MODERN VULNERABILITY EXPLOITATION: FORMAT STRING ATTACKS

History of Format String Attacks

- Format String Attacks
  - First noted in 1990 as a result of fuzz testing on csh
  - First used as an attack vector on ProFTPd
    - 1999 Bugtraq mailing list
  - Became popular in 2000
    - Paper appropriately called "Format String Attacks"

Format Strings

- Format Strings
  - Used to display values in various formats
  - printf(fmt_string, arg1, arg2, …);
    - Format string escape character: %

```c
void main(int argc, char** argv) {
    int    i  = 123;
    double d1 = 12.345;
    double d2 = 10.75;
    char  *s1 = "Unit";
    char  *s2 = "Value";

    printf("%s% 10s\n", s1, s2);
    printf("---------------\n");
    printf("Hex:     %d (0x%0.4x)\n", i, i);
    printf("Money:   $%0.2f\n", d1);
    printf("Percent: %0.2f%%\n", d2);
}
```

```
format>not_vuln.exe
Unit      Value
---------------
Hex:      123 (0x007b)
Money:    $12.35
Percent:  10.75%
```

Format String Functions

- Format String Functions
  - printf()
  - fprintf()
  - sprintf()
  - snprintf()
  - vfprintf()
  - vprintf()
  - vsprintf()
  - vsnprintf()
  - wprintf()

Format String Formatters

- Format String Formatters
  - %c
    - Print a character
  - %d
    - Print a decimal
  - %e, %E
    - Print a float or double in signed E notation (1.3E3)
  - %f
    - Print a float or double in decimal notation (like 12.345)

Format String Formatters

- Format String Formatters
  - %o
    - Print an octal number
  - %p
    - Print a pointer (equivalent to %o.8X)
  - %s
    - Prints the string at address
  - %x, %X
    - Print a hex number

Format String Attack

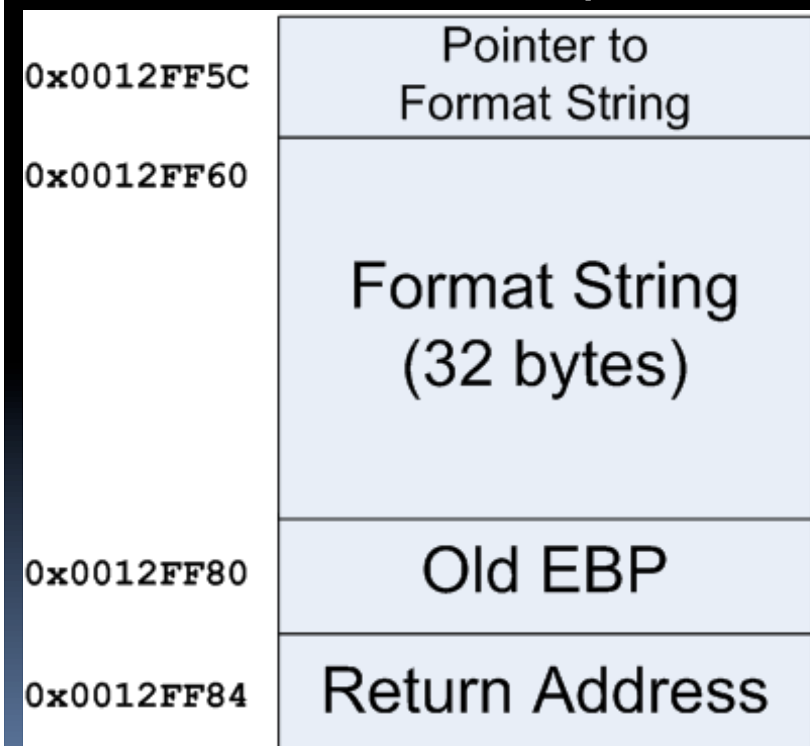
- Format String Attacks
  - The vulnerability is called a format string bug
  - Occurs when a hacker can control the format string

