



ADITYA K SOOD

Auditing Rich Internet Applications – Testing RIA Strategically

Traversing along Flash and FLEX

Difficulty



This research deals with insecurities in designing FLEX based applications from a developer perspective. The application's behavior depends on code written at the backend. It has been noticed that most of an application's flaws are the outcome of insecure or bad code.

It results in the generation of an application attack surface through which a number of attacks can occur. With ever increasing technology elements, the complexity of applications have also increased, but it is necessary to construct robust applications that are not vulnerable to attack. The development play critical role in this. The Rich Internet Application is a cross platform framework which provides an environment to run server based applications as desktop applications. For Example AIR. The specific AIR applications are written in FLEX Builder and are used as a single package. Usually AIR applications are also considered to be FLEX based applications. The only difference is deployment of those applications. The AIR applications run in Adobe run-time as singlet desktop application. There are number of problems in designing secure and effective FLEX applications, this research sheds light over those insecurities covering security impacts.

Scope

The applications can be run as cross platform applications. They are designed as unanimously and can be run on any platform such as windows, LINUX and MAC OS. The client requires proper run-time environment to be installed on the system which will undertake the applications as such. The applications use web technologies for development of desktop applications. The trend is changing and the scope is very versatile.

About

This paper covers the strategic testing procedure for testing rich internet applications including flash, flex and air. The basic structure of this methodology is to design procedures that are equally applicable to all environments. A hierarchical model is implemented with detailed examples and semantics used to perform the tests. This helps user to test the applications effectively.

Integrated Working Model FLEX [RIA]: Brief Overview

The development of AIR applications can be strategically done in FLEX. FLEX is an application building framework for creating applications to run in flash player [SWF] or Adobe Run Time environment. [AIR]. In this, MX calling convention is used for the application development. The name-space specification is required for setting the standard upon which application works. The URI is specified with XMLNS: MX. The finest part is the generation of XML file simultaneously with the method file comprised of instructions to be followed. The integrated model is presented below. Based on this model, standard components and AIR applications are designed in FLEX (see Figure 1).

This model serves as the base for developing RIA applications. When creating a window style application, the base tag `<mx:Windowed Application>` is used. Numbers

WHAT YOU WILL LEARN...

User will learn about the testing and auditing of Rich Internet Applications.

Detailed methodology with tools usage.

User will learn new techniques and the way to apply them in real time scenarios.

WHAT YOU SHOULD KNOW

Basic knowledge of Rich Internet Applications will be useful.

Knowledge of auditing RIA tools will be an advantage.

of objects are called under this tag. For implementing Action Script, the `<MX:script>` tag is used. In this the script elements are invoked under CDATA structure. On the other side CSS style scripts are called by specifying an `<MX:style>` tag.

A very general flow of code is presented below:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx=
http://www.adobe.com/2008/mxml
layout="absolute"
title="AIR Security Checks">
<mx:Label text="AIR Security Checks"
horizontalCenter="0"
verticalCenter="0"/>
</mx:WindowedApplication>
```

The output of the code see Figure 2. The application is constructed in this way. A simple label text is undertaken as output. For enhanced development and component designing Action Script 3.0 is preferred. So this brief overview presents how the applications are generated under FLEX. Now we will discuss the coding problems and will see the relative security impacts on the system.

Analytical View

The proper design of code plays a crucial role in optimizing FLEX applications. The structure of the code presents a standard

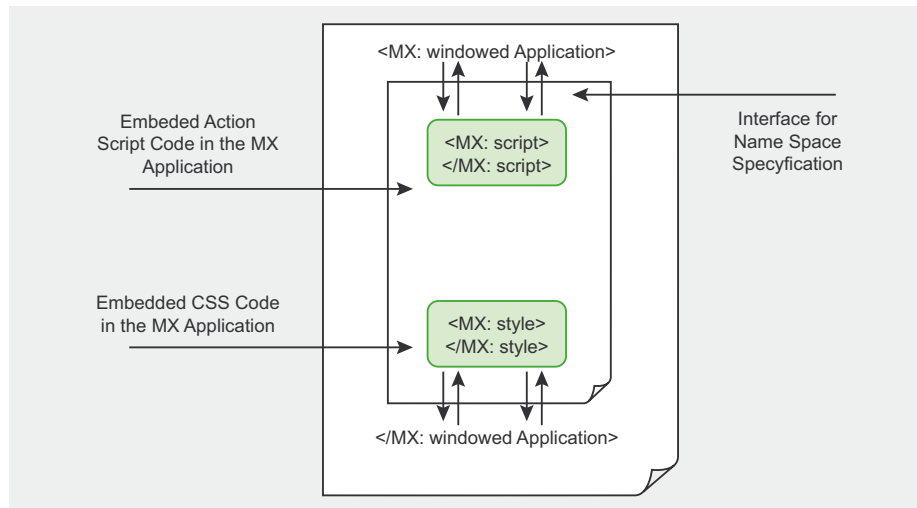


Figure 1. Internal view of RIA

Figure 2. Example of RIA running

Listing 1. Decompiling SWF file

