Coalesce Model to Prevent Format String Attacks

Jayant Shekhar[1], Seema Yadav[2], Khaleel Ahmad[3],

CSE/IT Dept. S.I.T.E., SVSU,
Meerut-250002, India

Abstract

Format string attacks cause serious security problems in the field of software security in computer world. Testing and implementation against to Format String vulnerabilities can avoid result due to working of Format String Bugs. In this paper, we have proposed eight novel approaches to prevent format string attacks and combination of these approaches named as Coalesce Model. With the help of this model we check our coding in such a manner that no any type of format string attacks occurs. We check the system implementation of any applications with the help of coalesce model against corruption of application states, and giving the control to attacker. Our work addresses Format String vulnerabilities related to ANSI C library. The result indicates that the proposed model is effective to testing of Format String Vulnerability.

Keyword: *Format function, Software security, Format string attacks, Vulnerability.*

1. Introduction

Several types of computer security attacks arise from software bugs. Format string attacks are most important type of buffer overflow attacks in software security. Format string attacks damage our important information or data first time in year 2000 in computer world. Format string attacks are also called Variadic functions. Format string attacks are result of flexible features of C language by representation of data or information and use of pointer. Flexible feature of C language give more choices to programmer for system programming but C language lacks safety and function arguments checking in the function. Format string function attacks apply to all format string functions in the C library [3][8][16][19] such as given in table1.This is possible to declare functions that take variable number of pointers in C and C++ [3][8][9][12][14]. Format string function are also called conversion function, which is used to convert primitive data types into a human understandable form [8][12][18][22]. Format string function is used in any type of C program to giving output message, printing error message etc. Format string bugs arise to the reason of passing arguments in not a safe way. Format string attacks can be used locally or remotely [3][8][12][14][20][22]. Remote format string attacks are wu-ftpd, BSd ftpd, rpc.stat and local format string attacks are lpr, LPRng, BSD and fstat [8]. Wu-ftpd attacks one of the most widely used on FTP server in computer network security. All attacks program we find out from the Internet, overwrite the function pointers or return address to execute a remote root shell. Format string is also called ASCIIZ or ASCIZ (used to refer a null-terminated ASCII string).

Code for understand the format string problem

```
Wrong use of printf function
Int func (char *employee)
{
printf (employee);
}          // this is the unsafe function
```

Correct form of printf function

```
int func (char *employee)
{
printf ("%s", employee);
}      // this printf function gives the safe function
```

If the attacker control the format string arguments of a Format Function in a victim application, it is very easy for attacker to read or write the application address space.

Format string bugs are very dangerous in computer and software security. It gives full control to attacker of the application to damage our system security. Format string attacks can be used to execute harmful or crashing the program [3][6][14]. Format string attacks arises from the same dark corner as many other security holes, means it comes by laziness of programming. In format string function there is no any single function in the ANSI C definition [3[6][8][9][12][14][22], there are some basic format string functions (family member) [1][6][8][12][14][22][18]which is given below in table 1 and format function specifier of the format function are given in table 2.

Table1. Format String Function

S. N.	Format Function	Output of the Format Function	Pass as
1	fprintf	Prints output to a File stream	Through copying
2	printf	Prints output to the stdout stream	Through copying
3	sprintf	Prints output into a String	Through copying
4	snprintf	Prints output into a String with length checking	Through copying
5	vfprintf	Prints output to a File stream from a va arg structure	Through pointer
6	vprintf	Prints output to the stdout stream from a va arg structure	Through pointer
7	vsprintf	Prints output into a String from a va arg structure	Through pointer
8	vsnprintf	Prints output into a String with length checking from a va arg structure	Through pointer
9	syslog	Output to the log file	Through copying
10	vsyslog	Output to the log file from a va arg structure	Through pointer
11	err	Output as a error	Through copying

Table2. Specifier of The Function

Parameter	Output	Passed as to function
%d	decimal (int)	value
%x	hexadecimal (unsigned int)	value
%s	string ((const) (unsigned) char *)	reference
%n	number of bytes written so far, (* int)	reference
%u	unsigned decimal (unsigned int)	value
%%	% character(literal)	reference
%p	External representation of pointer to void	reference

We take a printf Format Function to explain how the format string looks in a stack. See figure 1.

