CROSS-SITE REQUEST FORGERY

IN-DEPTH ANALYSIS • CYBER GATES • 2011

/*
INTRODUCTION
****/***

Cross-Site Request Forgery (CSRF in short) is a kind of a web application vulnerability which allows malicious website to send unauthorized requests to a vulnerable website using active session of its authorized users.

In simple words, it's when an **"evil"** website posts a new status in your twitter account on your visit while the login session is active on twitter.

/*
CSRF BASICS
****/***

A simple example of this is the following hidden html code inside the evil.com webpage:

```
<img src="http://twitter.com/home?status=evil.com" style="display:none"/>
```

Many web developers use POST instead of GET requests to avoid this kind of a malicious attack. But this approach is useless as the following html code may be used to bypass that kind of a protection.

```
<div style="display:none">
<iframe name="hiddenFrame"></iframe>
<form name="Form" action="http://site.com/post.php" target="hiddenFrame"
method="POST">
<input  type="text"  name="message"  value="I like www.evil.com" />
<input type="submit" />
</form>
<script>document.Form.submit();</script>
</div>
```

/*
USLESS DEFENSES
****/***

The following are the useless defenses:

1. Only accept POST
   This stops simple link-based attacks (IMG, frames, etc.)
   But hidden POST requests can be created with frames, scripts, etc
2. Referrer checking
   Some users prohibit referrers, so you can't just require referrer headers
   Techniques to selectively create HTTP request without referrers exist
3. Requiring multi-step transactions
   CSRF attack can perform each step in order

*/**
DEFENSE
**/*
The approach used by many web developers is the CAPTCHA systems and one-time tokens.
CAPTCHA systems are widely used but asking a user to fill the text in the CAPTCHA image every time user submits a form will make him/her leave your website. And that’s why one-time tokens are used instead.
Unlike the CAPTCHA systems one-time tokens are unique values stored in a webpage form hidden field and in session at the same time to compare them after the page form submission.
Mechanism used to generate one-time tokens can be found using brute force attacks. But brute forcing one-time tokens is useful only if the mechanism is widely used by web developers.
For example the following PHP code:

```
<?php
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
?>
```

*/**
DEFENSE USING ONE-TIME TOKENS
**/*
To understand better how this system works, let's take a look to a simple webpage which has a form with one-time token:

index.php(Victim website)

```
<?php session_start();?>
<html>
<head>
      <title>GOOD.COM</title>
</head>
<body>
<?php
$token = md5(uniqid(rand(),true));
$_SESSION['token'] = $token;
?>
<form name="messageForm" action="post.php" method="POST">
      <input type="text" name="message">
      <input type="submit" value="Post">
      <input type="hidden" name="token" value="<?php echo $token?>">
</form>
</body>
</html>
```

And the webpage which processes the request and stores the message only if the given token is correct:

post.php (Victim website)

```
<?php
session_start();
if($_SESSION['token'] == $_POST['token']){
      $message = $_POST['message'];
      echo "<b>Message:</b><br/>".$message;
      $file = fopen('messages.txt','a');
      fwrite($file,$message."\r\n");
      fclose($file);
} else {
      echo 'Bad request.';
}
?>
```

/*
IN-DEPTH ANALYSIS
****/***

In-depth analysis showed that an attacker might use an advanced version of the framing method to perform the task and send POST requests without guessing the token.

index.php (Evil website)

```
<html>
<head>
      <title>BAD.COM</title>
      <script language="javascript">
      function submitForm(){
      var token =
      window.frames[0].document.forms["messageForm"].elements["token"].value;
      var myForm = document.myForm;
      myForm.token.value = token;
      myForm.submit();
      }
      </script>
</head>
<body onLoad="submitForm();">
<div style="display:none">
      <iframe src="http://good.com/index.php"></iframe>
      <form name="myForm" target="hidden" action=http://good.com/post.php
      method="POST">
            <input type="text" name="message" value="I like www.bad.com" />
            <input type="hidden" name="token" value="" />
            <input type="submit" value="Post">
      </form>
</div>
</body>
</html>
```

For security reasons the **same origin policy** in browsers restricts access for browser-side programming languages such as JavaScript to access a remote content and the browser throws the following exception:

```
Permission denied to access property 'document'
var token = window.frames[0].document.forms['messageForm'].token.value;
```

Browser's settings are not hard to modify. So the best way for web application security is to secure web application itself.

*/**
FRAME BUSTING
**/*

The best way to protect web applications against CSRF attacks is using FrameKillers as well as one-time tokens.
FrameKillers are small piece of javascript codes used to protect web pages from being framed:

```
<script type="text/javascript">
  if(top != self) top.location.replace(location);
</script>
```

Different FrameKillers are used by web developers and different techniques are used to bypass them:

```
<script>
window.onbeforeunload=function(){
    return "Do you want to leave this page?" ;
}
</script>
<iframe src="http://www.good.com"></iframe>
```

*/**
BEST PRACTICES
**/*

And the best example of FrameKiller is the following:

```
<style> html{ display : none; } </style>
<script>
if( self == top ){ document.documentElement.style.display='block';}
else { top.location = self.location; }
</script>
```

Which protects web application even if an attacker browses the webpage with javascript disabled option in the browser.

*/**

REFERENCES

**/*

1. Cross-Site Request Forgery
   http://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29
   http://projects.webappsec.org/w/page/13246919/Cross-Site-Request-Forgery
2. Same Origin Policy
   http://en.wikipedia.org/wiki/Same_origin_policy
3. FrameKiller(Frame Busting)
   http://en.wikipedia.org/wiki/Framekiller
   http://seclab.stanford.edu/websec/framebusting/framebust.pdf

*/**

AUTHOR

**/*

Samvel Gevorgyan
Founder & Managing Director, CYBER GATES
www.cybergates.am
samvel.gevorgyan@cybergates.am

Samvel Gevorgyan is Founder and Managing Director of CYBER GATES Information Security Consulting, Testing and Research Company and has over 5 years of experience working in the IT industry. He started his career as a web designer in 2006. Then he seriously began learning web programming and web security concepts which allowed him to gain more knowledge in web design, web programming techniques and information security.
All this experience contributed to Samvel's work ethics, for he started to pay attention to each line of the code for good optimization and protection from different kinds of malicious attacks such as XSS(Cross-Site Scripting), SQL Injection, CSRF(Cross-Site Request Forgery), etc. Thus Samvel has transformed his job to a higher level, and he is gradually becoming more complete security professional.