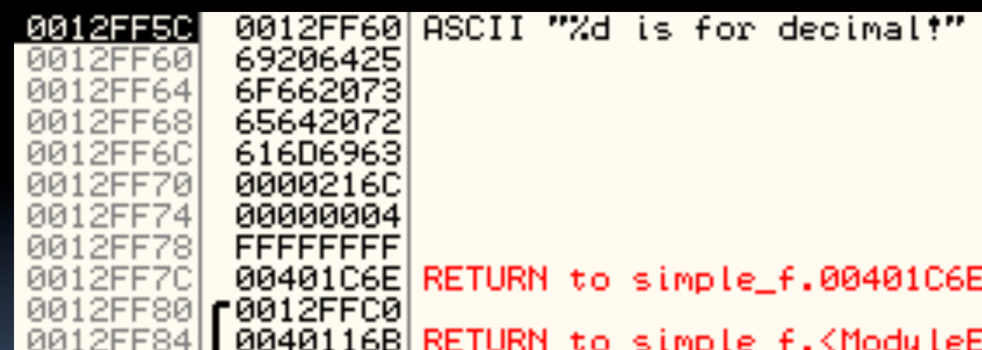
```
void main(int argc, char** argv) {
    char s[32];

    gets(s);
    printf("You typed: ");
    printf(s);
    printf("\n");
}
```

```
format>simple_format.exe
%d is for decimal!
You typed: 1763730469 is for decimal!
```

Format String Attack

- Analysis of the Format String Attack
  - When the printf() call is made, the stack looks like:





Format String Attack

- Analysis of the Format String Attack
  - What is 1763730469?

```
format>simple_format.exe
%d is for decimal!
You typed: 1763730469 is for decimal!
```

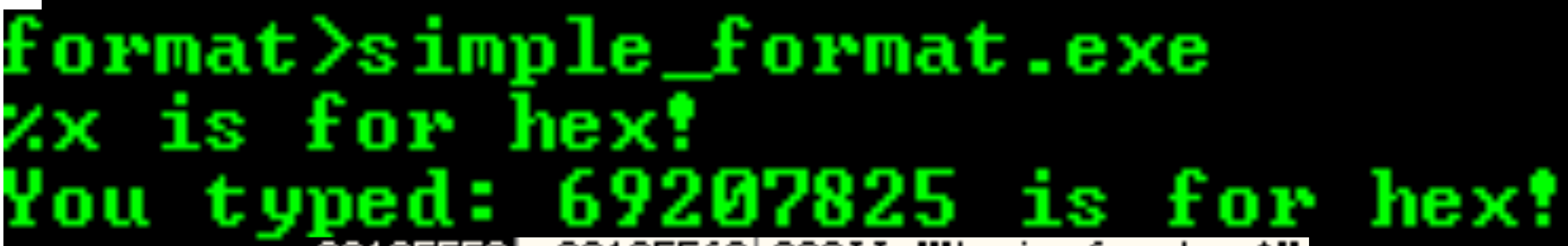
  - Let's try %x instead of %d

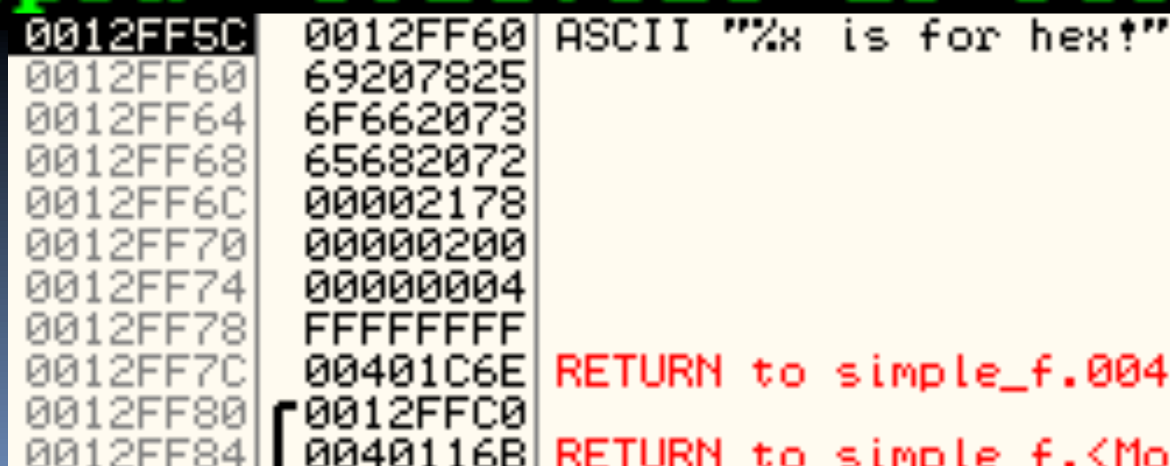
```
format>simple_format.exe
%x is for hex!
You typed: 69207825 is for hex!
```

