Software development security 101

Marc Espie <espie@lse.epita.fr>

April 17, 2018

Introduction

There are more conferences for attackers than conferences for safety. That is the problem.

Theo de Raadt

Introduction

Basic goals

- Re-explain the basic ecosystem of software from a security perspective
- Give you enough vocabulary to pass internship questions
- Dispell misconceptions about development security

Advanced goals

- Modern mitigation and development techniques
- Introduction to source-code review and auditing (from a security perspective)

Prerequisites

Setting limits

- You've mostly done C and Unix so far
- That does always matter
- And a few problems which are not C specific

The fine print

- All your fancy languages have C/C++ runtimes
- Unix has a fine security model

Bibliography

- Building Secure Software (Viega, McGraw, ISBN 0-201-72152-X
- OpenBSD papers: `http://www.openbsd.org/papers/`
- Ted Unangst's FLAK: `http://www.tedunangst.com/flak/`

How to pass the exam

Simple questions

- You will have access to lecture notes
- So if I ask you to define a term, you shouldn't copy your notes
- You should be able to *demonstrate* that you *understand* the term by explaining it in your own words.
- There should be nothing fuzzy about it, give concrete examples
- Obviously you should be able to create your own examples
- Imagine an internship interview

How to pass the exam 2

Advanced questions

- There will be source code samples to audit (and fix)
- It won't be 100% clean
- It won't be exactly like "standard epita code"
- If it's different it's not necessarily wrong
- Beware of wrong assumptions
- The security issues to fix will be nasty ones, always

How to pass the exam 3

- I don't care if you write in English or French
- ... but only write in proper English
- beware of attendance, there might be a pop-quizz

The developement cycle

- Classical shops write specs
- ... and have devs who implement them
- ... and db experts who write databases
- ... and system engineers who work on deployment
- ... and testers
- ... and security auditors
- **This does not work**

Why not

Specialization is for insects

Robert A. Heinlein

Why not

- if the auditors find a bug
- ... sometimes it's because the design is wrong
- auditors can't catch it all
- ... so devs must know about good practices

Testers vs Devs

- You don't want to pit testers vs devs
- a good tester is invaluable
- ... document and fix bugs

Testers vs Devs

- You don't want to pit testers vs devs
- a good tester is invaluable
- ... document and fix bugs

Database experts

- A lot of "database experts" don't even know about SQL injection
- Don't get me started on Php

Release!

- so you get to "ship a release" (end software product: V5.0)
- ... that's not always the end
- are you the vendor ?
- ... not the case for Unix distros

Branch and Support

- Before release: branched for that version (say: 5.0 beta)
- Resources devoted to 5.0
- After release: keep on current
- Residual resources devoted to 5.0

A bug

- You got to fix it... and possibly ship 5.1
- ... but wait that means testing
- what about branch 4 ? and 3 ?
- End of life for a product (EOL)
- Extended support release (ESR)

Security bugs

- A bug is not a security hole
- Most attacks are based on a **series of bugs**
- We want **defense in depth**
- Fixing one bug stops the attack!
- An attack is also called an **exploit**
- Software has **vulnerabilities**

Who done it

- Developer found the bug
- External user found the bug
- ... recognized as a security issue ?
- ... External user nice or not ?

Who done it

- Proving it's a security issue ?
- Being pro-active about it
- Fixing it without letting the bad guys know

Vendors and Open-Source and...

- Was reported on bugtraq
- ... multiple times
- CVE: common vulnerabilities and exposures

Timely releases

- Don't release on friday
- Account for vendors
- Have a "secure" channel for bugs
- Worst case scenario: **zero days**

Misconception

- It's too complicated it won't be exploitable
- The IE5 url overflow
- Because it's encoded as 16 bit characters

Not a security issue

- Software components get reused all the time
- Plan to be successful

OpenSource vs Closed Source

- Closed Source is not more secure
- Lots of people know how to reverse-engineer
- The "sweep under the carpet" effect
- Example: Crafting exploits from Windows Update

I don't do bugs

- It takes one bug
- Everything is exploitable eventually

Buffer overflow

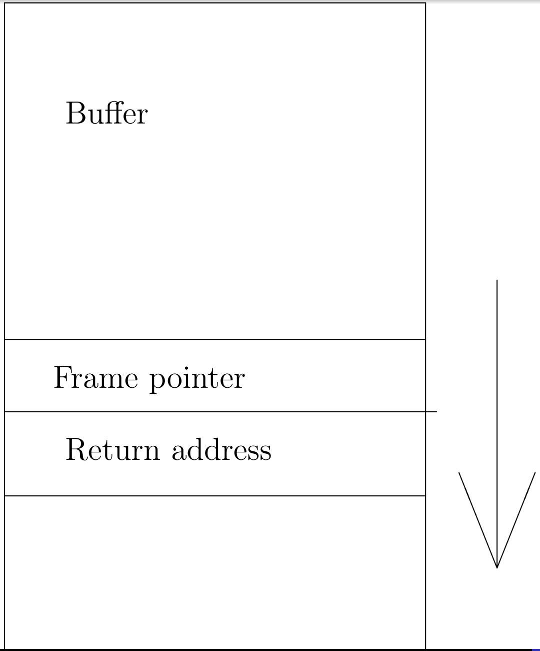
What's this

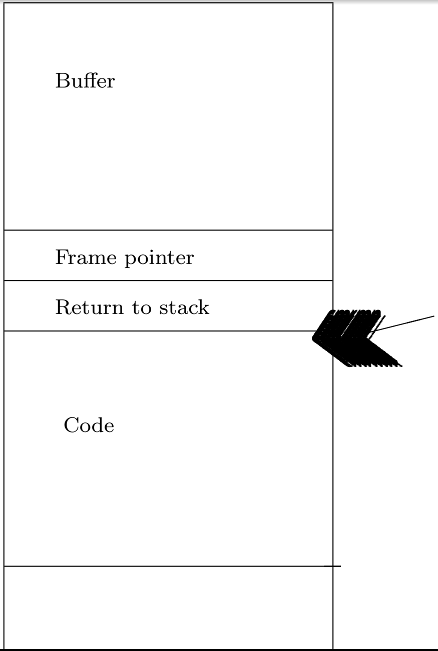
```
void f() {
        char buffer[70];

        ...

        gets(buffer); // problematic line

}
```


