

University of Rochester
CSC290B
Introduction to Computer Security

Bugs

February 12, 2009

Software Flaws

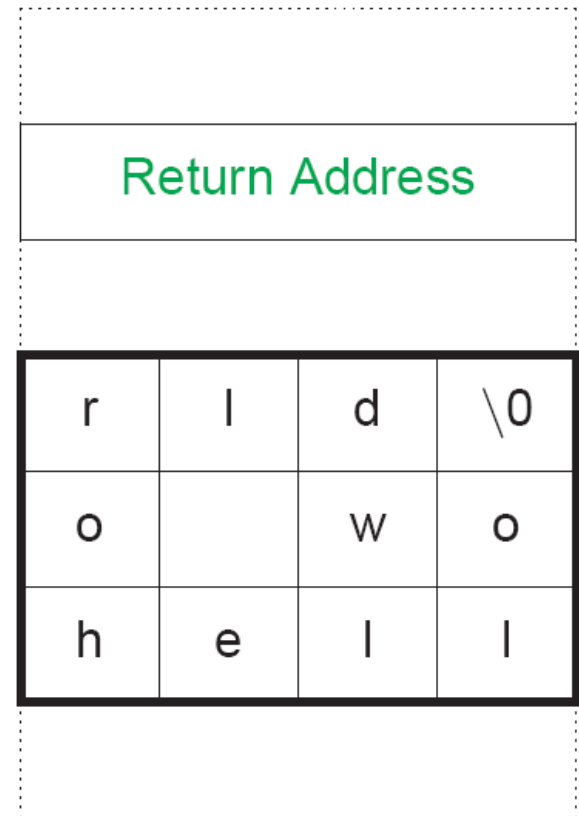
- Design flaws are bad enough
- Implementation flaws (bugs)
 - Violate the assumptions of the software authors
 - Enabling program behavior that's undefined or not predicted by the software authors

Buffer Overflows

- Responsible for about half of all security vulnerabilities
- Fundamental problems:
 - Character strings in C are actually arrays of chars
 - There is no array bounds checking done in C
- Attacker's goal: overflow the array in a controlled fashion

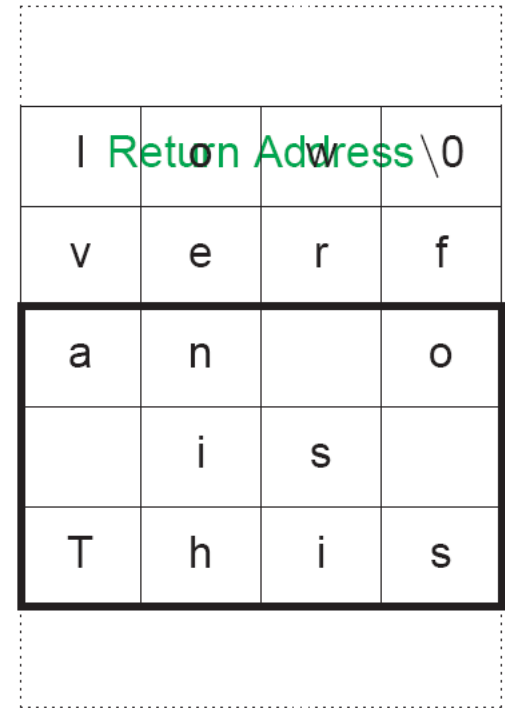
Stack Frame

- When a function is called, the return address is stored on the stack. Lower in memory, all local variables are stored.



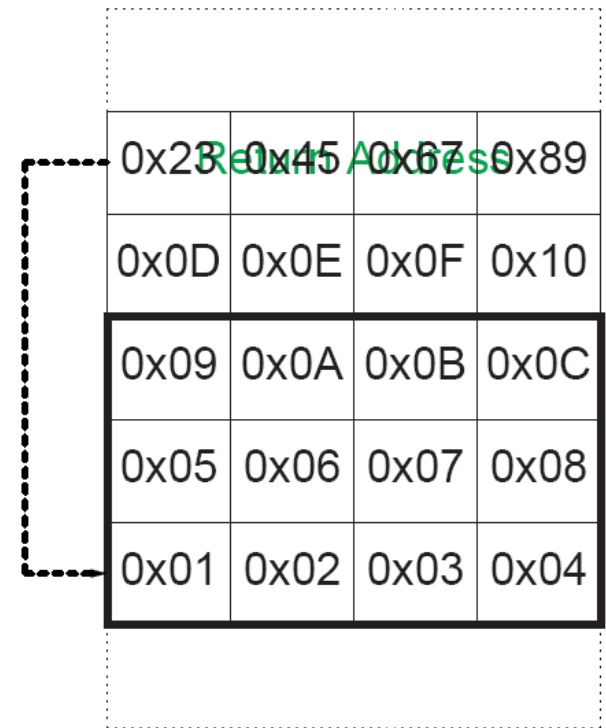
Buffer Overflow

- If the array bounds are exceeded, the return address can be overwritten.