```
movieClip 37 FPushButtonSymbol {

#initclip
function FPushButtonClass() {
    this.init();
}

FPushButtonClass.prototype = new FUIComponentClass();
Object.registerClass('FPushButtonSymbol',
    FPushButtonClass);
FPushButtonClass.prototype.init = function () {
    super.setSize(this._width, this._height);
    this.boundingBox_mc.unloadMovie();
    this.attachMovie('fpb_states', 'fpbState_mc', 1);
    this.attachMovie('FLabelSymbol', 'fLabel_mc', 2);
    this.attachMovie('fpb_hitArea', 'fpb_hitArea_mc', 3);
    super.init();
    this.btnState = false;
    this.setClickHandler(this.clickHandler);
    this._xscale = 100;
    this._yscale = 100;
    this.setSize(this.width, this.height);
    if (this.label != undefined) {
        this.setLabel(this.label);
    }
    this.ROLE_SYSTEM_PUSHBUTTON = 43;
    this.STATE_SYSTEM_PRESSED = 8;
    this.EVENT_OBJECT_STATECHANGE = 32778;
    this.EVENT_OBJECT_NAMECHANGE = 32780;
    this._accImpl.master = this;
    this._accImpl.stub = false;
    this._accImpl.get_accRole = this.get_accRole;
    this._accImpl.get_accName = this.get_accName;

    this._accImpl.get_accState = this.get_accState;
    this._accImpl.get_accDefaultAction = this.get_accDefaultAction;
    this._accImpl.accDoDefaultAction = this.accDoDefaultAction;
};

FPushButtonClass.prototype.setHitArea = function (w, h) {
    var hit = this.fpb_hitArea_mc;
    this.hitArea = hit;
    hit._visible = false;
    hit._width = w;
    hit._height = arguments.length > 1 ? h : hit._height;
};

FPushButtonClass.prototype.setSize = function (w, h) {
    w = w < 6 ? 6 : w;
    if (arguments.length > 1) {
        if (h < 6) {
            h = 6;
        }
    }
    super.setSize(w, h);
    this.setLabel(this.getLabel());
    this.arrangeLabel();
    this.setHitArea(w, h);
    this.boundingBox_mc._width = w;
    this.boundingBox_mc._height = h;
    this.drawFrame();
    if (this.focused) {
        super.myOnSetFocus();
    }
    this.initContentPos('fLabel_mc');
};
```

working of each instruction. The specification of the time parameter enhances the testing of a code snippet by measuring the CPU time. It is required for completing a working functionality of defined code instructions. Basically the Elapsed Time is undertaken for optimization purposes. The application size matters a lot, if an auditor is working on large scale applications it is not advisable to measure a test the whole application in one

prompt. The client side optimization depends on a number of factors. These factors are, primarily, the code design and the execution procedure. During execution time a number of resources are undertaken, like RAM, CPU Cycles etc. While testing the FLEX application, the usage of resources is always scrutinized. This helps in understanding the consumption of relative components when an application is executed as a process. The optimization

of an application is necessary to use resources in a well sustained manner. The FLEX application can either run in direct flash player or embedded in browsers. We will go through the optimization procedure based

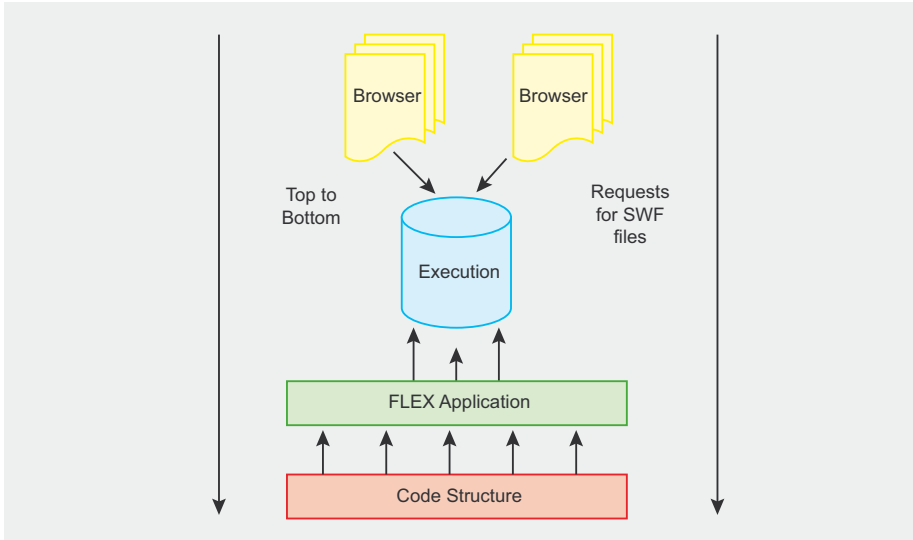


Figure 3. Working flow Model

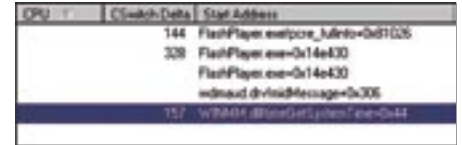


Figure 4. Thread Info of Flashplayer process



Figure 5. Dependency of Flash Player Process

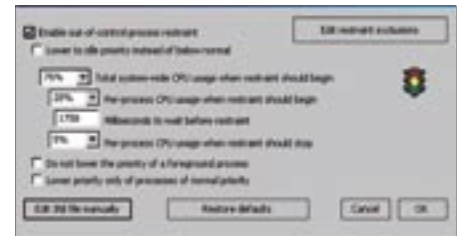


Figure 6. Setting Process Parameters with Process Lasso Tool

Listing 2. Disassembly of a SWF file

