



Community Home > Software > Enterprise Security Blogs > HP Security Research Blog > Playing with Adobe Flash Player Exploits and Byte Code

Article Options

Subscribe

## HP Security Research Blog

The HP Security Research blog provides a platform for security experts from across HP to discuss innovative research, industry observations, and updates on the threat landscape to help organizations proactively identify and manage risk.

# Playing with Adobe Flash Player Exploits and Byte Code

Matt\_Oh | June 10, 2014 - last edited June 11, 2014

Post a Comment

Adobe Flash Player has been a major target for exploits and malware in recent years. I wrote about [CVE-2014-1776](#) and [CVE-2014-0515](#) exploits just a few weeks ago. CVE-2014-1776 is an IE vulnerability, but the exploit found in the wild was using an Adobe Flash Player file to achieve reliable exploitation against ASLR and DEP. CVE-2014-0515 was a vulnerability with the Adobe Flash Player Pixel Bender component.

Basically, SWF files are not something you can avoid analyzing if you are dealing with real-life exploits. A good methodology when analyzing SWF files is also very beneficial for current malware research. I talked about automating SWF exploits and malware analysis in a previous [presentation](#), but here I want to share a more manual methodology you can use for daily research. All the tools are free and some of them are open source. For this example, I used a sample with a SHA1 value of `300a7e4d54eca8641d7a19ceb4ab68bb76696816`. This sample exploits the CVE-2014-0515 vulnerability.

## Get ready...

Basically, there are many tools, commercial or otherwise, that you can use to analyze SWF files. There are many different decompilers available. One of the best tools I found was [JPEXS Free Flash Decompiler](#) aka *FFDec*. I use it to acquire source code for ActionScript byte code. [Adobe Flex SDK](#) is also a useful tool. It has a very basic SWF dumping tool called *swfdump.exe* and can dump ActionScript byte code, too. The *mxmlc.exe* is a compiler that can compile Flex applications, but you can also use it to compile ActionScript files to a SWF file. The [RABCDasm](#) package is also used here. It is an ActionScript disassembler/assembler package. The following tool sets are provided with this package:

- ablexport.exe: Export ActionScript byte code from a SWF file
- abcreplace.exe: Embed ActionScript byte code to a SWF file
- swfbinexport.exe: Export DefineBinaryData tag from a SWF file
- swfbinreplace.exe: Embed DefineBinaryData tag from a SWF file
- rabcasm.exe: ActionScript byte code assembler
- rabcasm.exe: ActionScript byte code disassembler

The first step when analyzing a SWF exploit is to look at its various components. One SWF file is composed of multiple tags. Figure 1 shows the output from the *swfdumptool*. From this example, the *DefineBinaryData* and *DoABC2* tags look interesting. The *DefineBinaryData* tag contains binary data used in other places and the *DoABC2* tag contains ActionScript byte code.

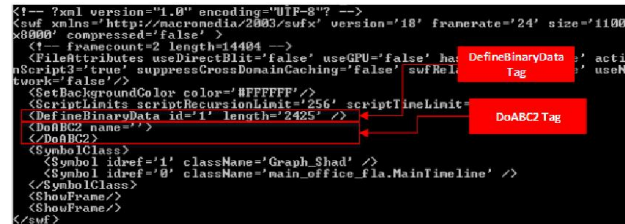


Figure 1 Swfdump result

You especially need to find a way to play with the ActionScript code, as it contains core parts of the exploit in many cases. I suggest two different methodologies here. One is the 'decompile and reconstruct' method and the other is the 'disassemble and reconstruct' method. Choosing the right method depends on how the decompiling technique works against the sample files. The 'decompile and reconstruct' method acquires source code through the decompiler and reconstructs the exploit after modification. The good thing about this method is that you can freely edit the source code and experiment with it as you wish. The bad thing is, in many cases, the decompiler won't work on malicious SWF files. When the decompiler fails, your only remaining option is to use the 'disassemble and reconstruct' approach. I'm going to explain both methodologies here.

