Guide to Cracking EnableDebugger2 Password in SWF/Flash

by devadraco@gmail.com (21 June 2009)

1. Introduction

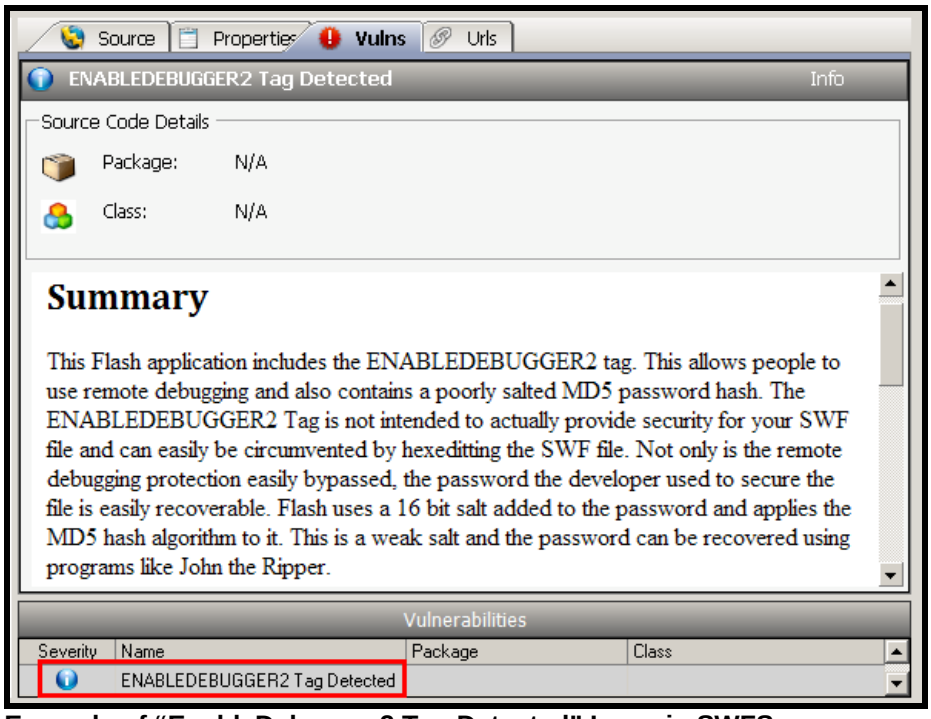
Caveat Emptor!!! Let me first state that I am not an ActionScript programmer. The reason I wrote this article was because I needed to crack an EnableDebugger2 password in a Flash file, but could not find any DIY guide to do it. From the references I found from Google, the details of the hashing scheme used were very sketchy and threw me off the correct trail for hours. This article represents hours of research, experimentation and source code review, and I've provided all the technical details that I feel is important for my peers reading this article.

This article presumes an intermediate level of technical competency.

2. Background

To begin this article, let me first introduce how I came upon the problem. I've recently started using HP's SWFScan (https://h30406.www3.hp.com/campaigns/2009/wwcampaign/1-5TUVE/index.php?key=swf), which is a free (as in beer) Flash security tool from the people who created WebInspect, to perform penetration testing on Flash files.

One of the more interesting findings highlighted by SWFScan is "**ENABLEDEBUGGER2 Tag Detected**". From SWFScan's FAQ (http://www.communities.hp.com/securitysoftware/blogs/spilabs/archive/2009/03/20/hp-swfscan-faq.aspx), this is an issue highlighted by Adobe's best practices.



Example of "EnableDebugger2 Tag Detected" Issue in SWFScan

From SWFScan, we noted that the ENABLEDEBUGGER2 tag is used to allow people to use remote debugging, but the access is controlled by a password. Unfortunately, ENABLEDEBUGGER2 Tag can easily be circumvented (more on it later). Furthermore, the password is stored as a poorly salted MD5 password hash and thus the password used to secure the file is recoverable. Flash uses a 16 bit salt added to the password and applies the MD5 hash algorithm to it. SWFScan states that this is a weak salt and the password can be recovered using programs like John the Ripper. The rest of the article will show you how to do just that.

Why is this important you may ask? It is because some developers have a tendency to reuse passwords, and it is a possibility that the same password might be used in other more sensitive parts of the environment.

3. Sample SWF File Creation

If you already have a SWF file you need to crack, you can skip this section. Else, I will show you how to create a dummy SWF file with the ENABLEDEBUGGER2 tag enabled and password-protected.