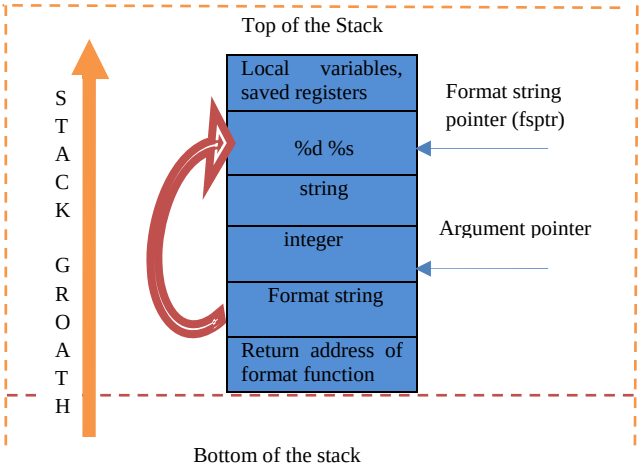

Figure 1.Stack of printf((“%d %s,” integer ,string)

In this example (figure 1) the format string takes two specifier (%x and %s) and two arguments (string and integer) which is corresponds by these specifiers (which is integer and string type). Return address which is given by the function is saved followed by the address of format string and arguments of the function. There are two pointers used to keep track of format string and arguments. The first position of argument pointer is comes after the address of format string.

Function of the format string functions [4][6][8][19][21]

•convert simple C data types to a string.
• allow specifying the format of the representation
• process the resulting (system messaging functions) string (output to stderr, stdout, syslog, etc).

Working of the format string functions [12][14][19][21]

• It controls the behavior of the function.
• It specifies the type of parameters that should be printed as a output.
• Parameters of the function are saved on the stack (pushed).

2. Literature Survey

Timothy Tsai and Navjot Singh [3] developed a tool Libsafe that is a shared library tool and used to prevent Format String Attacks during runtime of the function. The library intercepts format function call and check that function safely executed or not. If a function call does not overwrite the return address of the function with %n specifiers, then it is considered as a safe execution. Otherwise, a warning message is logged and the process is terminated or aborted. This tool is not effective for many other types of attack that do not overwrite the return addresses of the function (e.g., arbitrary reading from stack).

Crispin Cowan and Greg Kroah-Hartman [10] developed a Format Guard tool that is more effective to stop the format string bugs and does not impose compatibility problems to functions. FormatGuard is incorporated into WireX’s Immunix distribution in Linux and server products. FormatGuard tool is used to prevent format string bugs during the compilation and linking stages of the format function. This tools counts the number of arguments which is passed during compile time and matches this count to number of the specifiers of the format string function during program runtime. If there is a mismatch occurred then a warning about format string attacks is logged and the format function call is aborted.

Dekok [17] designed a PScan tool to detect format string attacks to printf family functions. They give two principles of detecting Format string attacks which are: (i) a format string is not constant and (ii) it is the last argument of a function call. This tool not works for vsprintf family of format functions.

Li, W. and Chiueh, T.[18], developed Lisbon tool that is used to protect applications against Format String attacks to insert a canary word at the end of arguments of the format function; it can really stop known format string attacks to run time of the applications.

Robbins, T [22] developed Lib Format tool that is used to detect Format string bugs during runtime. Libformat tool kill the applications if format strings are in writable memory and contain %n specifiers. It can't prevent other attacks related to reading arbitrary memory (e.g., supplying more number of specifiers than the arguments).

3. Novel Approaches Preventing for Format String Attacks

3.1-checking the size of string before using syslog() functions (CBOSYF)

This is our first approach to preventing format string attacks. The syslog**()** function writes message to the system message logger, the message is then written to the system console, log files, logged-in users because syslog() take string as input, but depending on the implementation it does not check the boundary of string before using it. Because of this reason when we use syslog() function, first check the string which is passed to this function, otherwise we invite the FSB.

3.2- Fix the boundary to sprint() function (FBTSpF)

When we do not check the boundary of the sprintf function, there is more possibilities to format string attacks. So prevention from format string attacks we first check the boundary of this function.e.g:

Sprint(buffer,errorcommon:%100",user) not as a
Sprint (output,buffer)

3.3- Check that the number of argument is equal to specifier of function (CArgETS)

When we pass arguments to any format function then first check that number of arguments is equal to specifier or not, if not then there is more possibility to format string attack. Such as sprintf ("%n%d%n", a, b, c) not as a sprintf("%n", a ,b, c)

3.4-use %.nd in the place of %nd (%.nd)

In this approach we clarify that (.n) is the precision that specifies the minimum number of digits to be written after the decimal point, .n is the precision number treated as .n and %d in function but %nd treated as %n and %d specifier, so it give two specifier for one arguments, it gives invitation to format string attacks. For example when we take %.2d, it means 2 numbers of digits to be printed after the decimal point. So because of this we always use (%.nd in the place of %nd) carefully to preventing format string, otherwise we invite the Format String Bugs.