## Set...

When debugging SWF samples, it is beneficial to use a debug version of Flash player. You can download the developer versions of Flash Player [here](#). With these developer binaries, you need to setup a Flash Player environment with *mm.cfg*. You can read more detailed instructions [here](#). For Windows systems, you need to place an *mm.cfg* file under the user profile folder and add lines related to *trace* functionality. The *mm.cfg* file I used has the following lines to enable error reporting and *trace* output:

```
ErrorReportingEnable=1
TraceOutputFileEnable=1
```

When a SWF file is run in the debug version of Flash Player, it generates a debug log file in the following location on Windows systems:

```
%PROFILE%\AppData\Roaming\Macromedia\Flash Player\Logs\flashlog.txt
```

## Go! The decompile & reconstruct method

Some Flash-based exploits and malware can be decompiled without any issues, as was the case with our example. Figure 2 shows the screen when I decompiled our sample file. You can export source code files to a folder by pressing the *Export selection* button.

## My Community

[Getting Started](#)

[Join the Conversation](#)

[Register Now](#)

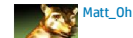
[Already a member?](#)

[Log in](#)

[Quick Links](#)

## Search

## About the Author



Twitter: @ohjeongwook

## Featured



## Follow Us



## Latest Articles

- [Software Development KITchen sink](#)
- [HP Security Research OSINT \(OpenSource Intelligenc...](#)
- [Milestone today, good times ahead](#)
- [HP Security Research OSINT \(OpenSource Intelligenc...](#)
- [Cloud "shiny rocks" and your SOC](#)
- [HP Security Research OSINT \(OpenSource Intelligenc...](#)
- [HP Security Research OSINT \(OpenSource Intelligenc...](#)
- [Analyzing CVE-2015-1635 from cause to cure](#)
- [Crypto Manifesto 2015](#)
- [HP Security Research OSINT \(OpenSource Intelligenc...](#)

## Latest Comments

- [SasiSiddharth on: Analyzing CVE-2015-1635 from cause to cure](#)
- [Percy Rotteveel on: Crypto Manifesto 2015](#)
- [Ivan Sh on: Full details on CVE-2015-096 and the failed MS10-...](#)
- [Sandeep Sagar on: HPSR, Microsoft, disclosure, and the \\$125,000 bug ...](#)
- [Giorgio di Grazia on: POS malware - a look at Dexter and Decebal](#)
- [Nicholas Fagan on: Technical analysis of the SandWorm Vulnerability \(...\)](#)
- [Ajayi Oluwaseun Emmanuel on: Hacking my smart TV - an old new thing](#)
- [alvaro\\_munoz on: Update your Struts 1 ClassLoader manipulation filt...](#)
- [Lotie Brink on: Reverse Engineering NAND Flash Memory - POS device...](#)
- [Philimanjaro on: Hacking POS Terminal for Fun and Non-profit](#)

## Archives

- [May 2015](#)
- [April 2015](#)
- [March 2015](#)
- [February 2015](#)
- [View Complete Archives](#)

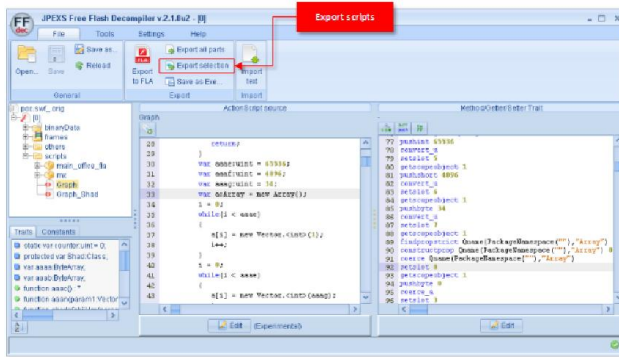


Figure 2 FFDec decompiling

Figure 3 shows the source code that is exported from *FFDec*. You just need to compile it back to a SWF file after modifying the code as you want.

