Secure C Coding

...yeah right

- Andrew Zonenberg | Alex Radocea

Agenda

- Some Quick Review
  - Data Representation
  - Pointer Arithmetic
  - Memory Management
- Basic C Vulnerabilities
  - Memory Corruption
  - Ignoring Return values
  - Typos

Everything is made of bits

```
int main(){
    char one[] = ”JARS”;
    char two[] = {0x74, 0x65, 0x82, 0x83};
    short three[] = {16714, 21330};
    int four = 1397899594;
    float five = 9.03038500864E11;
   __asm{
       dec edx
       inc ecx
       push edx
       push ebx
   }
}
```

Two's complement trivia

- Under 32-bit signed number arithmetic using 2's complement number representation:

  What is abs(-2147483648)?

C string representation is all about the NUL byte termination

- 47 4f 4f 53 45 00
  GOOSE.|
- char buf[]=”hi”;
  sizeof(buf) = ?

Photo Credit:
http://www.flickr.com/photo
/benimoto/911325473/

Pointer Arithmetic Quiz

```
void *x = 0x1337c000;
char *c = (char *)x;
short *s = (short *)x;
int *i = (int *)x;
double *d = (double *)x;

x + 1 = ?
c + 1 = ?
s + 1 = ?
i  + 1 = ?
d + 1 = ?
```

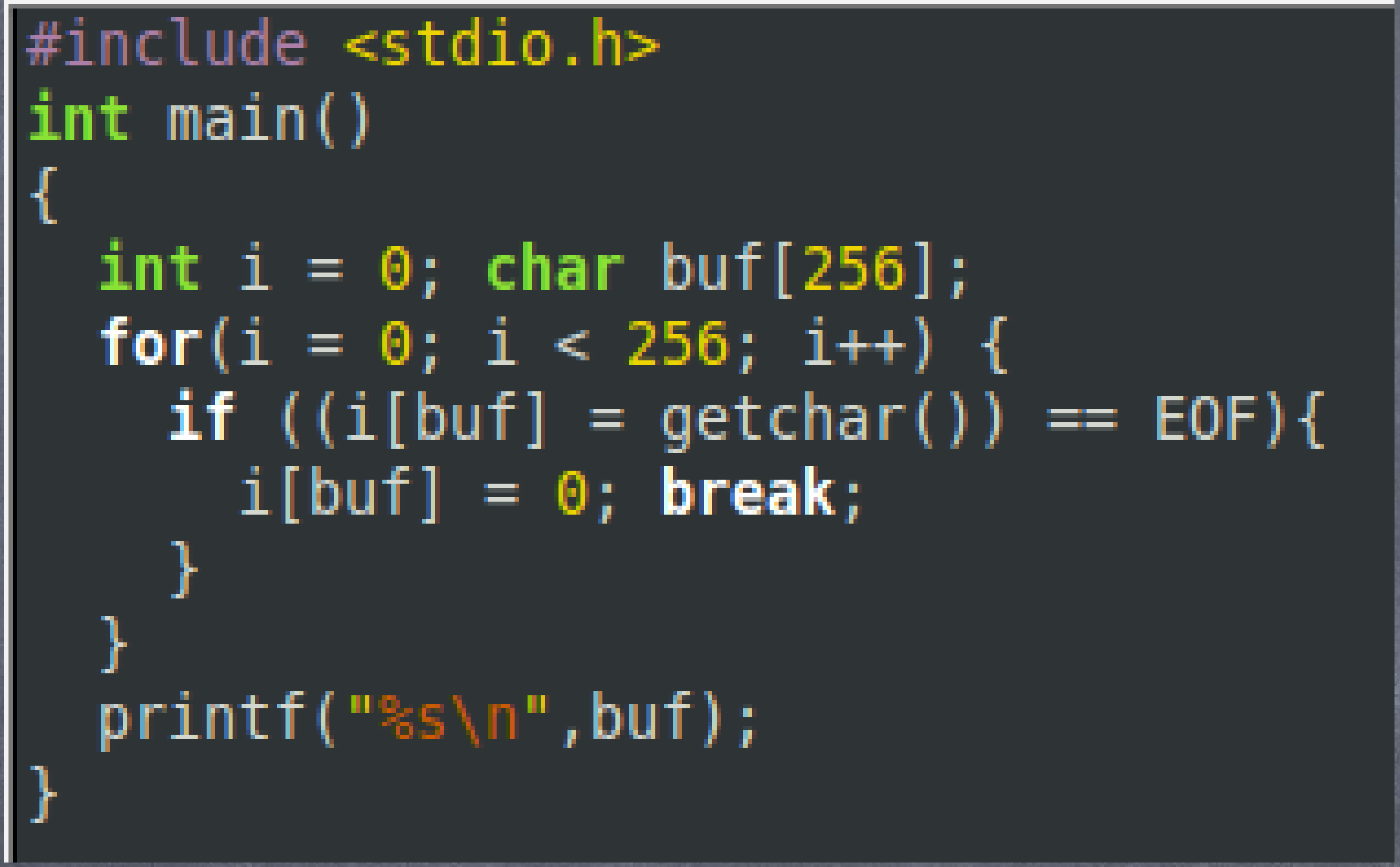
This is the pattern.

- (ptr *)p + count => p + sizeof(ptr_type)*count
- double *p = 400;
  p + 5 => p + sizeof(double)*5 = 440
- unsigned short *x = 400;
  x + 10 => ??