3.5-Avoid to fix the length of format string function (AFLOFoSF)

Not specify any type of length to function such as h, l or L because it takes one type of data types which is declared in that function. Detailed is given below

- h- It interpreted the arguments as a short int or un-signed short int (only applies to integer specifiers: i, d, o, u, x and X).
- l- The argument is interpreted as a long int or unsigned long int for integer specifiers (i, d, o, u, x and X), and as a wide character or wide character string for specifiers c and s.
- L- The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g and G).

3.6-Not use format string function without specifier (NUFWS)

This situation occurs when a programmer writes the programs like this printf (str) as a short-hand for

printf ("%s", str). It is easier to type;

Unfortunately, it is also vulnerable if the attacker inserts spurious % directives in the str string. So as to remove this type of problem first we check that arguments of the function take specifier or not. If checking is true then execute otherwise aborted this type of application.

3.7-Not use conversion specifier without arguments in format string functions (NUSWArg)

To understand this problem we take an example such as printf ("%08x.%08x.%08x\n");
This works, because we instruct the printf function to retrieve three parameters from the stack and display them as 8-digit padded hexadecimal numbers. So a possible output may look like: 40012980.080628c4.bffff7a4.
These values comes from the stack memory by the partial dump, starting from the current bottom upward to the top of the stack assuming that the stack grows towards the low addresses. Depending on the size of the format string buffer and the size of the output buffer, we can reconstruct more or less large parts of the stack memory by using this type of technique. In some cases you can also retrieve the entire stack memory. A stack dump gives most important data or secures information about the program flow and local function variables and may be very helpful for finding the correct offsets for a successful exploitation. So when we use format function then first check that specifier of the function takes arguments or not. If yes then send application to execution otherwise reject the application.

3.8-Avoid using of % n features (AU%nFer)

In this approach we clarify that n is the function argument treated as a pointer to an integer (or integer variant such as a short). The number of output characters is stored in the addresses pointed by the arguments of the function. So, if we specify %n specifier in the format string, then the number of output characters is written to the location specified by the arguments of the function.

The result of spurious %n specifier in printf() format strings is that the attacker can "walk" back up the stack some number of words by inserting some number of %d directives, until they reach a suitable word on the stack and treating that word such as integer in format string. This directive is the most dangerous in format string because it induces printf to write data back to the argument list. So in our novel approaches, we suggest that to avoid the use of %n directive in format string functions.

4 - Coalesce Model

In Coalesce model we combine all eight novel approaches which is given in section III to prevent from Format String Attacks in a secure way. This model filters all the attacks which are related to Format String Attacks that means all the vulnerable applications are rejected or aborted and executed only safe functions. So we can say that this model (see in figure 2) is better to prevent Format String Attacks.

