation: Ein User lädt ein Formular in mehrere Browserfenster

gleichzeitig und drückt nun im ersten Browserfenster einen Knopf. Das Script rechnet und die Erfolgsseite wird angezeigt mit dem gesperrten Formularfeld (bzw. nicht mehr angezeigten). Nun switcht der User ein Browserfenster weiter und vollführt dort nochmals dieselbe Aktion (Gebäude bauen, Forschung starten was auch immer). Dies kann also dazu führen das Aktionen gleichzeitig laufen obwohl dies in einem Browserfenster allein nicht möglich wäre.

Manipulation der SESSION

<i>Fleisch</i> <i>Gemüse</i>	<i>Schnitzel</i> <i>Salami</i>
	<i>Fenchel</i> <i>Salat</i>

Die Session zu verändern ist nur dort möglich wo ein Browsergame eine Lücke offen hält. Ein kleines Beispiel dazu. Wir haben ein Formular 1, welches uns Informationen und Auswahlmöglichkeiten anzeigt. Je nach Auswahl wird der User zu Formular 2 weitergeleitet. Die Daten von Formular 1 werden in der Session gespeichert. In Formular 2 kann der User nun wieder etwas auswählen und nach dem abschicken werden alle Daten in der DB gespeichert. Halt. Die Daten aus Formular 1 und Formular 2 müssen nicht zwingend überein stimmen. Auch wenn nach der Auswahl von Formular 1 nur bestimmte Teile von Formular 2 abgezeigt werden ist es doch möglich in diesem all die Daten in der Session zu verändern. Dazu nötig ist nicht mal ein Tool, ein neues Browserfenster reicht. In die zwei Browserfenster öffnen wir nun Formular 1, zwei mal. Formular 1 bietet uns zwei Auswahlmöglichkeiten: Fleisch oder Gemüse. Wir entscheiden uns für Fleisch und sehen in Formular 2 Fleischsorten. Würden wir jetzt dort etwas auswählen und das Formular absenden wäre alles in Butter: Kategorie Fleisch und Unterkategorie Schnitzel (z.B.) würden in die Datenbank gespeichert werden. Nun aber das Problem mit dem Browserfenster. Anstatt Formular 2 abzuschicken wählen wir erst mal im andern Browserfenster "Gemüse" aus und schicke das ab. Dort erscheint nun die Gemüseauswahl. Die ignorieren wir aber und wechseln wieder zu unserer Fleischauswahl und senden das Formular nun. Kategorie "Gemüse" und Unterkategorie "Schnitzel". Ihr seht schon worauf es hinaus läuft... Um hier die Session in der Bahn zu halten müssen wir erst prüfen ob die Daten aus Formular 1 auch zu denen aus Formular 2 passen. In einem komplexen Browsergame meist gar nicht so einfach.

Schreiben von Eingabe-Daten in die Datenbank

Wichtigste Regel: Nur Erlauben was unbedingt nötig. Wird ein vom User definierter Benutzername in die Datenbank geschrieben brauchen nicht alle Sonderzeichen erlaubt zu

sein. Daher stellt sich die Frage welche Sonderzeichen sollen alle erlaubt sein? Es existieren zwei verschiedene Filterverfahren: Das Blacklist-Verfahren und das Whitelist-Verfahren. Ersteres sucht nach Zeichen bzw. Wörtern in der Blacklist und scheitert wenn eines gefunden. Das Whitelist-Verfahren läuft genau andersrum: Es definiert alles erlaubte und scheitert wenn etwas nicht erlaubt ist. Für Sonderzeichen ist ganz klar das Whitelist-Verfahren zu bevorzugen da es einfacher und sicherher ist:

```
function checkAllowedChars( $string ){  
    if( preg_match('#^[\\w\\d\\-\\ä\\ö\\ü]+$#is', $string ) ){  
        return true;  
    }  
    return false;  
}
```

Die Funktion prüft nun mittels einer RegEx (Regular Expression) auf gut oder böse. Als gut definiert sind alle Zeichen a-z bzw. A-Z, alle Zahlen 0-9, die zwei Stricke _ und – sowie die kleinen deutschen Umlaute.

Auch eine E-Mailadresse lässt sich validieren, eine einfache Funktion dazu sieht so aus:

```
function checkEmail( $string ){  
    if( preg_match('#^[\\w-]+(?:\\.\\[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}$#is', $string ) ){  
        return true;  
    }  
    return false;  
}
```

ABER: Das heisst jetzt noch lange nicht das die E-Mailadresse auch wirklich gültig ist, vielmehr ist nur sichergestellt das die Zeichenabfolge stimmt. Um die E-Mail wirklich zu prüfen müsst ihr vom Benutzer eine Bestätigung verlangen, daher ihr schickt ihm automatisch einen Aktivierungs-Link oder ähnlich.

Problematisch wird nun wenn wir dem User quasi alles erlauben müssen, daher wenn er eine Nachricht in ein Textfeld schreiben darf oder ähnlichen. Aber auch dies lässt sich meistern. Um gleich vorneweg einer Gefahr durch Code vorzubeugen wandeln wir alle Sonderzeichen in die entsprechenden HTML-Zeichen um. Bevor das ganze dann aber in die Datenbank kommt ist noch eine zweite Funktion wichtig: `mysql_real_escape_string()`. Diese Funktion maskiert verdeckt alle Zeichen die jetzt noch irgendwie für eine MySQL-Injection taugen können. Der fertige Code sieht so aus:

```
$input = mysql_real_escape_string( htmlspecialchars( $_POST['input'] ) );
```

Damit können wir nun sicher sein das auch hier keine Angriffsmöglichkeit mehr besteht.

XSS

Cross Site Scripting ist die Technik mittels Schadcode Informationen über eine Website heraus zu finden. Funktioniert überall dort wo Eingabe-Daten nicht validiert oder zumindest escaped werden.

```
echo $_POST['input'];
```

Mit dem Code oben wird direkt der Text aus einem Textfeld dargestellt – ohne ihn zu prüfen, escapen oder die Sonderzeichen umzuwandeln. Ein böser User schreibt nun

```
<script type="text/javascript">  
alert( 'XSS-Angriff' );  
</script>
```

in das Textfeld und schickt es ab. Sobald der Inhalt des Textfeldes angezeigt wird erhält der Lesende eine Javascript-Nachricht. Natürlich harmlos.

```
<script type="text/javascript">  
document.location.href = 'http://www.meine_seite.de';  
</script>
```

Schon schlimmer: Sobald die Seite geladen ist wird der Lesende auf die Internet-Adresse des Angressorts weitergeleitet. Der Hacker könnte auch Daten im Hintergrund zu seiner Seite senden ohne das der User etwas merkt. Damit könnte er wichtige Informationen preis geben welche der Hacker dann wiederum nutzen kann um im Spiel selber Vorteile zu schaffen.

Links

<http://www.tu-chemnitz.de/urz/www/php/rsrc/phpsec.pdf>

<http://www.flashforum.de/forum/forumdisplay.php?f=65>