You can choose to use any SWF or ActionScript 2/3 compilers, but I'll be using Adobe Flex SDK (http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3) in this article. First, download and extract the SDK. Please note that I am using the DOS syntax as I'm working from a Windows environment. Please change the syntax as needed if you're working in a UNIX environment.

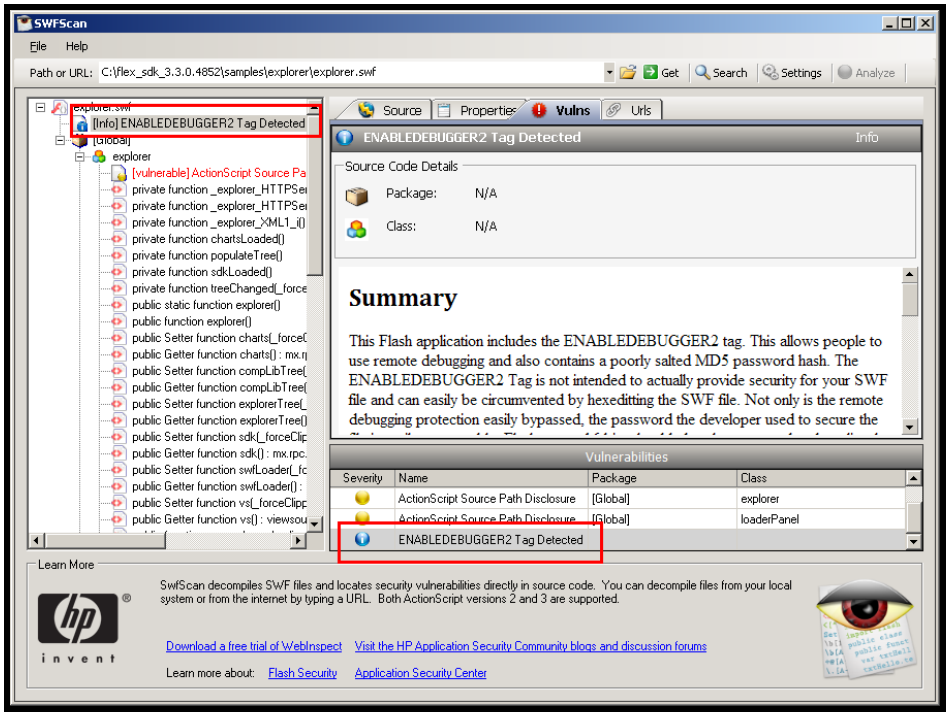
We just need a sample, so we do not need to compile all the scripts in the "sample" directory provided by the SDK. The following command compiles the "explorer" script with "**abc**" as the password.

Run:

```
C:\flex_sdk_3.3.0.4852\samples\explorer>..\..\bin\mxmlc.exe -debug -debug-password
abc explorer.mxml

Loading configuration file C:\ flex_sdk_3.3.0.4852\frameworks\flex-config.xml
C:\flex_sdk_3.3.0.4852\samples\explorer\explorer.swf (601267 bytes)
```

When we check the compiled "explorer.swf" file with SWFScan, we can see that the "ENABLEDEBUGGER2 Tag Detected" issue is present.



Screenshot of SWFScan's Analysis of "explorer.swf"

4. Password Hash

To get the password hash, I used “flasm” (http://www.nowrap.de/flasm.html), a free command line assembler/disassembler of Flash ActionScript bytecode. The source code is also available, which is of great help when I was researching the SWF file structure.

Download and extract “flasm”. Copy the “explorer.swf” file to flasm’s directory.

Run:

```
C:\ flasm16win>flasm.exe -d explorer.swf

movie 'explorer.swf' compressed // flash 9, total frames: 2, frame rate: 24 fps,
 500x375 px

  fileAttributes attrUseNetwork,attrActionScript3,attrHasMetadata

  metadata '<rdf:RDF xmlns:rdf=\'http://www.w3.org/1999/02/22-rdf-syntax-ns#\'><
rdf:Description rdf:about=\'\' xmlns:dc=\'http://purl.org/dc/elements/1.1\'><dc:
format>application/x-shockwave-flash</dc:format><dc:title>Adobe Flex 3 Applicati
on</dc:title><dc:description>http://www.adobe.com/products/flex</dc:description>
<dc:publisher>unknown</dc:publisher><dc:creator>unknown</dc:creator><dc:language
>EN</dc:language><dc:date>Jun 21, 2009</dc:date></rdf:Description></rdf:RDF>'

  enableDebugger2 '$1$ZH$B14iwyCzzcXcqLaJz0Mif0'

  // unknown tag 63 length 16

  scriptLimits recursion 1000 timeout 60

  // unknown tag 82 length 207466

  // unknown tag 76 length 40

  defineMovieClip 3 // total frames: 1
  end // of defineMovieClip 3

...
<The rest of the output are not included>
```