Name	Date modified	Type
main_office fla	6/2/2014 7:51 PM	File folder
mx	6/2/2014 7:51 PM	File folder
Graph.as	6/2/2014 7:51 PM	AS File
Graph_Shad.as	6/2/2014 7:51 PM	AS File

Figure 3 Exported source code

Figure 4 shows good examples when adding *trace* messages to the important parts of the exploit. The additional code dumps out bytes that are passed to a vulnerable object and later dumps out an index of an array element that is corrupted by this operation.

```

trace('Start while loop');
while(1)
{
    shader = new Shader();
    try
    {
        var ba:ByteArray = new this.Shad() as ByteArray;
        trace("Setting bytes:" + ba.length);
        var str:string = dump(ba);
        trace(str);
        shader.bytecode = ba;
    }
    catch(e:*)
    {
        //
    }
    while(1 < vector_count)
    {
        if(a[1].length > 256)
        {
            corrupt_index = 1;
            trace("Found corruption at " + 1 + " length: " + a[1].length);
            break;
        }
        ++;
    }
    if(1 != vector_count)
    {
        if(a[corrupt_index][vector_size + 1] > 0)
        {
            break;
        }
    }
    a.push(new vector.<int>(vector_size));
}
trace('End while loop');

```

Figure 4 Adding trace messages

Adobe provides a free compiler (*mxmcl*) with *Adobe Flex SDK*. The main class of the source code is the *main\_office flaMainTimeline.as* file. The following command line shows how you can compile the whole source code tree. The *-omit-trace-statements=false* option prevents *trace* statements from being removed by optimization.

```

mxmcl -omit-trace-statements=false -static-link-runtime-shared-libraries=true -compiler.source-path=.
main_office flaMainTimeline.as

```

Figure 5 shows the *mxmcl* command result: You might see some warnings and errors. In many cases, you can ignore warnings on no type declarations, but if critical errors happen, you need to figure out how to fix them. Sometimes, the errors mean that the decompilation result itself might be erroneous. In that case, you'd be better off going with the 'disassemble and reconstruct' method.

```

C:\mat\as>mxmcl -omit-trace-statements=false -static-link-runtime-shared-libraries=true -compiler.source-path=. main_office flaMainTimeline.as
Loading configuration file C:\mat\Bin\flex\frameworks\flex-config.xml
C:\mat\as>main_office flaMainTimeline.as(14): col: 18 Warning: variable 'ft' has no type declaration.
    public var ft;

C:\mat\as>main_office flaMainTimeline.as(16): col: 16 Warning: function 'frame1' will be scoped to the default namespace: MainTimeline: internal. It will not be visible outside of this package.
    function frame1() : * {

C:\mat\as>Graph.as(163): col: 18 Warning: var 'counter' will be scoped to the default namespace: Sprite: internal. It will not be visible outside of this package.
    static var counter:uint = 0;

C:\mat\as>Graph.as(167): col: 11 Warning: var 'aaaa' will be scoped to the default namespace: Graph: internal. It will not be visible outside of this package.

```

Figure 5 mxmcl result

Now we have built *main\_office flaMainTimeline.swf*. The following command extracts the *DoABC2* tag from the SWF file and saves it to the *main\_office flaMainTimeline-0.abc* byte code file:

```

abcexport main_office flaMainTimeline.swf

```

The following command embeds the *main\_office flaMainTimeline-0.abc* byte code file back to original *poc.swf\_file*:

```

abcplace.exe poc.swf_0 main_office flaMainTimeline-0.abc

```

Now we have an exploit file that has our own modified version of ActionScript byte code embedded in it. The benefit of this method is that it will not touch other parts of the SWF file. Sometimes malicious data presents in areas other than the *DoABC2* byte code tag. When that happens, working on those tags with SWF-related tools can destroy the original format. In that case the 'decompile and reconstruct' method is useful for preserving the original malicious areas.

## Disassemble & reconstruct

This method is more interesting than the 'decompile and reconstruct' method. Often when you're dealing with SWF files, they may use heavy obfuscation. Other exploits use the glitch in the ActionScript byte code interpreter for exploitation. In that case, the byte code itself is broken in such a way that decompiling is not possible. The decompiler technology is very helpful, but there are too many cases where it can fail. The worst problem is a silent fail. When the decompiler presents you with valid-looking ActionScript code, there is always a chance that the interpretation of the byte code is wrong. There is no good way to verify that the decompiled source code is valid. You should always be careful when you're dealing with malicious SWF files.