  - 1763730469 != 0x69207825… Let's see why

Format String Attack

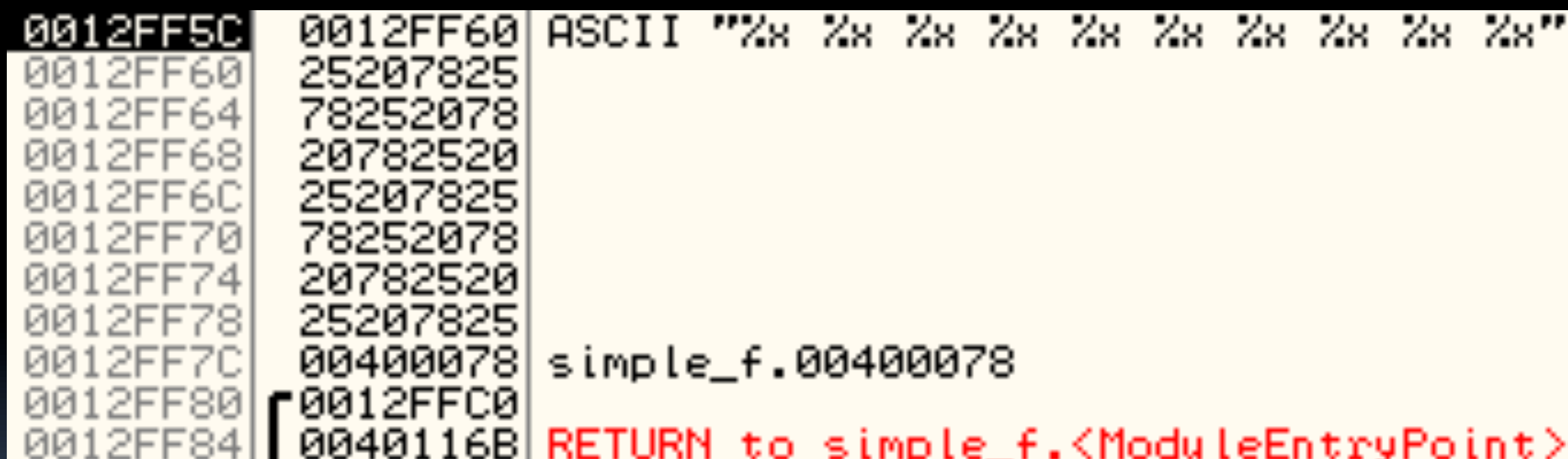
- Analysis of the Format String Attack
  - If we take a look at our stack, it all makes sense
  - 0x69207825 is taken as a parameter!