We note that the password hash of EnableDebugger2 tag is “**1ZH$B14iwyCzzcXcqLaJz0Mif0**”.

5. SWF File Format

Before we get to the password cracking, let me summarize some details about the SWF file format. You can get more details from the homepage of “flasm” (http://www.nowrap.de/flasm.html), and from Adobe’s “SWF File Format Specification Version 9” (http://www.adobe.com/devnet/swf/pdf/swf_file_format_spec_v9.pdf).

The SWF file format starts with a header. After the header is a series of tagged data blocks:

```
[Header] [FileAttributes tag] [Tag] [Tag] [Tag] ... [End tag]
```

The great thing about the tags are that all tags share a common format, so any program parsing a SWF file can skip over blocks it does not understand. This ability enables tags to be removed, inserted, or modified by tools that process a SWF file. This is important as it allows tags to be changed using hex-editors or other external programs.

We will look at the “protect”, “enableDebugger” and “enableDebugger2” tags next. The homepage of “flasm” provides an excellent introduction to these tags and the appropriate portion is reproduced in Appendix A.

The “protect” tag is used as a hint for authoring programs, to prevent the SWF file from loading for editing. A password can be specified starting from SWF 5, but is optional. The “protect” tag is not a security feature as it requires the other programs loading the SWF file to respect this tag.

The “enableDebugger” tag introduced in SWF 5 and deprecated in SWF 6, is another attempt to secure the content of SWF. It is always protected by password and this tag allows remote debugging of the SWF file. Without the password, debugger will not access the SWF file. The same will occur if the password is deleted.

But if you disassemble the SWF file, change “enableDebugger” parameter to '1.e$7cXTDev5MooPv3voVnOMX1', and reassemble the file; empty password will be accepted. You can do this using “flasm”. Another way to do this is to change the binaries of the SWF file directly with a hex-editing type of tool if the tool can read the tag blocks correctly.

The “enableDebugger2” tag was introduced in SWF 6 as Flash MX allows debugging on source code level. It is also meant as a replacement to the “enableDebugger” tag, which is ignored in SWF 6 and later.

If you are going to study the source code of “flasm” or other SWF tools, you need to take note of the “Tag type” in the “Header” field of the “Tag” data blocks in the SWF file.

For example, in “action.h” of “flasm”, we note the following:

```
TAG_PROTECT             = 24, /* the author doesn't want the file to be opened */
TAG_ENABLEDEBUGGER      = 58,
TAG_ENABLEDEBUGGER2     = 64,
```

The “enableDebugger2” tag is represented by the tag number 64.

As we mentioned previously, all tags share a common format, so any program parsing a SWF file can skip over blocks it does not understand. For example in the previous section, the output of the disassembled “explore.swf” file contains:

```
// unknown tag 82 length 207466
// unknown tag 76 length 40
```

These are the tag blocks that “flasm” do not understand.

Therefore, if you use other tools or is writing your own tool, tag number 64 is the “enableDebugger2” tag.

6. Salted MD5 Hash

From the previous sections, we noted that the password hash of “abc” is “1ZH$B14iwyCzzcXcqLaJz0Mif0”. You might notice that the hash is shorter than raw MD5 hashes (for example, as used by the “famous” Wordpress web application).

A typical MD5 hash is typically expressed as a 32 digit hexadecimal number, but is unsuitable to be used to store password hashes as the MD5 hashes can be pre-computed. A better method is to use salting: where instead of just hashing the password, the password is used as a key to hash the “salt” value.