```

frame 0
  constants 'component', '_parent', 'arrow', 'arrow_
    mc', 'registerSkinElement', 'face',
    'face_mc', 'shadow', 'shadow_mc',
    'darkshadow', 'darkshadow_mc',
    'highlight', 'highlight_mc',
    'highlight3D', 'highlight3D_mc'
  push 'component', '_parent'
  getVariable
  push '_parent'
  getMember
  varEquals
  push 'arrow', 'arrow_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  callMethod
  pop
  push 'face', 'face_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  callMethod
  pop
  push 'shadow', 'shadow_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  
```

```

  callMethod
  pop
  push 'darkshadow', 'darkshadow_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  callMethod
  pop
  push 'highlight', 'highlight_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  callMethod
  pop
  push 'highlight3D', 'highlight3D_mc'
  getVariable
  push 2, 'component'
  getVariable
  push 'registerSkinElement'
  callMethod
  pop
end // of frame 0
end // of defineMovieClip 104
defineMovieClip 105 // total frames: 1
end // of defineMovieClip 105

defineMovieClip 106 // total frames: 1
end // of defineMovieClip 106

defineMovieClip 107 // total frames: 1
end // of defineMovieClip 107
  
```

on underlined model: We will follow a Top to Bottom approach for testing and auditing FLEX applications (see Figure 3).

We will start up with the process initialization routine when a flash application is loaded into flash player or a browser. A process is created which runs in the memory. Usually if flash player is used then flashplayer.exe is loaded into memory for execution of the FLEX applications. The application is in execution state and thereby consuming resources on the system. A good auditing method of optimization includes variation in working algorithm of applications to test the stringent affects on the operating system. This procedure includes RAM usage, CPU variation etc. Before getting into code intricacies of FLEX applications it is necessary to test the execution behavior of the base processes. In order to work over this factor we will derive some of the basic

testing factors that prove useful in testing optimization of the applications.

The following concepts are to be observed while dynamically checking the FLEX applications. Basically we are doing an assessment of FLASH applications. In this approach and methodology number of parameters is tested to check the application response. This mechanism is followed only for runtime analysis and assessment prior to testing an application's code in raw format. This is crucial for having a peripheral knowledge of structured execution and elapsed time.

PHASE 1: Peripheral Testing of Running Flash Player Process

The Methodology and working concepts: Testing FLEX application

Analyzing Thread Semantics of a Process [running FLEX application]

Whenever a process is created in a system a number of threads are initialized based on the execution behavior of an application. For every single process there must be threads in the system. The analysis of threads running inside a process whenever a FLEX application is executing in a system context provides internal information of objects that are being accessed and created by the application. Understanding of thread execution pattern is one of the

basic elements in assessment of FLEX applications. It gives knowledge of thr various objects of the system used by the process. Figure ??? shows how underlying threads are extracted with execution of a flash application in a flash player. Let's see in Figure 4

Detailed information of running threads can be extracted from this layout. This is necessary because in high end systems, running heavy applications can cause considerable memory usage and memory leaks. This memory leaking is a result of basically an executing thread which has failed to return back. Due to this factor, system resources are getting consumed. If a tester has knowledge of running threads it becomes easy for a tester to scrutinize the inherited threads in a system. Therefore collecting information on threads is the first step for application assessment.

Dependency Checking of a Process [running FLEX application]

A process is composed of number of functions that are called from number of modules. This process is accomplished

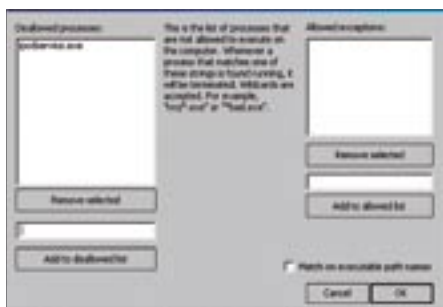


Figure 7. Process Testing Allow / Disallow in Process Lasso Tool

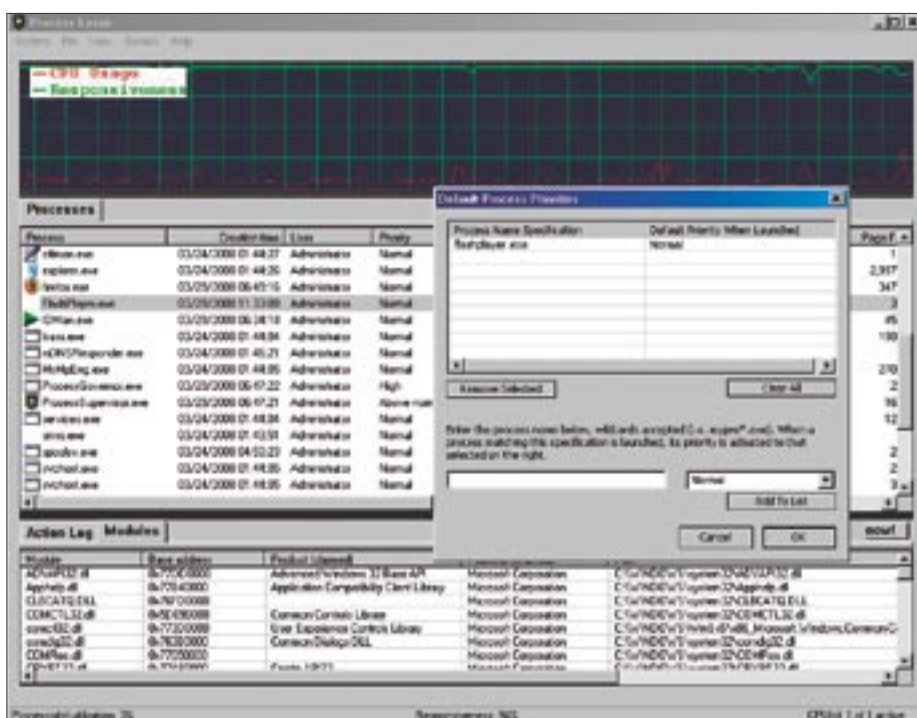


Figure 8. Process Lasso In Action

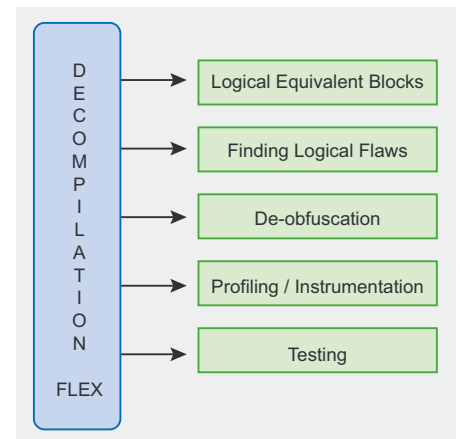


Figure 9. FLEX Decomposition Model

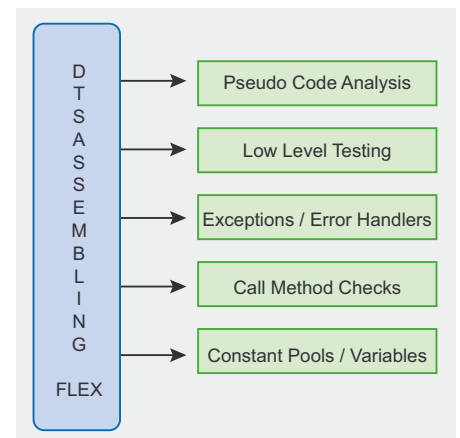


Figure 10. FLEX Disassembling Model

through Dynamic Link Libraries. The flash player requires number of modules to be loaded dynamically at run time. The information of modules is necessary because

it helps a tester in determining the nature of function called and whether that function is loaded successfully or not. This factor affects the CPU state because some applications try

again and again to load required functions from the modules, which in turn affects the robustness of a FLEX application because if an application failed to load a module it might not work properly. Example for flash player listed module is required (see Figure 5). This allows a tester to look into the failed dependencies of a process.

Listing 3. Dumping objects from SWF file