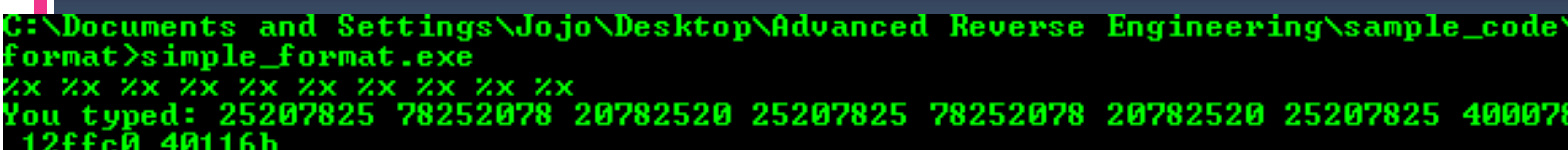




Memory Disclosure and the Format String Attack

- Format String Attack
  - Memory disclosure is a serious vulnerability





Memory Overwrite and the Format String Attack

- Format String Attack
  - %n formatting character
    - Writes the number of characters printed thus far to an integer pointer
    - Aka 4-byte overwrite address pointer

```c
void main(int argc, char** argv) {
    int i1, i2, i3;

    printf(".%n\n", &i1);
    printf("..%n\n", &i2);
    printf("...%n\n", &i3);
    printf("%d, %d, %d\n", i1, i2, i3);
}
```

```
format>simple_format2.exe
.
..
...
1, 2, 3
```

Memory Overwrite and the Format String Attack

- Format String Attack

```
0012FF5C  0012FF60  ASCII "%x %x %x %x %x %x %x %x %x %n"
0012FF60  25207825
0012FF64  78252078
0012FF68  20782520
0012FF6C  25207825
0012FF70  78252078
0012FF74  20782520
0012FF78  25207825
0012FF7C  0040006E  ASCII "S mode.♪♪◘$"
0012FF80  0012FFC0
0012FF84  0040116B  RETURN to simple_f.<ModuleEntryPoint>+0B4 from simp
```

```
00401780  . 8908  MOV DWORD PTR DS:[EAX],ECX
```

```
EAX 0040116B
ECX 0000004D
```

Access violation when writing to [0040116B]

Memory Overwrite and the Format String Attack

- Format String Attack
  - %n formatting character
    - Continues counting (doesn't reset after another %n)
    - This means our write values are ever-increasing

```c
void main(int argc, char** argv) {
    int i1, i2;

    printf(".%n.%n\n", &i1, &i2);
    printf("%d, %d\n", i1, i2);
}
```

```
format>simple_format3.exe
..
1, 2
```

Memory Overwrite and the Format String Attack

- Format String Attack
  - %hn is available in some printf implementations
    - 16-bit (2-byte) overwrite
    - Ex.: We can overwrite 0xFFFFFFFF with 0xFFFF0001

```
void main(int argc, char** argv) {
    int i = -1;

    printf(".%hn\n", &i);
    printf("%d\n", i);

}
```

```
0012FF74   00406030  ASCII ".%hn◘"
0012FF78   0012FF7C
0012FF7C   FFFFFFFF
0012FF80  ┌0012FFC0
0012FF84  │00401116  RETURN to simp
```

```
0012FF74   00406030  ASCII ".%hn◘"
0012FF78   0012FF7C
0012FF7C   FFFF0001
0012FF80  ┌0012FFC0
0012FF84  │00401116  RETURN to simp
```

```
format>simple_format4.exe
.
-65535
```

Format String Attack Essentials

- Need to Know and Understand
  - The number of chars printed is our write value
  - The aligned parameter on the stack for our %n or %hn is our write address

