Format string attack

From OWASP

This is an Attack. To view all attacks, please see the Attack Category page.

Last revision: 04/16/2015

Description

The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

To understand the attack, it’s necessary to understand the components that constitute it.

•The Format Function is an ANSI C conversion function, like printf, fprintf, which converts a primitive variable of the programming language into a human-readable string representation.

•The Format String is the argument of the Format Function and is an ASCII Z string which contains text and format parameters, like: printf ("The magic number is: %d\n", 1911);

•The Format String Parameter, like %x %s defines the type of conversion of the format function.

The attack could be executed when the application doesn’t properly validate the submitted input. In this case, if a Format String parameter, like %x, is inserted into the posted data, the string is parsed by the Format Function, and the conversion specified in the parameters is executed. However, the Format Function is expecting more arguments as input, and if these arguments are not supplied, the function could read or write the stack.

In this way, it is possible to define a well-crafted input that could change the behavior of the format function, permitting the attacker to cause denial of service or to execute arbitrary commands.

If the application uses Format Functions in the source-code, which is able to interpret formatting characters, the attacker could explore the vulnerability by inserting formatting characters in a form of the website. For example, if the printf function is used to print the username inserted in some fields of the page, the website could be vulnerable to this kind of attack, as showed below:

```
printf (userName);
```

Following are some examples of Format Functions, which if not treated, can expose the application to the Format String Attack.

Table 1. Format Functions

Format function	Description
fprint	Writes the printf to a file
printf	Output a formatted string
sprintf	Prints into a string
snprintf	Prints into a string checking the length
vfprintf	Prints the a va_arg structure to a file
vprintf	Prints the va_arg structure to stdout
vsprintf	Prints the va_arg to a string
vsnprintf	Prints the va_arg to a string checking the length

Below are some format parameters which can be used and their consequences:

- "%x" Read data from the stack
- "%s" Read character strings from the process' memory
- "%n" Write an integer to locations in the process' memory

To discover whether the application is vulnerable to this type of attack, it's necessary to verify if the format function accepts and parses the format string parameters shown in table 2.

Table 2. Common parameters used in a Format String Attack.

Parameters	Output	Passed as
%%	% character (literal)	Reference
%p	External representation of a pointer to void	Reference
%d	Decimal	Value
%c	Character	
%u	Unsigned decimal	Value
%x	Hexadecimal	Value
%s	String	Reference
%n	Writes the number of characters into a pointer	Reference

Risk Factors

TBD

Examples

Example1

This example demonstrates how the application can behave when the format function does not receive the necessary treatments for validation in the input of format string.

First is the application operating with normal behavior and normal inputs, then, the application operating when the attacker inputs the format string and the resulting behavior.

Below is the source-code used for the example.

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

int main (int argc, char **argv)
{
        char buf [100];
        int x = 1 ;
```

```
        snprintf ( buf, sizeof buf, argv [1] ) ;
        buf [ sizeof buf -1 ] = 0;
        printf ( "Buffer size is: (%d) \nData input: %s \n" , strlen (buf) , buf ) ;
        printf ( "X equals: %d/ in hex: %#x\nMemory address for x: (%p) \n" , x, x, &x) ;
        return 0 ;
}
```

Next is the output that the program supplies when running with expected inputs. In this case the program received the string "Bob" as input and returned it in the output.

```
./formattest "Bob"
```

```
Buffer size is (3)
Data input : Bob
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

Now the format string vulnerability will be explored. If the format string parameter "%x %x" is inserted in the input string, when the format function parses the argument, the output will display the name Bob, but instead of showing the %x string, the application will show the contents of a memory address.

```
./formattest "Bob %x %x"
```

```
Buffer size is (14)
Data input : Bob bffff 8740
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

The inputs Bob and the format strings parameters will be attributed to the variable buf inside the code which should take the place of the %s in the Data input. So now the printf argument looks like:

```
printf ( "Buffer size is: (%d) \n Data input: Bob %x %x \n" , strlen (buf) , buf ) ;
```

When the application prints the results, the format function will interpret the format string inputs, showing the content of a memory address.

Example 2

Denial of Service

In this case, when an invalid memory address is requested, normally the program is terminated.

```
printf (userName);
```

The attacker could insert a sequence of format strings, making the program show the memory address where a lot of other data are stored, then, the attacker increases the possibility that the program will read an illegal address, crashing the program and causing its non-availability.

```
printf (%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s);
```

Related Threat Agents

- contractors
- internal software developer

Related Attacks

- Code Injection

Related Vulnerabilities

- Buffer Overflow

Related Controls

- Category:Input Validation

References

- http://www.webappsec.org/projects/threat/classes/format_string_attack.shtml
- http://en.wikipedia.org/wiki/Format_string_attack
- http://seclists.org/bugtraq/2005/Dec/0030.html