```
[Action Objects]
Running Status: F:\Audit\flash_auditing>swfdump.exe -a nav.swf | more
( 4 bytes) action: Push Lookup:0 ("component") Lookup:1 ("_parent")
( 0 bytes) action: GetVariable
( 2 bytes) action: Push Lookup:1 ("_parent")
( 0 bytes) action: GetMember
( 0 bytes) action: DefineLocal
( 4 bytes) action: Push Lookup:2 ("face") Lookup:3 ("frame5")

[Text Objects]
Running Status: F:\Audit\flash_auditing>swfdump.exe -t nav.swf | more
4 DEFINESPRITE defines id 0096
35 DOACTION
16 PLACEOBJECT2 places id 0089 at depth 0001 name "face_mc"
17 PLACEOBJECT2 places id 0091 at depth 0003 name "arrow_mc"
21 PLACEOBJECT2 places id 0092 at depth 0005 name "highlight_mc"
18 PLACEOBJECT2 places id 0093 at depth 0007 name "shadow_mc"
20 PLACEOBJECT2 places id 0094 at depth 0009 name "darkshadow_mc"
21 PLACEOBJECT2 places id 0095 at depth 0011 name "highlight3D_mc"
0 SHOWFRAME 1 (00:00:00,000)
0 END

[Placement Objects]
Running Status: F:\Audit\flash_auditing>swfdump.exe -p nav.swf | more
11 FRAMELABEL "Symbol_10" has 1 extra bytes (ANCHOR)
6 PLACEOBJECT2 places id 0001 at depth 0001
| Matrix
| 1.000 0.000 0.00
| 0.000 1.000 0.00
0 SHOWFRAME 1 (00:00:00,000) (label "Symbol_10")
0 END
```

Listing 4. Milling XML code from SWF file