Format String Attack Formatter Sizes

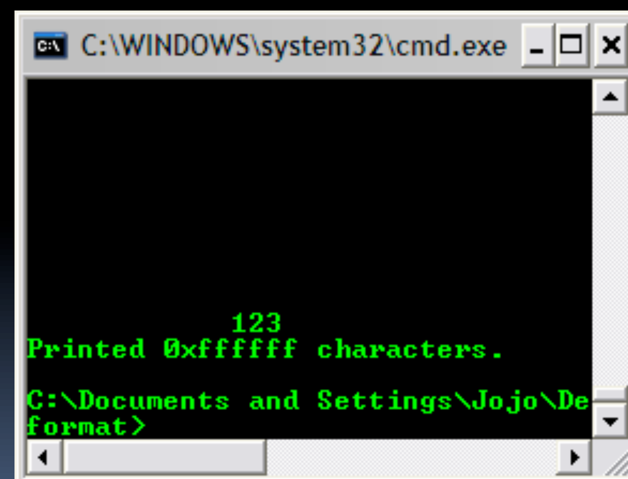
- Format String Formatter information
  - Keeping our attack string small saves space
    - Space that could be used for NOP sled/shellcode
  - Sizes
    - %c: 2 fmt / 4 mem / 1 count
    - %d, %x…: 2 fmt / 4 mem / ? count
    - %p: 2 fmt / 4 mem / 8 count
    - %f: 2 fmt / 8 mem / ? count
      - Risky, divide by zero is possible
    - %.f: 3 fmt / 8 mem / ? count
      - Avoids division by zero
    - %s: variable (2 fmt / n mem / n-1 count)
      - Up to and including first null byte
      - Risky, can access violate

Creative Format String Attacks

- Large Scale Memory Disclosure
  - Use %s to disclose memory up to the first null byte
  - It is possible to access violate if you hit the end of the stack before you find a null terminator
  - However, null bytes are numerous
    - So you can bank on this trick a few times

Creative Format String Attacks

- Format String Memory Math
  - Use predictable values in memory to help you!
  - There exists a * qualifier
  - Can safely print a large number of chars
    - Can take a long time (minutes)




```c
void main(int argc, char** argv) {
    int i;

    printf("%*d%n\n", 0xffffff, 123, &i);
    printf("Printed 0x%x characters.\n", i);
}
```

  - Handy trick, especially for harder-to-hack Windows

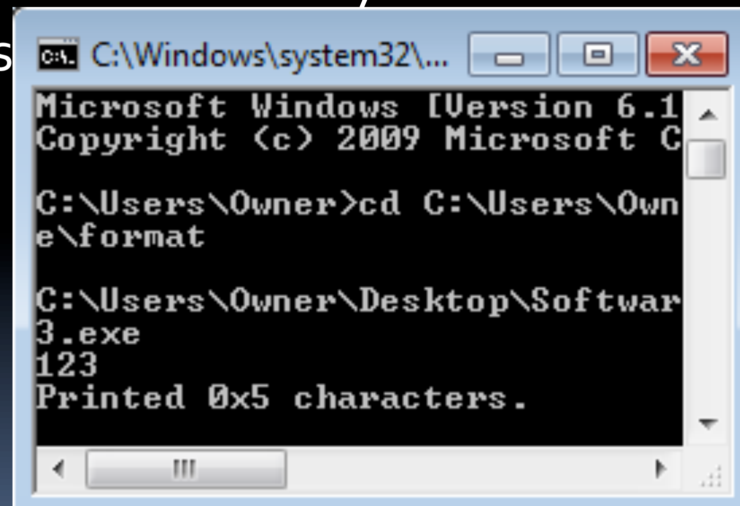
Creative Format String Attacks

- Format String Memory Math
  - If your format string is on the stack, use it!
  - Lowland addresses are hard to generate
    - You cannot have null bytes in your format string

Creative Format String Attacks

- Format String Memory Math
  - Hide lowland addresses as their negative value
  - The * qualifier will left justify for negative params
  - Can safely print a large numbers with null bytes
    - Now we can write lowland vals