Even the ”hex”perts get it wrong.

- CVE-2009-3234
- Incomplete fix for buffer overflow in perf_copy_attr, signed off by core developer(s)
- Vulnerable code should always get special care and attention, where there's one bug there's often many more.
- http://lkml.org/lkml/2009/9/19/155

Pointer Trivia



```
#include <stdio.h>
int main()
{
  int i = 0; char buf[256];
  for(i = 0; i < 256; i++) {
    if ((i[buf] = getchar()) == EOF){
      i[buf] = 0; break;
    }
  }
  printf("%s\n",buf);
}
```

- Will this compile? What happens?

Memory management in a nutshell

- The Stack
  - Fixed size buffers*
  - Flow control information
    - Function pointers
    - Activation records
  - Implicitly cleaned up
  - Uninitialized

- The Heap
- Dynamic size
- Flow control informatio
  - Function pointers
  - Internal memory structures
- Explicitly cleaned up
- Uninitialized

Stack → First in First Out

```
int func(int a, int b, int c){
  int x;
  char y;
  FILE* f;
  char buffer[1000];
  ...
  func(1,2,3);
  ...
}
```

etsylove.ning.com

Misc Stack Info

- Stack cookies mitigate buffer overflows
- Security mechanisms rearrange variable allocation where possible to ensure cookies work, prevent pointer overwrites
- alloca(int sz); → dynamic stack allocation
- Void func(int sz){ int buf[sz]; };
  C99 variable-length arrays ->Phrack 63-13

Heap allocation

- C-style
- buf = malloc(sz);
- free(buf);

- C++
- buf = new char[sz];
- delete []buf

Heap Zoo

- Linux – doug lea malloc based implementations
- FreeBSD – phkmalloc
- Windows – RTL heap
- Mac OS -- Bertrand Serlet
- Older unixes → (System V) - tree based heap

Heap Misc Info

- Pointers, flags, and other control information used to manage the chunks
- Control information can be used for generic exploitation (”Once upon a free()...” Phrack 57-9)

More Info

- realloc() is extremely tricky to use correctly
- Forgetting to free memory is a memory leak
- Memory allocation functions fail

Memory corruption

- Data is overwritten or modified to enter an ”undefined” program state.
- Causes include arithmetic errors, bad error checking, uninitialized memory usage, and unintended code flow paths.
- Not a recoverable state (some programs will try anyway)

What is wrong with this code?

```
int main(int argc, char *argv[]){
  char buf[256];
  strcpy(buf,argv[1]);
}
```

A typical attack scenario

- 1) Hijack control flow information (function pointer, return address) with memory corruption
- 2) Redirect execution to an unexpected state or injected code (shellcode)
- 3) Install backdoor, maintain access

Common Terminology

- Stack overflow → ran out of stack memory (recursive function)
- Buffer overflow/overrun → data is copied beyond the end of the buffer
- Buffer underrun → data is copied before the start of the buffer

Spot the bug in thttpd defang

```
static void defang (char* str, char* dfstr, int dfsize )
{
    char* cp1;   char* cp2;
    for ( cp1 = str, cp2 = dfstr; *cp1 != '\0' && cp2 - dfstr < dfsize - 1; ++cp1, ++cp2 )
    {
     switch ( *cp1 )
     {
     case '<':
       *cp2++ = '&';  *cp2++ = 'l';  *cp2++ = 't'; *cp2 = ';';    break;
     case '>':
       *cp2++ = '&';  *cp2++ = 'g';  *cp2++ = 't'; *cp2 = ';';   break;
     default:
               *cp2 = *cp1;      break; }
     }
     *cp2 = '\0';
 }
```

Ignoring return values has security implications

- Improper privilege separation
- Unexpected system states
- Memory corruption
- Uninitialized memory

Trivia

- initgroups(USER, pw->pw_gid);
- setgid(pw->pw_gid);
- setuid(pw->pw_uid);
- execv("/bin/sh",0);

- Which functions can fail?

Hint: only one function to misuse

```
void func(int fd){
char buf[256];
char *ptr = buf, *end = &buf[sizeof(buf)];
buf = ptr;
  while(ptr < end){
    ptr += read(fd, ptr, 1);
  }
}
```

See Lars' CVE-2009-0017

Typos

- Typos in C, C++ can be hilarious
- Only takes a few characters
- Awesome.

Isn't this cute?

```
if(authenticated=1){
   do stuff
}
```

This too, right?

-
  ```
  if(!authenticated);
     return
  ```

What's wrong with this code?

```
char * func(int fd)
{
   unsigned int len;
   len = read_data(4);
   char *data = malloc(len);
   recv(fd, &data, len, 0);
   return data;
}
```

Spoiler page

- Similar to ActiveX bugs that came out last summer
- Ironically code is from ”security enhancements”

```
hr = pStream->Read((void*)&pbArray,
(ULONG)cbSize, NULL);
```

should be

```
hr = pStream->Read((void*)pbArray,
(ULONG)cbSize, NULL);
```

http://arstechnica.com/microsoft/news/2009/07/a-single-extra-resulted-in-ie-exploit.ars

Oops

- Obj *o = new obj[100];
- delete o;

Constants

- #define SZ 40
- char buf[20]; strncpy(buf, src, SZ-2); buf[SZ-1] = 0;
- Constants are signed by default (0 vs 0U).

Upcoming

- Advanced heap issues
- Off by ones
- Integer safety
  - underflows, overflows, signedness
  - truncation, typecasting