The following command exports the ActionScript byte code to a separate file named *poc-0.abc*.

```
abcexport poc.swf_
```

Now you can use a disassembler to disassemble the *poc-0.abc* file. The following command disassembles the *poc-0.abc* file and creates a folder named *poc-0* with ActionScript disassembly files. (Figure 6)

```
rabcdasm.exe poc-0.abc
```

Name	Date modified	Type
main_office fla	6/3/2014 3:01 PM	File folder
mx	6/3/2014 3:01 PM	File folder
Graph.class.asasm	6/4/2014 10:25 AM	ASASM File
Graph.script.asasm	6/3/2014 3:01 PM	ASASM File
Graph_Shad.class.asasm	6/3/2014 3:01 PM	ASASM File
Graph_Shad.script.asasm	6/3/2014 3:01 PM	ASASM File
poc-0.main.asasm	6/3/2014 3:05 PM	ASASM File

Figure 6 ActionScript disassembly files

You can edit these ASASM files and re-assemble them to a SWF file. To work on these files, you need to understand ActionScript byte code instructions. A good reference for the ActionScript virtual machine and byte code can be found [here](#). Figure 7 shows a good example of using the *trace* call to dump out useful information. It resolves the *trace* function first (*findpropstrict*), after pushing a string object (*pushstring*) and calling a *trace* function (*callpropvoid*). This dumps out a message to a log file when the routine is called.

1	Class	
2	refid "Graph"	
3	instance QName(PackageNamespace(""), "Graph")	
4	extends QName(PackageNamespace("Flash.display"), "Sprite")	
5	flag SEALED	
6	flag PROTECTEENS	
7	protectedns ProtectedNamespace("Graph")	
8	init	
9	refid "Graph/instance/init"	
10	flag NEED_ACTIVATION	
11	body	
12	maxstack 6	
13	localcount 4	
14	initstackdepth 10	
15	maxstackdepth 15	
16	code	
17	getlocal0	
18	pushscope	
19		
20	newactivation	
21	dup	
22	setlocal1	
23		
24	pushscope	
25	getscopeobject	1
26	pushundefined	
27	coerce_a	
28	setslot	1
29		
30		
31		
32		
33		

Figure 7 Adding a trace statement

Calling a simple *trace* function may not be enough; in many cases, you might want a more complicated operation. However, writing very complicated logic with byte code instructions is challenging. One method we can use is to write a utility function that can do complex operations in a separate ActionScript file and combine it with existing code. You can just call the utility method and provide data to it for further processing. For example, if we want to dump byte array data in a hex form, we create an *Util.as* file with *DumpByteArray* method. (Figure 8)

```
package
{
    import flash.text.*;
    import flash.utils.*;
    import flash.net.*;

    public class Util
    {
        public function Util() {
        }

        public static function DumpByteArray(buffer:ByteArray):void
        {
            var out:string = fillup("offset", 8, " ") + " 00 01 02 03 04 05 06";
            var offset:int = 0;
            var l:int = buffer.length;
            var row:string = "";
            buffer.position = 0;
            for (var i:int = 0; i < l; i += 16)
            {
                row += fillup(offset.toString(16), toUpperCase(), 8, "0") +
                var n:int = Math.min(16, buffer.length - buffer.position);
                var string:string = "";
                for (var j:int = 0; j < n; ++j)
                {
                    if (j < n)
                    {
                        var value:int = buffer.readUnsignedShort();
                        string += value >= 32 ? String.fromCharCode(value).toString(16).toUpperCase() : "00";
                        offset++;
                    }
                    else
                    {
                        row += " ";
                    }
                }
                row += " " + string + "\n";
            }
            out += row;
            trace(out);
        }
    }
}
```

Figure 8 Util class with DumpByteArray method

Next, we compile the *Util.as* file, export the *DoABC2* tag, and disassemble it to ASASM files using the following command sequences. The folder *Util-0* has multiple ASASM files in it after this. You can just copy them to the original *poc-0* folder. (Figure 9)

```
mxmhc -omit-trace-statements=false -static-link-runtime-shared-libraries=true -compiler.source-path=. Util.as
abcexport Util.swf
rabcdasm Util-0.abc
```