```
Running Status: F:\Audit\flash_auditing>swfmill swf2xml nav.swf nav.xml
Output:
<actions>
  <Dictionary>
    <strings>
      <String value="component"/>
      <String value="_parent"/>
      <String value="face"/>
      <String value="frame5"/>
      <String value="registerSkinElement"/>
      <String value="shadow"/>
      <String value="frame3"/>
      <String value="darkshadow"/>
      <String value="frame1"/>
    </strings>
  </Dictionary>
  <PushData>
    <items>
      <StackDictionaryLookup index="0"/>
      <StackDictionaryLookup index="1"/>
    </items>
  </PushData>
  <GetVariable/>
  <PushData>
</actions>
```

Process Parameter Testing

The process is always associated with number of parameters. These parameters are tested with different values to check the process reaction and associated system response. In order to test the process from an initial state while executing, one should go for underlined parameter testing. Let's have a look.

Process Restraining Checks:

For testing an assessment of any application in a system, the process restraining mechanism should be followed. Basically, process restraining is a method of lowering down the priority of a process in a pool so that other processes can consume the resources from CPU if required. This mechanism does not lower down the execution of a process but simultaneously provides an edge to other processes for using the CPU cycles directly. If other processes are using it then the running process can use all cycles. This concept is very useful in analyzing run time stats of malware and also memory exhaustion programs written in FLEX. The system is exploited at the backend continuously. Memory leaking can be tested. By changing the priority of a running FLEX application the system context of a process by looking at the CPU graph. It also shows how well the process is sharing resources with other applications. The properties can be specified as illustrated: see Figure 6.

Another step of testing includes disallowing a process with relation to other processes as: see Figure 7. This stops a declared process from running in memory thereby providing resultant CPU cycles to test process more effectively.

Application Thread Boosting

The working of threads is disseminated into background and foreground threads. For every thread, time slices are provided for proper execution of a thread. In order to test

a thread during execution, Thread boosting is carried out. This process is applied primarily for the foreground threads. For Example: windows by default follow this process of thread boosting. The boosting term refers to a provision of more time slice to a specific thread. This technique is applied in specific situations where background threads are interfering with foreground threads, allowing the foreground thread to be boosted.

Application Page Faults

Page faults occur when a process is trying to load a virtual memory address which has not been loaded or initialized. If a process results in a number of page faults, system and application performance will be degraded. So running FLEX applications should be tested to check the number of page faults caused by the process.

These are the standard techniques to be followed directly. One can tune the performance for speeding up the process and a logging mechanism to test the running FLEX application more effectively.

Note: The techniques can be directly implemented with Process Lasso from BITSUM technologies. A good tool to

implement concept driven testing as mentioned above.

A view of Process Lasso: see Figure 8. This layout presents a peripheral testing phase, analyzing a running FLEX application in a flash player and the various threads related to it. A simple testing layout is presented. Since most of the core files are in SWF format, we will jump into the conversion mechanism to change the compiled form into simple code form for better analysis. The code discrepancies will be checked afterwards. Let's get into second phase:-

Phase 2: Code Conversions of FLEX Applications: - Reversing The Semantics

This phase second involves the conversion mechanism from SWF file format to raw code and pseudo-code for better analysis. This is the static process of analyzing the inbuilt structures of flash applications. This is really necessary from testing and assessment point of view. The reversing of FLEX applications provides a plethora of information that is required for target application analysis. In this process, the main aim of tester is to reverse the

application to the point that information and functionality of code can be checked and cross tested. It includes de-compilation, pseudo code analysis through disassembly and static code analysis. Let's see.

Decompiling FLEX Applications

The process of reverse engineering is a good technique of looking at the source code. The FLEX applications need to be decompiled first for analyzing source code directly. The flash applications are mostly applied in compiled form. To understand the application logic it is necessary to decompile it. It provides information on fundamental applied code and becomes easier for a tester to analyze the application. The purpose is to get a clear representation of a program. The direct use information of functions can be extracted easily, and provides further knowledge of compiled instructions. Another use of this process is the automation, as the tester can design another program to scan the code for requisite insecure functions. This process favors the application code scanning. If a flash file is decompiled into raw code it can be fed to a scan engine for finding vulnerable code snippets (see Figure 9).

These are some of the basic factors for which a de-compilation process is followed. To decompile a FLEX application we simply use a de-compilation tool called as FLARE [http://www.nowrap.de/flare.html]. Running Status: c:\audti\flash_audit> flare <swf file>. The resultant file is produced in a FLR format. The code is decompiled as: in Listing 1.

Pseudo code Analysis: Disassembling FLEX Applications

The disassembly of FLEX applications into pseudo-code i.e. mainly into assembly level layout is an efficient process of testing. The pseudo-code analysis is based on

Listing 5. Extracting Objects