Buffer
Frame pointer
Return address


Buffer
Frame pointer
Return to stack
Code

How to avoid that

- Don't do bugs
- Know your APIs
- Prefer secure idioms
- Make code simple

Mitigation: canaries

- Function prolog inserts random data on the stack
- Function epilog checks the data didn't change

Example

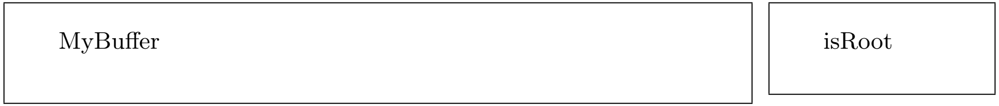
```
f:                                      # @f
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $80, %rsp
        movq    __guard_local(%rip), %rax
        movq    %rax, -8(%rbp)
        leaq    -80(%rbp), %rdi
        xorl    %eax, %eax
        callq   gets@PLT
        movq    __guard_local(%rip), %rax
        cmpq    -8(%rbp), %rax
        jne     .LBB0_2
        addq    $80, %rsp
        popq    %rbp
        retq
        leaq    .LSSH(%rip), %rdi
        callq   __stack_smash_handler@PLT
```

Less obvious ones

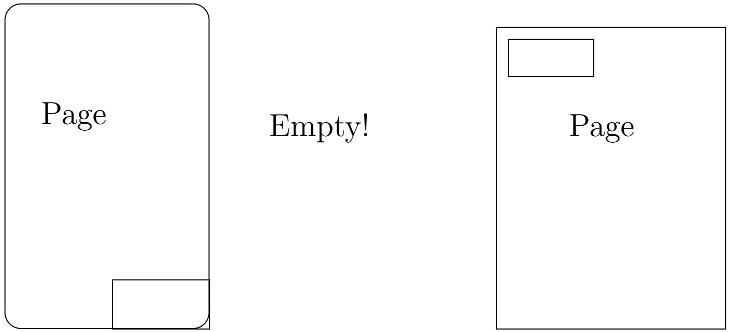
```
char *
make_filename(const char *dir, const char *file)
{
        char *r = emalloc(strlen(dir) + strlen(file) + 1);
        strcpy(r, dir);
        strcat(r, "/");
        strcat(r, file);
        return r;
}
```

But it's not on the stack

- If you use linked lists, you killed the next pointer
- If you have power-of-two allocators, you killed the next allocation


MyBuffer
isRoot

Mitigation: guard pages



Better APIs

- don't use strcpy, strcat
- don't use strncpy, strncat

```
struct utmp {
        char    ut_line[UT_LINESIZE];
        char    ut_name[UT_NAMESIZE];
        char    ut_host[UT_HOSTSIZE];
        time_t  ut_time;
};
```

- prefer strlcpy, strlcat

Example

```
char *dir, *file, pname[PATH_MAX];
...
if (strlcpy(pname, dir, sizeof(pname)) >= sizeof(pname))
        goto toolong;
if (strlcat(pname, file, sizeof(pname)) >= sizeof(pname))
        goto toolong;
```

The Drepper fallacy

- "But I don't write wrong code"
- The reason for slow adoption of strlcpy

Better APIs 2

- prefer snprintf to sprintf
- use asprintf if you must

Size everywhere

- you want to help auditors
- if a size isn't obvious, make it part of the API

Sturgeon's law

90% of all software is

- crap
- unimportant to optimize
- bogus
- copied-and-pasted
- imperfect

The Drepper fallacy 2

- You can't fix everything
- ... therefore don't fix anything
- "Low-hanging fruits"

Compilers help

```
char *
make_filename(const char *file, const char *dir)
{
        char buffer[MAXBUF];
        snprintf(buffer, sizeof buffer, "%s/%s", file, dir);
        return buffer;
}
```

But not always

```
char *
make_filename(const char *file, const char *dir)
{
        char *buffer = emalloc(MAXBUF);
        snprintf(buffer, sizeof buffer, "%s/%s", file, dir);
        return buffer;
}
```

ETOOMANYWARNINGS

```
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c blocks
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c huffma
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c crctab
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c randta
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c compre
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c decomp
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
cc -O2 -pipe   -Wall -Winline -fomit-frame-pointer -fno-strength-reduce -c bzlib.
cc: warning: optimization flag '-fno-strength-reduce' is not supported [-Wignored-
```

```
int *
alloc_array(int n)
{
        int *t = emalloc(n * sizeof(int));
        return t;
}

int *
read_array()
{
        int s = 0;
        scanf("%d", &s);
        if (s == 0)
                exit(1);
        int *t = alloc_array(s);
        for (int i = 0; i != s; i++)
                scanf("%d", t[i]);
        return t;
}
```