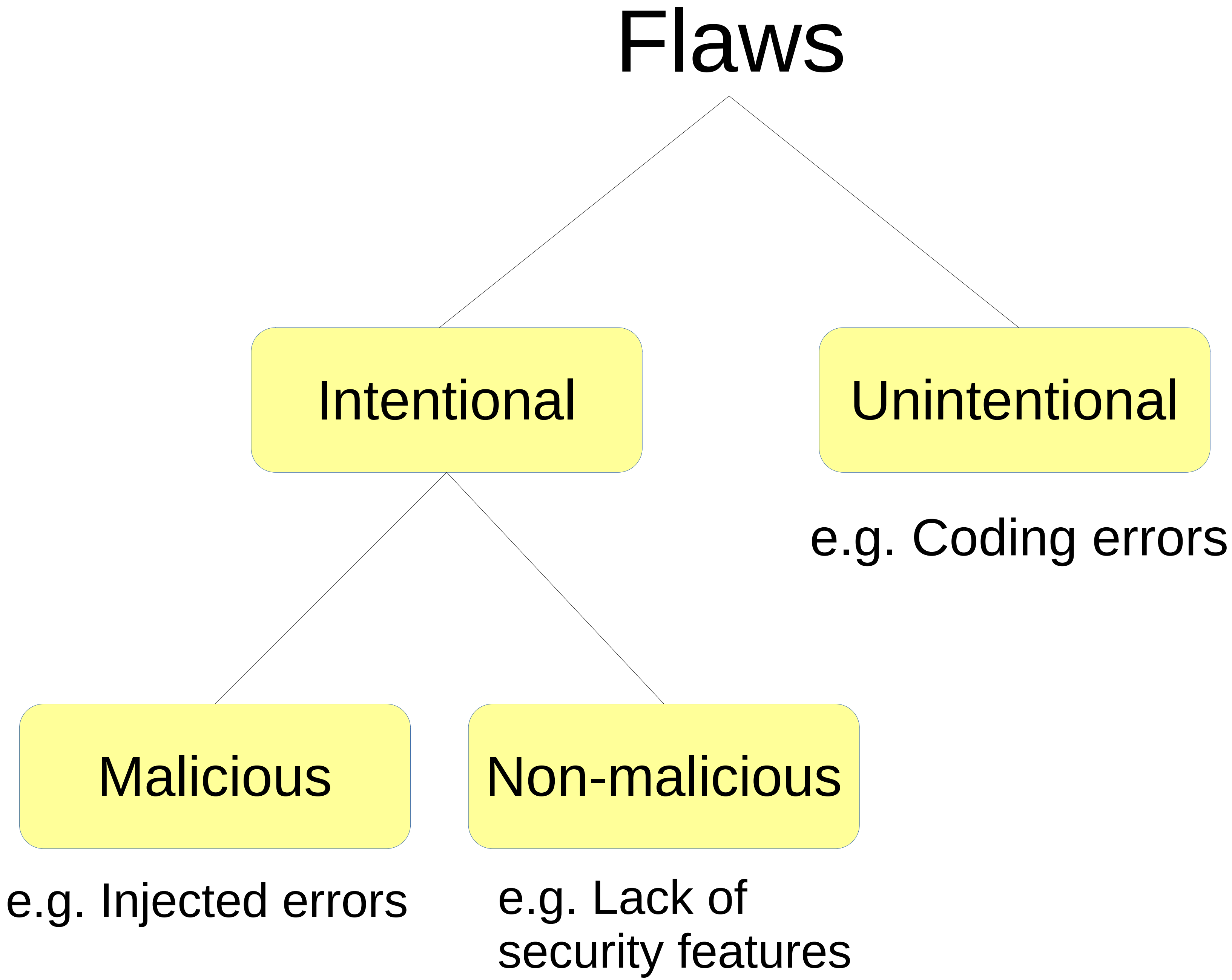
Module 2

# Software Security

# Software errors can kill a project

Mars Polar Lander (1999) – crashed on Mars

Sensors were programmed incorrectly and shut off engine;
not caught in testing


Flaws
Intentional
Unintentional
e.g. Coding errors
Malicious
Non-malicious
e.g. Injected errors
e.g. Lack of security features

# Unintentional Flaws

We will discuss two types of unintentional flaws:

<u>Local application flaws</u>

- Buffer overread, buffer overflow, TOCTTOU

<u>Web application flaws</u>

- XSS, XSRF, SQL Injection

# Buffer overread

Your own memory may look like this:

wake up; have breakfast; need to buy milk; turn off the lights; go to class; that man has a strange shirt; fall asleep; wake up

A web server’s memory may look like this:

Bob requests main page; Atta wants reply “Cat”; Li sets password to “sup3rsekr1t”; Kate wants image “derpy_cat”; Poe sets secret key; …

# Buffer overread

Bob requests main page; Atta wants reply “Cat”; Li sets password to “sup3rsekr1t”; Kate wants image “derpy_cat”; Poe sets secret key; ...