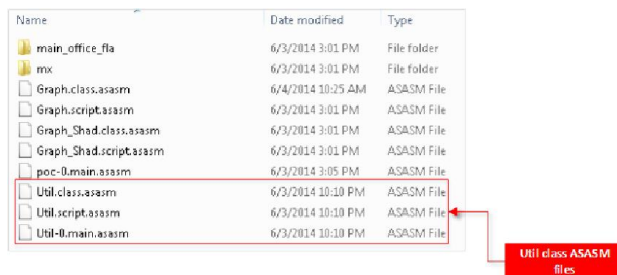


Figure 9 Add Util ASASM files to the existing package

Now you need to include the *Util* class in the main package. You can achieve that by adding a *#include* line to the *poc-0.main.asasm* file. (Figure 10)

```

1      #version 3
2      program
3      minorversion 16
4      majorversion 46
5
6      #include "mx/core/mx_internal.script.asasm"
7      #include "mx/core/IFlexAsset.script.asasm"
8      #include "mx/core/ByteArrayAsset.script.asasm"
9      #include "Graph.Shad.script.asasm"
10     #include "Graph.script.asasm"
11     #include "Util.script.asasm"
12     #include "main_office_fla/MainTimeline.script.asasm"
13
14     end ; program

```

Figure 10 Adding Util.script.asasm to poc-0.main.asasm

After this, you are ready to use the *Util.DumpByteArray* method. Figure 11 shows a good example of how you can call this method from an ASASM file. First, it resolves the *Util* class (*getlex*) and passes a *Graph.Shad* object to the *DumpByteArray* method (*callpropvoid*). The part where it creates the *Graph.Shad* object is copied over from the existing instructions that follow:

```

316      L206:
317      getscopeobject
318      getslot
319
320      #include "Util.class.asasm"
321      #include "Graph.Shad.class.asasm"
322      #include "Flash.utils.class.asasm"
323      #include "DumpByteArray.class.asasm"
324
325      L207:
326      L208:
327      L209:
328      L210:
329      L211:
330      L212:
331      L213:
332      L214:
333      L215:
334      L216:
335      L217:
336      L218:
337      L219:
338      L220:
339      L221:
340      L222:
341      L223:
342      L224:
343      L225:
344      L226:
345      L227:
346      L228:
347      L229:
348      L230:
349      L231:
350      L232:
351      L233:
352      L234:
353      L235:
354      L236:
355      L237:
356      L238:
357      L239:
358      L240:
359      L241:
360      L242:
361      L243:
362      L244:
363      L245:
364      L246:
365      L247:
366      L248:
367      L249:
368      L250:
369      L251:
370      L252:
371      L253:
372      L254:
373      L255:
374      L256:
375      L257:
376      L258:
377      L259:
378      L260:
379      L261:
380      L262:
381      L263:
382      L264:
383      L265:
384      L266:
385      L267:
386      L268:
387      L269:
388      L270:
389      L271:
390      L272:
391      L273:
392      L274:
393      L275:
394      L276:
395      L277:
396      L278:
397      L279:
398      L280:
399      L281:
400      L282:
401      L283:
402      L284:
403      L285:
404      L286:
405      L287:
406      L288:
407      L289:
408      L290:
409      L291:
410      L292:
411      L293:
412      L294:
413      L295:
414      L296:
415      L297:
416      L298:
417      L299:
418      L300:
419      L301:
420      L302:
421      L303:
422      L304:
423      L305:
424      L306:
425      L307:
426      L308:
427      L309:
428      L310:
429      L311:
430      L312:
431      L313:
432      L314:
433      L315:
434      L316:
435      L317:
436      L318:
437      L319:
438      L320:
439      L321:
440      L322:
441      L323:
442      L324:
443      L325:
444      L326:
445      L327:
446      L328:
447      L329:
448      L330:
449      L331:
450      L332:
451      L333:
452      L334:
453      L335:
454      L336:
455      L337:
456      L338:
457      L339:
458      L340:
459      L341:
460      L342:
461      L343:
462      L344:
463      L345:
464      L346:
465      L347:
466      L348:
467      L349:
468      L350:
469      L351:
470      L352:
471      L353:
472      L354:
473      L355:
474      L356:
475      L357:
476      L358:
477      L359:
478      L360:
479      L361:
480      L362:
481      L363:
482      L364:
483      L365:
484      L366:
485      L367:
486      L368:
487      L369:
488      L370:
