Advanced Format String Attacks
Presented by Paul Haas
Contents

- Background
- Abstract
- Definition
- Context
- Technique
- How-to
- Tools
- Exploits
- Conclusion
- Q&A
• Lead Web Application Security Engineer at Redspin, Inc with over 4 years experience in hundreds of audits.

• This talk is not associated with my company

• Defcon 13 CTF winner (Shellphish 2005)

• Alumni of UCSB’s Computer Security Group

• Mario Kart DS: Rob in Tank on Rainbow Road
#include <stdio.h>

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

Reading arbitrary locations
Writing arbitrary locations
Executing arbitrary code
Get a shell
Without

RTFM and Writing it yourself!
Format String Attack

- Software vulnerability in certain C functions that perform string formatting leading to the potential to write arbitrary data to arbitrary locations.

- Despite easy solutions, vulnerabilities and ignorance of issue still exist, hence the talk.

- Common in hackademic exercises.

- Talk assumes you have a basic idea of the attack (%x, %s, %n).

- Talk details technique but tools do not require it.
Brief History

• 1990: csh “Interaction Effect” crash: !o%8f

• 1999-09-17: proftpd 1.2.0pre6 “Argument attack/ snprintf Vulnerability” (BID 650)

• 2000-06-22: wu-ftp 2.6.0 Remote Format String Stack Overwrite Vulnerability (BID 1387)

• 2000-09-09: “Format String Attacks” whitepaper by Tim Newsham

• 2010-06-30: KVIRC DCC Directory Traversal and Multiple Format String Vulnerabilities (BID 40746)
Old Technique

• Manual popping up of stack using string of ‘%x’s
• Get overwrite address using other technique
• Search for shellcode in core after SEGFAULT
• Characters written using long value in %x or %c
• Final write to address using %n
• Frequent RTFM
• Write once, use once
Current Technique

• `%p` gives detailed information of stack location
• `%s` allows us to view known stack addresses as strings
• `%NNc` controls number of bytes written
• `%hhn` allows single byte writes
• Direct parameter access shortens format string:
  \[
  \%5$n = \%p\%p\%p\%p\%p\%n
  \]
New Technique

• Format String Attack allows us to dump stack
• Stack contains interesting information:
  data, code pointers, stack addresses
  our format string, format string’s address
  stack offset location of all of the above
• Knowledge of this gives us the address of any value on the stack
• These values are enough to write our exploit
Our Vulnerable Code

```c
#include <stdio.h>

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

# Compile and setup insecure environment

gcc printf.c -w -O0 -ggdb -std=c99 -static -D_FORTIFY_SOURCE=0 -fno-pie -Wno-format -Wno-format-security -fno-stack-protector -z norelro -z execstack -o printf

sudo sysctl -w kernel.randomize_va_space=0
```
Exploit Steps

• Dump stack values until format string is found
• Locate pointer address of format string
• Choose our overwrite address on the stack
• Point format string at overwrite address and write address of shellcode to end of string
• Adjust offsets for ‘chicken and egg’ problem:
  Address of format string based on its length
  Format string needs its own address to reference
• Method 1: Pass a long string of %p’s
  .`/printf `perl -E `say “%p”x200``

• Method 2: Execute binary in loop with %NNN$p
  for i in {001..200}; do echo -n “$i = “ ; ./printf “%$i\$p”; echo; done

• Search for hex representation of string
  $ = 0x24, % = 0x25 , p = 0x70

• Result will be stack offset of format string
Format String Address

• Execute binary in loop with sequential \%NNN$\s
  Will cause SEGFAULTS, may trip any IDS systems
  
  for i in {001..100}; do echo -n "$i = " ; ./printf "\%$i\$p:\%$i\$s"; echo; done | grep -v ^$

• Create format string only comprising of addresses obtained from stack dump

  Single execution/string prevents SEGFAULT

  Much more elegant, verifies constant stack
Offset + Address = WIN

Matching up an offset to a stack address allows us to learn the address of any location on the stack

Example:

Offset 100 (0xBFFFFFF100): Our format string

sizeof(pointer) = 4 bytes * 100 pointers = 400

Offset 1 = 0xBFFFFFF100 + 400 = 0xBFFFF290
Common exploit locations require binary examination tools: PLT, DTORS, LIBC

Advance format string attack could extract these from known binary headers (difficult)

Return addresses are stored on the stack

We know the stack address of each value

Overwrite these locations to point to shellcode
Issues

• Different format strings lengths effect stack addresses, yet we assume stack is constant
  
  Keep all strings to same modulus of sizeof(pointer)

• Format string may not align with stack address
  
  Keep padding requirement when addressing string

• Even with the correct modulus and pad, our string offset may be off
  
  Verify our exploit before we attempt it by reading rather than writing to our overwrite location
Result

- It is possible to create a format string exploit using only 2 executions of the vulnerable program with no program exceptions
- Math only, no bruteforcing necessary
- Incorporate shellcode as part of format string
- Smaller format string buffers are also possible
  
  8 bytes to examine a stack address
  
  Format string as small as 68 bytes + shellcode
Format String Auto Exploitation

• Proof of concept tool in Python
• Instructions for running on Backtrack 4
• Multiple exploit and overwrite options
• Missing some useful features:
  Separate execution of independent steps
  Architectures independent (x86 & 64)
  Read arbitrary locations rather than write
  Finer control over exploit
Metasploit Integration

- Control each step of the exploit individually or automate entire process
- Use as payload generator
- Uses Metasploit payload library for shellcode
- Integrates into other modules and injection functionality
- Functionality will be demonstrated during Defcon
Demonstrations

- Testing Code
- OverTheWire
- Known exploit
- 0-Day?
Summary

• The output from format string attacks gives you everything you need to know to go from discovery to compromise

• The exploitation process can be automated from start to finish

• Format string attacks are easy to fix, and now are easy to exploit as well

• There are plenty of vulnerable programs out there to discover and exploit
Questions?
Thanks

• The most recent version of this presentation and associated tools can be found on www.redspin.com and www.defcon.org

• Look for the incorporation of the tools in this talk in Metasploit in the near future

• Any follow-up questions can be addressed to phaas AT redspin DOT com

• Shouts to the Shellphish, G. Vigna “zanardi” and the Goats at Redspin {ap3r, jhaddix, fulg0re, D3, OwNpile, Yimmy & b3tty}