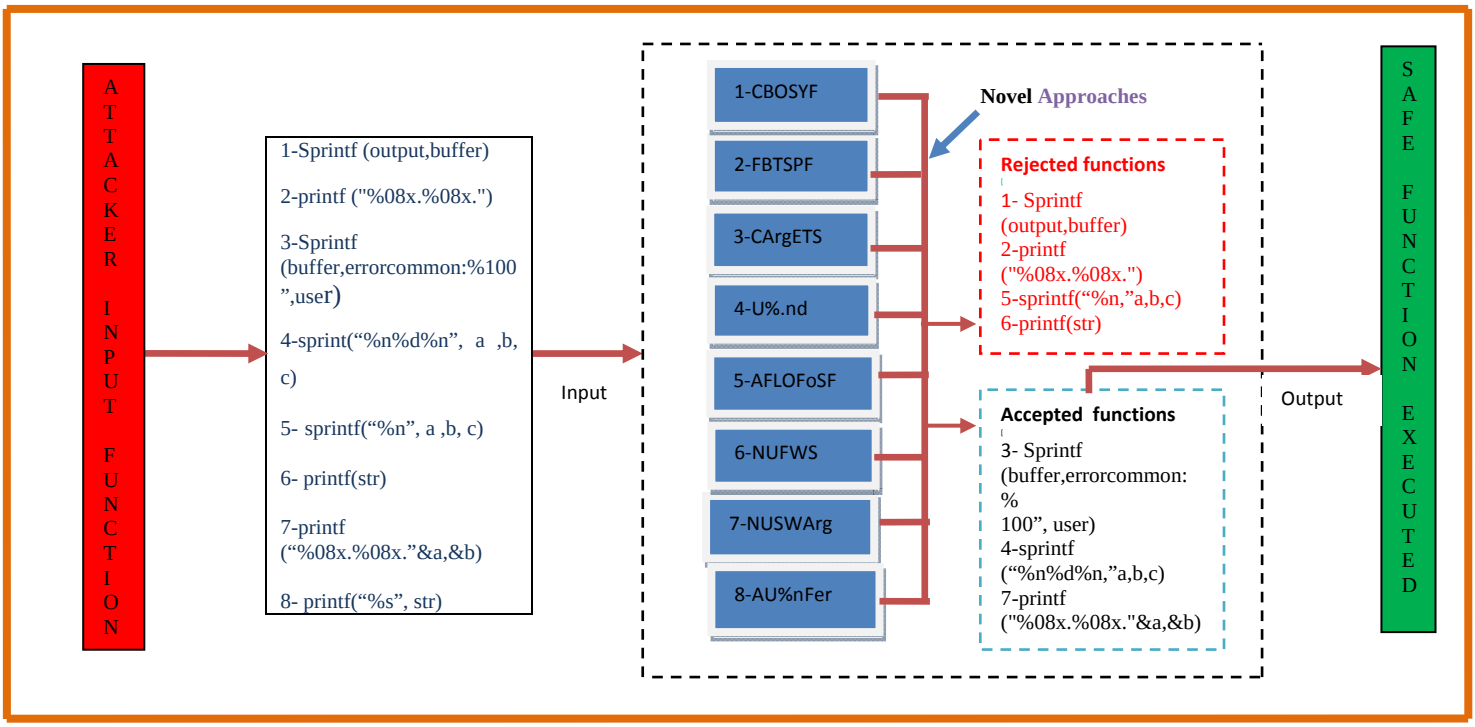

Figure 2- Coalesce Model for Preventing Format String Attacks

5- Comparisons of Coalesce Model with other Tool That Prevent Format String Attacks

TABLE3. COMPARISON OF COALESCE MODEL WITH OTHER PREVENTION TOOLS

Tool/ Work	Both Families covered(Printf and vsprintf)	Prevent to Stack read	Prevent to Stack write	Prevent to Argument retrieve	Check the Specifier width	Check the Specifier mismatch
Libsafe[3] tool	Yes	No	Yes	No	No	No
FormatGaurd[10] tool	No	Yes	Yes	No	No	No
PScan tool[17] tool	No	Yes	Yes	No	No	No
Lisbon[18] tool	Yes	Yes	Yes	No	No	No
Libformat [19] tool	Yes	No	Yes	No	No	No
Coalesce model	Yes	Yes	Yes	Yes	Yes	Yes

6 –CONCLUSION

Format String Bugs depth analyses report and the corresponding source code of the applications drive the development of Coalesce Models to depict and reason about software security vulnerability of the applications .Format string attacks are dangerous and most important security vulnerabilities that appeared in Year 2000 and continue to be a major cause of software vulnerabilities. This work proposed Coalesce model to testing of Format String Bugs that gives the safe function execution. The problem of testing which is observed in the extensive survey of the related work provided in this paper is not come under the Coalesce model. Coalesce Model is one of the most effective Model for defending against format string attacks. By applying our proposed model, an implementation can be tested for any type of Format String Bugs. So with the help of this Model the coming bugs can be reject or aborted and the loss comes by end user can be prevented. This Model reject the Format String Bugs in the source code of the applications which may lead various types of vulnerability such as arbitrary reading, writing , application crash and direct parameter access of the stack and heap memory. Coalesce Model check both (printf, vprintf) of Format Family Functions.

7. FUTURE WORK

Computer vulnerabilities corrupt our important data or steal confidential information such as user ID and Password of the system user; so we think that our future work is to design and implement finite state machine to prevent Format String attack. With the help of this Finite State Machine we improve our security performance and software vulnerability.