The MD5 password encryption algorithm used by Flash was written by Poul-Henning Kamp. The algorithm is freely distributable, and also resides in the FreeBSD tree at “src/lib/libcrypt/crypt-md5.c”. The format of the hash is a 28-characters string in the style of the traditional UNIX “crypt()” password encryption function:

$<ID>$<salt>$<pwd>

For “1ZH$B14iwyCzzcXcqLaJz0Mif0”, the 1 at the start signify the hash method; it is the traditional way to indicate a crypt-MD5 password. Everything between the second and third $ is the salt (“ZH”), and everything after the last $ is the hashed password (“B14iwyCzzcXcqLaJz0Mif0”). The character set used in "<salt>" and "<pwd>" are drawn from the regex set “[a-zA-Z0-9./]”.

Please note that there are many different salted MD5 password schemes. This implementation may be known as “FreeBSD” or “Crypt-MD5”. This is important when choosing a password cracker.

To confirm the MD5 scheme, I use the “Crypt::PasswdMD5” module from Perl. Install the Perl module using CPAN:

```
$ cpan –i Crypt::PasswdMD5
```

Please note that I am using the UNIX syntax from this point on as it is more convenient for me to use Perl and John the Ripper in Linux.

I’ve written the following script “testcrypt.pl” to test my hypothesis:

```
#!/bin/perl

use Crypt::PasswdMD5;

my $cryptedpassword = unix_md5_crypt('abc', 'ZH');
print $cryptedpassword . "\n";
```

Run:

```
$ ./testcrypt.pl
$1$ZH$B14iwyCzzcXcqLaJz0Mif0
```

As you can see, the hash is exactly the same as the “enableDebugger2” tag hash we got from “flasm”.

7. Cracking the Hash

The password cracker I'm using in this article is John the Ripper ("http://www.openwall.com/john/") or JtR. JtR is a fast password cracker that supports "Crypt-MD5". I will not go into the nuances of JtR as I'm only using a short password in this example, but you need to explore the features of JtR more if you intend to crack a real password.

From the "crypt()" manpage, we note that:

> In SHA implementation the entire key is significant here (instead of only the first 8 bytes in MD5).

This means that the maximum key-length is 8 characters, and this is a parameter that is very important in password cracking.

To use JtR, we need to put the hash into a Linux-style unshadowed "passwd" file format:

crack.password.db:

```
devadraco:$1$ZH$B14iwyCzzcXcqLaJz0Mif0:500:500:devadraco:/home/devadraco:/bin/bash
```

Run:

```
$ john crack.password.db
Loaded 1 password hash (FreeBSD MD5 [32/32])
abc              (devadraco)
guesses: 1  time: 0:00:00:01 100% (2)  c/s: 814  trying: abc
```

```
$ /usr/bin/john --show crack.password.db
devadraco:abc:500:500:devadraco:/home/devadraco:/bin/bash

1 password hash cracked, 0 left
```

There we have it! We recovered the password "**abc**". Please note that brute-forcing an 8 character password is not easy and using dictionary-attacks mean a lot more tweaking of the JtR options.

<End>

Appendix A : Extract from “Flasm” Homepage

protect, enableDebugger and enableDebugger2 tags

“protect” was meant by Macromedia as a hint for authoring program, saying that the author of particular SWF doesn't wish it to be opened in Flash IDE. “protect” is not actually protecting anything, any program that deals with SWF can simply ignore it. In Flasm, “protect” will be shown, and can be added/deleted. You can place it anywhere in SWF, albeit usual location is somewhere near to the beginning. Note “protect” is not an action, so it has to be outside of action blocks. Passwords are encoded by Flash compiler into a 28 characters long string, consisting of these parts (Paul Cannon):

> The 1 at the start does signify an encryption scheme; it's the traditional way to indicate a crypt-MD5 password. Everything between the second and third $ is the salt, and everything after the last $ is the hashed password, in a sort of base-64 representation.

Flasm will show the encoded string, but not the password.

“enableDebugger” is another attempt to secure the content of SWF. Always protected by password (Flasm will show the encoded string), this tag gives you the ability to “remote debug” the SWF. If you don't know the password, debugger will not let you in. If you delete the password, debugger will not let you in. But if you change “enableDebugger” parameter to '1.e$7cXTDev5MooPv3voVnOMX1', empty password will be accepted.

To say it clear one more time: above tags, including encrypted passwords, give you no protection and can be safely deleted or altered.

Flash MX allows debugging on source code level, so there is a new tag “enableDebugger2”, which is used instead of “enableDebugger”. It makes no difference at all. However, Flasm will not show another tag (63) or contents of external file used by debugger, don't know anything about their format.