Buffer Overflow Attack

- Put code in the early part of the buffer, then change the return address to point to it. When the function exits, the injected code is executed.



How Can Such Things Happen?

- C has lots of built-in functions that don't check array bounds
- Programmers frequently don't check, either
- The attacker supplies too-long input

Sample Problematic Code

```
char line[512];
```

```
...
```

```
gets(line);
```

- That's from the 4.3BSD fingerd command, exploited in 1988...

Bad versus Good

- `gets()` `fgets()`
- `strcpy()` `strncpy()`
- `strcat()` `strncat()`
- `sprintf()` `snprintf()`

Java vs. C

- “Daddy, what’s an `IndexOutOfBoundsException`?”
- “It’s why I’m teaching you Java instead of C.”

Indirect Buffer Overflows

```
void f(char *s)
{
    sprintf(s, "....");
}

void g()
{
    char buf[128];
    f(buf);
}
```

- Function `f` doesn't even know the size of the array!

Issues for the Attacker

- Finding vulnerable programs
- NUL bytes
- Uncertainty about addresses

Finding Vulnerable Programs

- Use nm and grep to spot use of dangerous routines
- Probe via very-long inputs or application-specific torture tests (fuzzing)
- Look at source or disassembled/decompiled code

NUL Bytes

- C strings can't have embedded 0 bytes
 - Some instructions do contain them
 - Solution: use different instruction sequence
- Some addresses may contain them
 - Use different address (see address uncertainty, below)

Address Uncertainty

- Pad the evil instructions with NOPs
- This is called a landing zone
- Set the return address to anywhere in the landing zone

Canaries

- Compiler trick — available for gcc and Microsoft compilers
- Insert a random “canary” value between the return address and the rest of the stack frame
- Check if it’s intact before returning
- Any stack-smash attack will have to overwrite the canary to get to the return address
- Use a different random value each time the program is executed

Randomization

- Put stack at different random location each time program is executed
- Put heap at different random location as well
- Defeats attempts to address known locations
- But — makes debugging harder

Checking Code

- Look for suspect calls with static checkers
 - Sophisticated static checkers analyze data flow across procedure boundaries
- Use language feature like Perl's “taint” mode

Stack versus Heap or BSS Storage

- Easiest to exploit if the buffer is on the stack
- Exploits for heap- or BSS-resident buffers are also possible, though they're harder
- Heap and BSS attacks not preventable with canaries (but there are analogous techniques to protect malloc())
- Some operating systems can make such memory pages non-executable, which is a big help – an excellent default - but it breaks some applications (e.g. JIT compilers)

History of Buffer Overflows

- Long-recognized as a security issue
- First very visible exploit: Robert T. Morris' Internet Worm, November 1988.
- Popularized by Aleph One in November 1996; serious threat since then
 - The attack is theoretically difficult, but there are many canned exploit kits available now

Format String Errors

- Suppose `str` is input to the program
 - Wrong:

```
printf(str);
```
 - Right:

```
printf("%s", str);
```
- Format strings can be dangerous. . .
- Note: other functions (i.e., `syslog`) also take format strings

The %n Problem

- Rather complex; I won't try to explain the details here
- Fundamental issue: %n writes to a variable the number of bytes printed this far
- The statement

```
printf("Hello\n%n", &cnt)
```
- stores a 6 in integer variable cnt
- This can be used to overwrite memory locations
- Use tricks involving other references to (non-existent!) other arguments to let you write to someplace “useful”

The Underlying Issues

- Problem 1: C has strange semantics
 - The only defense is to know the language thoroughly
 - You also have to know possible exploits
 - There are integer overflow attacks, too
 - E.g. unexpected results when array indexes are cast between signed and unsigned values
- Problem 2: programs don't always validate their inputs

Input Validation

- Trust nothing supplied by the user
- Must define inputs before they can be checked
- “A program whose behavior has not been specified cannot be buggy, only surprising.”
- Example: is a newline a valid character in a username?

Filtering

- Example: `fgets()` stops at a newline; you can't find any embedded
 - But watch for unterminated buffer — what if the input line is too long?
- Note that `argv` has no such guarantee
- Email: check recipient name against `/etc/passwd` — no funny characters there

Being Careful Near the Shell

- If user input is being passed to the shell, be especially careful
- Watch for `popen()` and `system()`
- Dangerous characters include:
`~\$^&()={ } [] | ; : ' " ? < > \
- That's most of the special characters!
- You're always much better off with a “good” list than a “bad” list
- Example: on some Unix systems, `^` is treated the same as `|`. Why?
 - Because on some models of Teletype, `^` printed as `↑`, which looked similar

Knowing the Semantics

- Sometimes check that there are no / characters in a program name
- Why? To ensure that the reference is to a file in the current directory
- Do you need to check \ as well?
 - Will the program ever run on Windows? Note that URLs on Windows use /, but the file system uses \

Summary

- Trust nothing
- Specify acceptable inputs
- Check everything
- Understand the semantics of anything you invoke
- Try to use a better language than C