```
[-i] 181 Shapes: ID(s) 1, 3, 6, 9, 31, 33, 35, 39, 41, 43, 45, 47, 52, 54, 57, 64, 66,
    70, 78, 84, 90, 102, 110, 118, 1
5, 127, 129, 131, 134, 137, 142, 145, 150, 155, 171-173, 177, 180-183, 192, 194, 196-
    201, 203-209, 211-213, 223, 225-23
[-i] 149 MovieClips: ID(s) 2, 4, 5, 7, 8, 10-30, 32, 34, 36-38, 40, 42, 44, 46, 48-51,
    53, 55, 56, 58-63, 65, 67-69, 71
[-j] 19 JPEGs: ID(s) 179, 191, 426, 436, 441, 446, 520, 523, 527, 530, 557, 560, 563,
    565, 567, 569, 580, 590, 605
[-F] 11 Fonts: ID(s) 122, 153, 174, 184, 187, 215, 244, 421, 505, 517, 584
[-f] 1 Frame: ID(s) 0
```

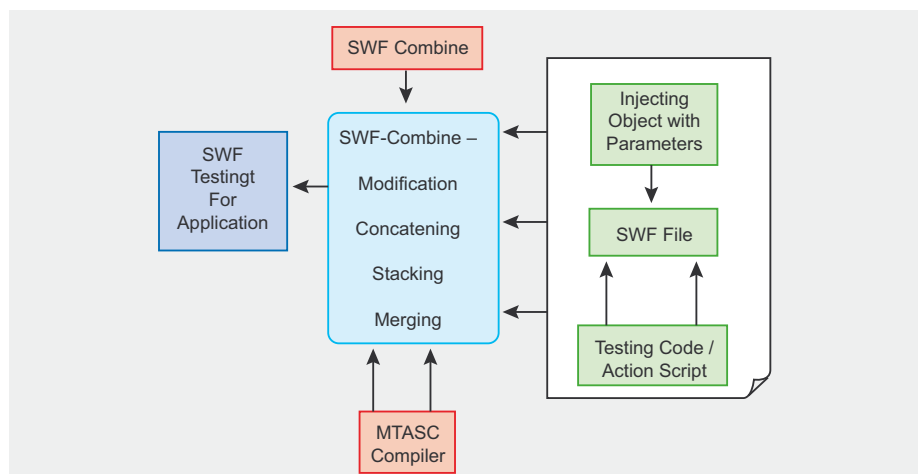


Figure 11. Combining Code in SWF Files Model

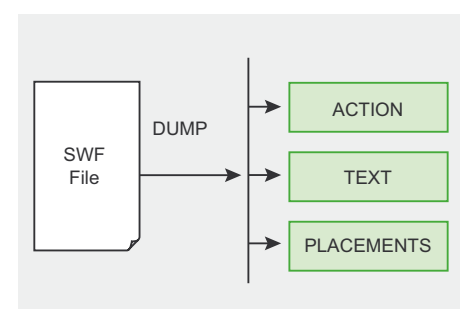


Figure 12. Parameters dumping from SWF files

Logically Dumping of SWF Files:

Due to the large size of SWF files it is necessary to dump the contents logically. The disassembling is an overall process. Sometimes a tester requires dumping SWF files to view certain specific objects in files. The logical dumping provides specific information required to carry out the analysis. The SWF files are basically tested under the following model.

The SWF dump is basically analyzed for three objects. The objects include action, text and placements elements inside the SWF file. SWF-dump tool is used for fetching required objects. Let's look at the the following output: see Listing 3.

Milling XML code from SWF Files

Another possible technique is to change the SWF file into raw XML format. This is process is useful in understanding the hierarchy of tags in XML layout. Moreover it provides greater control to the tester allowing better analysis of SWF file. Due to extensive size and complexity of objects used in SWF file, it becomes critical to traverse through XML format. The conversion mechanism of one format to another is always considered as a reliable operation from a testing point of view. This can be accomplished by using SWF-Mill tool. Let's see in Listing 4.

Extracting Objects from SWF Files

This approach is very useful in extracting a number of objects from SWF files. The objects include different types of pictures like JPEG, Gif, PNG etc. Other objects include sound streams and frame numbers. This technique is reliable when a tester has to analyze a large SWF file. It is not possible to trace along every section of a file looking for desired objects, so the extraction procedure is quite beneficial, as it only extracts specific objects from the file, thereby leaving the file integrity intact.

Running Status: F:\Audit\flash_auditing>swfextract -v nav.xml see in Listing 5.

Application Profiling

Application profiling is a process of testing applications to understand the run-time behavior statistics of various objects running inside FLEX application. The execution is eventually structured as a running process in a system, but in profiling, intermediate

tests are performed to check the working of application. This technique is useful in both optimization and in malware analysis. The time parameter of various instructions is checked to determine the time taken to complete the process. Time is a dependent factor of system involvement, basically the processing time of a single instruction is to be checked (see Figure 13 and Listing 5).

When an application is profiled: see Figure 14.

This approach provides information of the step by step running time of a number of instructions. This is helpful in understanding RAM usage and CPU processing power required when carrying out instructions.

Code scanning of dumped SWF Files

Flex based applications such as flash applications are utilised in byte code format, which means data is passed in a stream of bytes. The browser holds a flash object as an embedded object.

The various web protocols are used for transference of data in stream of bytes from the web server. For this, the bytecode interpreter is required for reading incoming data. For embedding the swf files in browser IFRAME or frame tags are usually used. The security can be implemented through programmatic behavior in which security elements are placed in a code itself. This results in component based security.

Listing 6. FLEX Builder Profiling Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<!-- logging/CheckDebugger.mxml -->
<mx:Script><![CDATA[
import flash.system.Capabilities;

private function checkDebug():String
{
    if (Capabilities.isDebugger) { return "Debugger version of Flash Player!"; }
    else { return "Flash Player!"; }
}

private function checkPrint():String
{
    if (Capabilities.hasPrinting) { return "Printing Supported!"; }
    else { return "Printing Not Supported!"; }
}

private function checkTls():String
{
    if (Capabilities.hasTLS) { return "TLS (Transport Layer Security) Supported!"; }
    else { return "TLS (Transport Layer Security) Not Supported!"; }
}

private function checkLocalf():String
{
    if (Capabilities.localFileReadDisable == true) { return "Local File Read is Disabled!"; }
    else { return "Local File Read is Enabled!"; }
}

]]></mx:Script>
<mx:Text id="info" text="Extracted Flash Player / System Information"/>
<mx:TextArea id="debug_info" text="{checkDebug()}" width="300" height="20"/>
<mx:TextArea id="tls_info" text="{checkTls()}" width="300" height="20"/>
<mx:TextArea id="local_file_info" text="{checkLocalf()}" width="300" height="20"/>
<mx:TextArea id="print_info" text="{checkPrint()}" width="300" height="20"/>
<mx:TextArea id="os_info" text="{Capabilities.os}" width="300" height="20"/>
<mx:TextArea id="man_info" text="{Capabilities.manufacturer}" width="300" height="20"/>
<mx:TextArea id="ver_info" text="{Capabilities.version}" width="300" height="20"/>
<mx:TextArea id="myText" text="" width="300" height="50"/>
<mx:Button id="info_but" label="Server String Lookup" click="{myText.text=Capabilities.serverString}"/>
</mx:WindowedApplication>
```