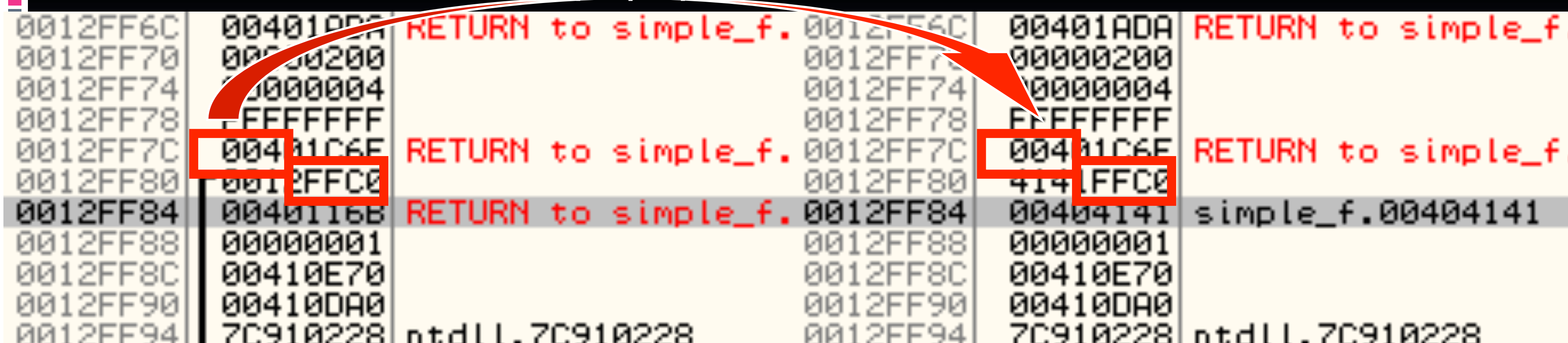
```
void main(int argc, char** argv) {
    int i;

    printf("%*d%n\n", -5, 123, &i);
    printf("Printed 0x%x characters.\n", i);
}
```



  - Handy trick, especially for harder-to-hack Windows

Creative Format String Attacks

- Advanced Format String Alignment
  - You are not limited to 4-byte write alignments
    - (in x86)
  - The address %n uses is byte-aligned
  - Ex:
    - Write 0x41414141 to 0x0012FF82



Linux Format String Attacks

- Direct Parameter Access
  - Not implemented by Windows (msvcrt.printf)
  - `%3$x`
    - Grab the 3rd parameter
  - Makes most formatting string attacks very compact
    - Ex: Write 0x0012FFC0 to 200th parameter
    - "`%.621786x%.621786x%.8x%.8x…%.8x%.8x%n`"
      - 816 characters
    - Reduces to "`%.622560x%.622560x%200$n`"
      - 24 characters

Windows Format String Attacks

- Windows Format String Limitations
  - Beware width specifiers!
    - Some limit the maximum number
      - WinNT 4.0, XP: %.516x maximum
      - Win2000: allows large values (%.622496x)
    - All typically store the expanded format string on the stack
      - Long format strings (like %.622496x) can overflow the stack
      - Crashes the program
  - No direct parameter access
    - Nope, none. Bummer.
    - So parameter alignment and size cognizance is important

Windows Format String Attacks

- Windows Format Formula
  - Math seed
    - Some (usually large) number we can embed for our * qualifier
  - Series of %p
    - To align the current parameter to our math seed
  - %*p
    - Use our math seed on the stack to boost our math
  - Series of %p
    - To align the current parameter to our address
  - %n or %hn
    - The number of printed chars is the address of our NOP sled
  - NOP sled/shellcode
    - NOP sled can be easily expanded to align our address
  - Overwrite address
    - This value is the address of our target function pointer

Capabilities of a Format String Attack

- DoS
  - Crash a service with a group of %n's
- Memory Disclosure
  - Reveals protected information
  - Precursor to defeat code execution protections
- Arbitrary Code Execution
  - Arbitrary memory overwrites make standard and advanced code execution vectors possible
- Protection Mechanism Bypass
  - A memory corruption exploit can overwrite a format string in memory to help bypass protections

Targets for a Format String Attack

- Control Pointers
  - Return pointer (ret)
  - Stack exception handlers (SEH)
  - Global offset table (GOT)
  - Virtual function pointers (vtable)
  - PEB function pointer
  - Thread environment block (TEB)
  - Unhandled exception filter (UEF)
  - Vectored exception handling (VEH)
  - Destructors (.DTORS)
  - atexit handlers
  - C library hooks
  - Callbacks
  - Function pointers in general
- Data
  - Any variables

Questions/Comments?