489      L371:
490      L372:
491      L373:
492      L374:
493      L375:
494      L376:
495      L377:
496      L378:
497      L379:
498      L380:
499      L381:
500      L382:
501      L383:
502      L384:
503      L385:
504      L386:
505      L387:
506      L388:
507      L389:
508      L390:
509      L391:
510      L392:
511      L393:
512      L394:
513      L395:
514      L396:
515      L397:
516      L398:
517      L399:
518      L400:
519      L401:
520      L402:
521      L403:
522      L404:
523      L405:
524      L406:
525      L407:
526      L408:
527      L409:
528      L410:
529      L411:
530      L412:
531      L413:
532      L414:
533      L415:
534      L416:
535      L417:
536      L418:
537      L419:
538      L420:
539      L421:
540      L422:
541      L423:
542      L424:
543      L425:
544      L426:
545      L427:
546      L428:
547      L429:
548      L430:
549      L431:
550      L432:
551      L433:
552      L434:
553      L435:
554      L436:
555      L437:
556      L438:
557      L439:
558      L440:
559      L441:
560      L442:
561      L443:
562      L444:
563      L445:
564      L446:
565      L447:
566      L448:
567      L449:
568      L450:
569      L451:
570      L452:
571      L453:
572      L454:
573      L455:
574      L456:
575      L457:
576      L458:
577      L459:
578      L460:
579      L461:
580      L462:
581      L463:
582      L464:
583      L465:
584      L466:
585      L467:
586      L468:
587      L469:
588      L470:
589      L471:
590      L472:
591      L473:
592      L474:
593      L475:
594      L476:
595      L477:
596      L478:
597      L479:
598      L480:
599      L481:
600      L482:
601      L483:
602      L484:
603      L485:
604      L486:
605      L487:
606      L488:
607      L489:
608      L490:
609      L491:
610      L492:
611      L493:
612      L494:
613      L495:
614      L496:
615      L497:
616      L498:
617      L499:
618      L500:
619      L501:
620      L502:
621      L503:
622      L504:
623      L505:
624      L506:
625      L507:
626      L508:
627      L509:
628      L510:
629      L511:
630      L512:
631      L513:
632      L514:
633      L515:
634      L516:
635      L517:
636      L518:
637      L519:
638      L520:
639      L521:
640      L522:
641      L523:
642      L524:
643      L525:
644      L526:
645      L527:
646      L528:
647      L529:
648      L530:
649      L531:
650      L532:
651      L533:
652      L534:
653      L535:
654      L536:
655      L537:
656      L538:
657      L539:
658      L540:
659      L541:
660      L542:
661      L543:
662      L544:
663      L545:
664      L546:
665      L547:
666      L548:
667      L549:
668      L550:
669      L551:
670      L552:
671      L553:
672      L554:
673      L555:
674      L556:
675      L557:
676      L558:
677      L559:
678      L560:
679      L561:
680      L562:
681      L563:
682      L564:
683      L565:
684      L566:
685      L567:
686      L568:
687      L569:
688      L570:
689      L571:
690      L572:
691      L573:
692      L574:
693      L575:
694      L576:
695      L577:
696      L578:
697      L579:
698      L580:
699      L581:
700      L582:
701      L583:
702      L584:
703      L585:
704      L586:
705      L587:
706      L588:
707      L589:
708      L590:
709      L591:
710      L592:
711      L593:
712      L594:
713      L595:
714      L596:
715      L597:
716      L598:
717      L599:
718      L600:
719      L601:
720      L602:
721      L603:
722      L604:
723      L605:
724      L606:
725      L607:
726      L608:
727      L609:
728      L610:
729      L611:
730      L612:
731      L613:
732      L614:
733      L615:
734      L616:
735      L617:
736      L618:
737      L619:
738      L620:
739      L621:
740      L622:
741      L623:
742      L624:
743      L625:
744      L626:
745      L627:
746      L628:
747      L629:
748      L630:
749      L631:
750      L632:
751      L633:
752      L634:
753      L635:
754      L636:
755      L637:
756      L638:
757      L639:
758      L640:
759      L641:
760      L642:
761      L643:
762      L644:
763      L645:
764      L646:
765      L647:
766      L648:
767      L649:
768      L650:
769      L651:
770      L652:
771      L653:
772      L654:
773      L655:
774      L656:
775      L657:
776      L658:
777      L659:
778      L660:
779      L661:
780      L662:
781      L663:
782      L664:
783      L665:
784      L666:
785      L667:
786      L668:
787      L669:
788      L670:
789      L671:
790      L672:
791      L673:
792      L674:
793      L675:
794      L676:
795      L677:
796      L678:
797      L679:
798      L680:
799      L681:
800      L682:
801      L683:
802      L684:
803      L685:
804      L686:
805      L687:
806      L688:
807      L689:
808      L690:
809      L691:
810      L692:
811      L693:
812      L694:
813      L695:
814      L696:
815      L697:
816      L698:
817      L699:
818      L700:
819      L701:
820      L702:
821      L703:
822      L704:
823      L705:
824      L706:
825      L707:
826      L708:
827      L709:
828      L710:
829      L711:
830      L712:
831      L713:
832      L714:
833      L715:
834      L716:
835      L717:
836      L718:
837      L719:
838      L720:
839      L721:
840      L722:
841      L723:
842      L724:
843      L725:
844      L726:
845      L727:
846      L728:
847      L729:
848      L730:
849      L731:
850      L732:
851      L733:
852      L734:
853      L735:
854      L736:
855      L737:
856      L738:
857      L739:
858      L740:
859      L741:
860      L742:
861      L743:
862      L744:
863      L745:
864      L746:
865      L747:
866      L748:
867      L749:
868      L750:
869      L751:
870      L752:
871      L753:
872      L754:
873      L755:
874      L756:
875      L757:
876      L758:
877      L759:
878      L760:
879      L761:
880      L762:
881      L763:
882      L764:
883      L765:
884      L766:
885      L767:
886      L768:
887      L769:
888      L770:
889      L771:
890      L772:
891      L773:
892      L774:
893      L775:
894      L776:
895      L777:
896      L778:
897      L779:
898      L780:
899      L781:
900      L782:
901      L783:
902      L784:
903      L785:
904      L786:
905      L787:
906      L788:
907      L789:
908      L790:
909      L791:
910      L792:
911      L793:
912      L794:
913      L795:
914      L796:
915      L797:
916      L798:
917      L799:
918      L800:
919      L801:
920      L802:
921      L803:
922      L804:
923      L805:
924      L806:
925      L807:
926      L808:
927      L809:
928      L810:
929      L811:
930      L812:
931      L813:
932      L814:
933      L815:
934      L816:
935      L817:
936      L818:
937      L819:
938      L820:
939      L821:
940      L822:
941      L823:
942      L824:
943      L825:
944      L826:
945      L827:
946      L828:
947      L829:
948      L830:
949      L831:
950      L832:
951      L833:
952      L834:
953      L835:
954      L836:
955      L837:
956      L838:
957      L839:
958      L840:
959      L841:
960      L842:
961      L843:
962      L844:
963      L845:
964      L846:
965      L847:
966      L848:
967      L849:
968      L850:
969      L851:
970      L852:
971      L853:
972      L854:
973      L855:
974      L856:
975      L857:
976      L858:
977      L859:
978      L860:
979      L861:
980      L862:
981      L863:
982      L864:
983      L865:
984      L866:
985      L867:
986      L868:
987      L869:
988      L870:
989      L871:
990      L872:
991      L873:
992      L874:
993      L875:
994      L876:
995      L877:
996      L878:
997      L879:
998      L880:
999      L881:
1000     L882:

```

Figure 11 Adding a call to DumpByteArray method

The following command re-assembles the whole ASASM file tree and generates a *poc-0.main.abcf* file with new byte code in it:

```
rabscasm.exe poc-0\poc-0.main.asasm
```

Now replace the ActionScript byte code of poc.swf\_ with the contents of *poc-0.main.abcf*:

```
abcreplace.exe poc.swf_0 poc-0\poc-0.main.abcf
```

If you run the output SWF file on a debug version of Flash Player, a debug log is written to the flashlog.txt file. (Figure 12)

```

Graph init called
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 A5 01 00 00 00 A4 0B 00 43 72 79 73 74 61 6C 6C W...W..Crystall
00000010 69 7A 65 A0 0C 6E 61 6D 65 73 70 61 63 65 00 43 ize.namespace.C
00000020 72 79 73 74 61 6C 6C 69 7A 65 20 62 79 20 50 65 rystallize by Pe
00000030 74 72 69 20 4C 65 73 6B 69 6E 65 6E 00 A0 0C 76 tri Leskinen. V
00000040 65 6E 64 6F 72 00 00 A0 08 76 65 72 73 69 6F 6E endor... version
00000050 00 01 00 A0 0C 64 65 73 63 72 69 70 74 69 6F 6E ...description
00000060 00 43 72 79 73 74 61 6C 6C 69 7A 65 20 2D 66 69 .Crystallize -fi
00000070 6C 74 65 72 00 A1 01 02 00 00 0C 5F 4F 75 74 43 lter.i.....OutC
00000080 6F 6F 72 64 00 A1 01 01 00 00 02 73 69 7A 65 00 oord.i.....size.
00000090 A2 01 6D 69 6E 56 61 6C 75 65 00 3F 80 00 00 A2 &.minValue.?.&
000000A0 01 6D 61 78 56 61 6C 75 65 00 43 96 00 00 33 03 .maxvalue.c..3.
000000B0 00 C0 01 80 00 00 02 00 B0 40 02 00 10 40 1D 02 .A....'@...@..
000000C0 00 C1 03 00 10 00 30 03 00 F1 02 00 10 00 1D 01 .A....0..h.....
000000D0 00 F3 03 00 1B 00 A2 07 64 65 66 61 75 6C 74 56 .o....&.defaultV
000000E0 61 6C 75 65 00 A1 A0 00 00 00 0B 38 80 00 00 42 alue.A....S..B
000000F0 42 43 43 43 43 44 44 44 44 41 41 41 41 42 42 42 BCCCCDDDDAAAABBB
00000100 42 43 43 43 43 44 44 44 44 41 41 41 41 42 42 42 BCCCCDDDDAAAABBB
00000110 42 43 43 43 43 44 44 44 44 41 41 41 41 42 42 42 BCCCCDDDDAAAABBB
00000120 42 43 43 43 43 44 00 00 00 01 80 00 00 34 00 00 BCCCC4.....4..
00000130 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000140 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000150 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000160 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000170 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000180 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
00000190 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
000001A0 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
000001B0 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
000001C0 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..
000001D0 00 01 80 00 00 34 00 00 00 01 80 00 00 34 00 00 ....4.....4..

```

Figure 12 flashlog.txt content

## Summary

Adobe Flash Player is often used for exploitation and critical vulnerabilities are still found in the wild. It can even be used for defeating ASLR and DEP with other vulnerabilities. Even though many malicious SWF samples can be decompiled without problems, there are some heavily obfuscated or manipulated files that defy decompilation. Disassembling them and re-assembling is a good method to use in these cases. Even though this method requires a good understanding of ActionScript byte code instructions, if it is used in the right way, it is very powerful.

Leave a Comment

We encourage you to share your comments on this post. Comments are moderated and will be reviewed and posted as promptly as possible during regular business hours

To ensure your comment is published, be sure to follow the [Community Guidelines](#).

\*Name

\*Email

Website (optional)

Rich Text | [Preview](#)

Quote

\*Word verification by reCAPTCHA

  
[Get a sound challenge](#) [Help with word verification](#)

Cancel

Post Your Comment

powered by **Lithium**

The opinions expressed above are the personal opinions of the authors, not of HP. By using this site, you accept the [Terms of Use](#) and [Rules of Participation](#).

United States

About HP

- Contact us
- Newsroom
- Investor relations
- Living Progress
- Accessibility
- Events
- HP Labs
- Jobs

Social Media

- Consumer support forum
- Enterprise business community
- Developer community
- Corporate blogs

HP Partners

- Become a partner
- Find a reseller
- PartnerOne

Customer Support

- Power cord replacement
- Download drivers
- Register your product
- HP replacement parts
- Authorized service providers
- Training & certification
- Product recycling