Our primary aim is to scan the SWF files to find the vulnerable code or insecure code snippets which are being used in the compiled form of the flash application. It is not easy to break into the compiled structure of a binary application to scan the

code for vulnerable objects. The browser is equipped with flash plugin which is used to run flash applications directly from the browser. The parsing is done through plugin and through the LiveConnect interface communication is started.

Listing 7. XXXXXXXXX

```
Yahoo:
<cross-domain-policy>
<allow-access-from domain="*.yahoo.com" secure="false"/>
</cross-domain-policy>

<!-- http://twitter.com/crossdomain.xml -->
-
<cross-domain-policy>
<allow-access-from domain="*.twitter.com"/>
<allow-access-from domain="*.discoveringradiance.com"/>
<allow-access-from domain="*.umusic.com"/>
<allow-access-from domain="*.hippo.com.au"/>
</cross-domain-policy>

-http://api.flickr.com/crossdomain.xml

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

<!-- ws03.search.scd.yahoo.com compressed/chunked Fri Mar 28 00:02:26 PDT 2008 -->
-
<cross-domain-policy>
<allow-access-from domain="*" secure="false"/>
</cross-domain-policy>
-

<!-- http://www.youtube.com/crossdomain.xml -->
-
<cross-domain-policy>
<allow-access-from domain="*.youtube.com"/>
<allow-access-from domain="*.ytimg.com"/>
<allow-access-from domain="*.google.com"/>
</cross-domain-policy>

http://mypodcast.no/crossdomain.xml

<cross-domain-policy>
<allow-access-from domain="www.kingsize.no"/>
</cross-domain-policy>

-http://www.amazon.com/crossdomain.xml

<cross-domain-policy>
<allow-access-from domain="*.amazon.com"/>
<allow-access-from domain="amazon.com"/>
<allow-access-from domain="www.amazon.com"/>
<allow-access-from domain="pre-prod.amazon.com"/>
<allow-access-from domain="devo.amazon.com"/>
<allow-access-from domain="images.amazon.com"/>
<allow-access-from domain="anon.amazon.speedera.net"/>
<allow-access-from domain="*.amazon.ca"/>
<allow-access-from domain="*.amazon.de"/>
<allow-access-from domain="*.amazon.fr"/>
<allow-access-from domain="*.amazon.jp"/>
<allow-access-from domain="*.amazon.co.jp"/>
<allow-access-from domain="*.amazon.uk"/>
<allow-access-from domain="*.amazon.co.uk"/>
</cross-domain-policy>
```

The code scanning encompasses HTML applications too. This is because most of the functionality of flex applications are undertaken by embedding with HTML. The flex applications need to be decompiled first to FLR format [using flare tool]and after de-compilation the code is undertaken in raw format. Once the file is generated it can be audited by scanning to find vulnerable objects or insecure code snippets used in the design of the FLEX application. The model is presented below:

The flex based application needs to be decompiled first. The de-compilation process provides a base for the scanning of applications. The source code can be scanned by matching it with vulnerable code signatures or snippets.

The insecure objects to be scanned as decompiled SWF Files:

- System.security.loadPolicyFile
- Code Parameters to Check
 - Extracting URL: getUrl
 - Load Events: load*(URL,..) Functions, loadVariables(url, level),LoadMovie (url, target),LoadMovieNum (url, level),XML.load (url),LoadVars.load (url),Sound.loadSound(url , isStreaming);NetStream.play(url);
 - Field Setup: TextField.htmlText [Metadata Checks]
 - Conversion Checks: Flash to XML try { __flash__toXML(); } catch(e) { Undefined; }
- try{code}catch(e){location.reload()}}
- Variable Initializations- Global / Local
 - _level
 - _root
 - _global
- System.Security.allowDomain
- Debugging Code Checks in SWF Codes [Trace Parameter] <mx:Script><![CDATA[

```
private function traceEvent(event:
                                Event):void {
trace(event.currentTarget + ":" +
                                event.type);
}
]]></mx:Script>
```

Security Error Handler Checks:

```
private function
    triggerSecurityError():void {
var request:URLRequest =
    new URLRequest
    ("http://www.[yourDomain].com");
loader.load(request);
}
private function
    securityErrorHandler
    (event:SecurityErrorEvent):void {}
```

- Shared Objects Check : asfunction
- viewSourceURL property check in <mx:Application> [Enable / Disable]
- Input validation checks through <mx:validator> class

The scanning of code provides information on the various objects used.