REFERENCES

[1] M. F. Ringenburg and D. Grossman, “Preventing format string attacks via automatic and efficient dynamic checking,” In Proceedings of the 12th ACM conference on Computer and communication security ACM Press, November 7–11-2005.

[2] S. Nanda, W. Li, L. chung Lam, and T. cker Chiueh, “Bird: Binary interpretation using runtime disassembly,” In Proceedings of the 4th IEEE/ACM Conference on Code Generation and Optimization (CGO’06), March 2006.

[3] Tsai, T., and Singh, N., “Libsafe 2.0: Detection of format string vulnerability exploits,” Technical report, Avaya Labs,February 2001. version 3-21-01,PP.1-5.

[4] Security Team. Pfinger format string vulnerability http://www .securiteam .com/unixfocus/ 6K00N1P3FQ.html.

[5] Security Focus. Proftpd shutdown message format string vulnerability. http://www.security focus . com/ bid/14381/info.

[6] US-CERT. Format string input validation error in wuftpd site exec() function. http://www.kb.cert.org vuls/id/29823.

[7] Shuo Chen, Zbigniew Kalbarczyk, Jun Xu, Ravishankar and K. Iyer, “ A Data-Driven Finite State Machine Model for Analyzing Security Vulnerabilities,” Center for Reliable and High-Performance Computing Coordinated Science Laboratory University of Illinois at Urbana-Champaign 1308 W. Main Street, Urbana, IL 61801

[8] Andreas thuemmel, “Analysis of format string bugs,” a.thuemmel@web.dc-version 1.0, Format String Buggs and SITE EXEC exploit against wu-ftpd on 15-02-2001.

[9] Hossain Shahriar and Mohammad Zulkernine, “ Mutation-based Testing of Format String Bugs,” School of Computing Queen’s University, Kingston, Ontario, Canada, In proceeding of the 11th IEEE High Assurance Systems Engineering Symposium in 2008,page no.229-238.

[10] Crispin Cowan, Matt Barringer, Steve Beattie, and Greg Kroah- Hartman, “FormatGuard: Automatic Protection From printf Format String Vulnerabilities,” WireX Communications, Inc. published in the preceeding Of the USENIX security Symposium in 15-August-2001, Washington Dc.

[11] Lap-chung Lam and Tzi-cker Chiueh, “ Automatic Extraction of Accurate Application Specific Sandboxing Policy,” Networks, Inc. 99 Mark Tree RD, Suite 301, Centereach NY 11720, USA

[12] Scut / team teso, “Exploiting Format String Vulnerabilities,” September 1, 2001 version 1.2.Accessed from http:/doc.bughunter.net/Format-String/exploit-fs.html.

[13] Pankaj Kohli and Bezawada Bruhadeshwar, “FormatShield: A Binary Rewriting Defense against Format String Attacks,” Centre for Security Theory and Algorithmic Research (C-STAR) International Institute of Information Technology Hyderabad,Spinger ACISP 2008,LNCS 5107 page no. 376-390.

[14] Format String Attacks, Tim Newsham, Guardent Digital Infrastructure, Inc.September 2000.

[15] Fang Yu, Muath Alkhalaf and Tevfik Bultan “Generating Vulnerability Signatures for String Manipulating Programs Using Automata-based Forward and Backward Symbolic Analyses,” 2009 IEEE/ACM International Conference on Automated Software Engineering,1527-1366/09,PP 605-609.

[16] Shuo Chen, Jun Xu, and Ravishankar K. Iyer “Non-Control-Data Attacks Are Realistic Threats,” Center of Reliable and High Performance

Computing Cordinated Science Laboratory, University of Illinois at Urbana-Champaign,1308 W.

[17] DeKok,A., "Pscan (1.2-8) Format string security checker for C files", http://packages.debian.org/ etch/ pscan(Accessed January 2008)

[18] Li, W. and Chiueh, T., "Automated Format String Attack Prevention for Win32/X86 Binaries", In proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC), Miami, December 2007, pp. 398-409.

[19] The Shellcoder handbook,2nd edition, discovering and exploiting security holes.

[20] ITS4: Software Security Tool, Accessed from http: //www. cigital.com/its4/.

[21] Silva, A., "Format Strings," Gotfault Security Community, Version 2.5, Nov 2005, Accessed from http://www.milw0rm.com/papers/5 (April 2008).

[22] Robbins, T., Libformat, http://archives.neohapsis.com/ archives /linux/lsap/2000-q3/0444.html (Accessed January 2008)