Appendix

Browser Compatibility Checks. Flash player is not supported for playback in 64-bit browsers. It supports extensible playback in 32 bit browsers running on 64 bit operating systems.

One thing that needs to be tested is compatibility of flash player when running inside any browser. It has been noticed that a problem occurs when a DEP is enabled inside the browser. The problem occurs when a specific version of a required plug-in is not running. Usually

the designing of ADD ONS are based on ATL libraries. The ATL stands for Active Template Library and is used for creating COM object through classes in C++. The basic cause is the creation of 64 bit Active X controls.

The DEP is enabled to run processes through an execution protection mechanism there-by reducing memory exploitation. This feature requires a new version of Active X Control with newer version of ATL libraries. Due to this problem, some of the plug-ins such as Adobe flash player are required to be updated. The new add-on versions have been enhanced to work with the DEP enabled feature. From the client side security, this issue is not handled properly. The security of FLEX applications running in flash player in browsers needs to be tested efficiently to avoid unhandled exceptions. The plug-in is required to be designed as per the standard ATL libraries for execution compatibility.

For Example: running the old version of the flash player plug-in, the browser crashes completely. The compatibility check needs to be performed. Things need to look up against this issue:

- Checking the specific version of running browser.
- The plug-in versions that are added as dynamic ADD ONS.

- The Data Execution policy check [DEP] in Internet Explorer.

The compatibility problems can affect the security of running applications client side. For developing Active X controls, the developers should know which ATL libraries are to be used so that no application exceptions occur due to version incompatibility.

Cross Domain Files from different resources (see Listing 7).

Terminology

Profiling: Traversing through Program for Run Time behavioral Checks. Profiling is basically dissected into two parts. The segregation is done on the size of Code Segments analyzed and the interdependency of segments.

Macro-profiling: Performing Run Time checks for complex code segments. This type of profiling basically deals with large software code and the complexity of calls between them.

Micro-profiling: Performing Run Time checks for single line code or short code segments.

Throughput: The number of instructions executed by the processor in per unit time.

Latency: It is described as the time interval required to complete the on production cycle.

The profiling calculates the run-time usage and CPU utilization to query the resultant affect on the system.

- SWF: Small Web Format / ShockWave Flash
- FLA: Proprietary Flash source files used by Adobe Flash IDE
- AS: ActionScript
- Flex: Flash 9/ActionScript 3 IDE with interface libraries
- MXML: XML Interface Markup Language
- AMF: Action Message Format
- SWX: SWF Data Format
- ABC: ActionScript Byte Code
- RIA: Rich Internet Application.

Aditya K Sood, a.k.a. 0kn0ck

Aditya K Sood, a.k.a. 0kn0ck, is an independent security researcher and founder of SecNiche Security, a security research arena. He works for KPMG as a Security Auditor. His research articles have been featured in Usenix Login. He has given advisories to forefront companies. He is an active speaker at conferences such as EuSecWest, XCON, OWASP, and CERT-IN. His other projects include Mlabs, CERA, and TrioSec.

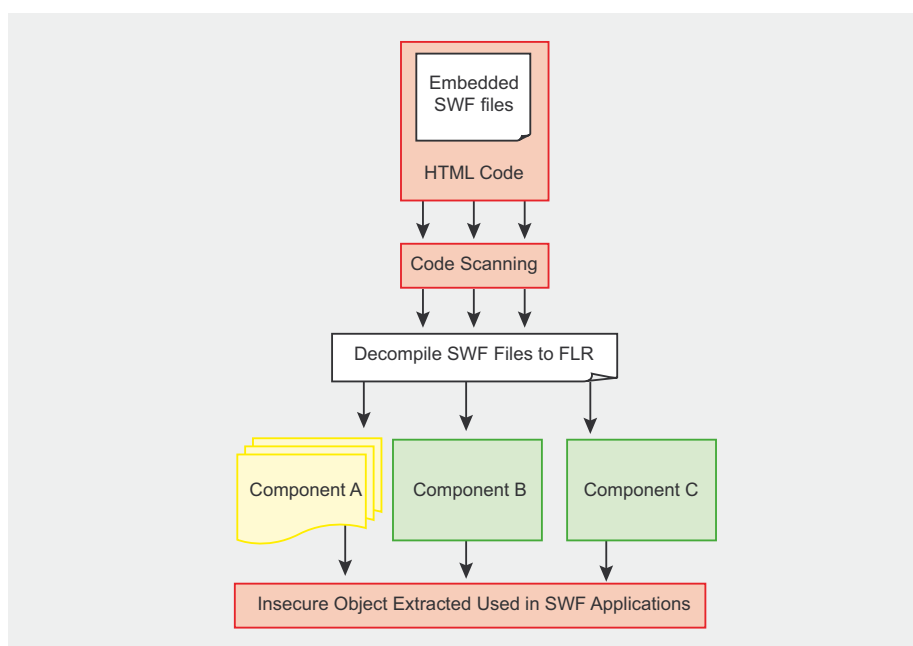


Figure 15. Extracting Components from SWF File – a layout