Memory

Please reply “Cat” (3 letters).

Cat

Please reply “Cat” (5 letters).

Cat”;

# Buffer overread

Bob requests main page; Atta wants reply “Cat”; Li sets password to “sup3rsekr1t”; Kate wants image “derpy_cat”; Poe sets secret key; ...

Memory

Please reply “Cat” (100 letters).

Cat”; Li sets password to “sup3rsekr1t”; Kate wants image “derpy_cat”; Poe sets secret key; ...

# Buffer overread

Heartbleed (2015)

*memcpy(bp, pl, payload);*

Supposed to be the size of that array, but user declares this

Returned to client

Points to an array

# Buffer overflow

Also “stack smashing”, “buffer overrun”

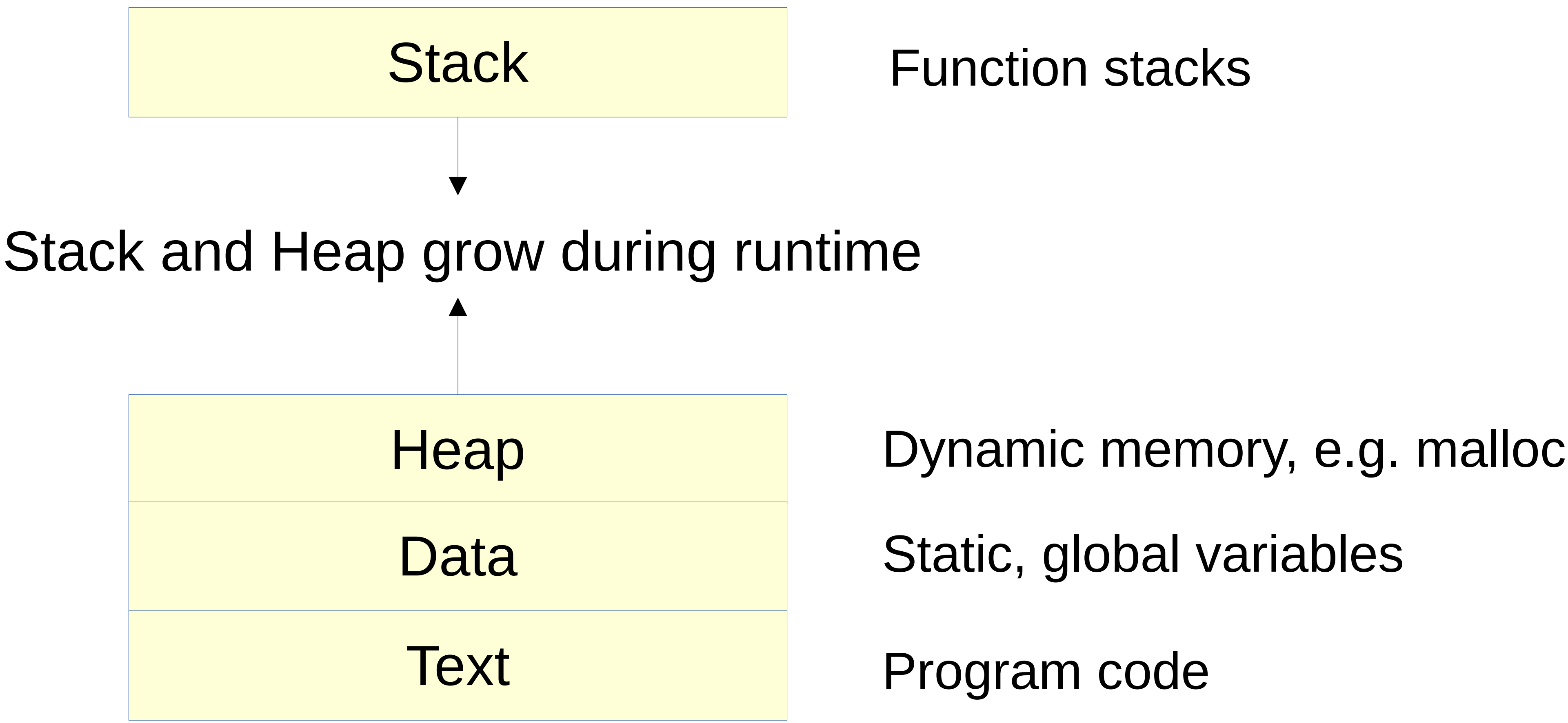
```
void input_username(...) {
   char username[16];
   printf(“Enter username:”);
   gets(username);
   ...
}
```

strcpy, gets, fgets, etc. can write more data than the target size

What if you could write directly into memory?

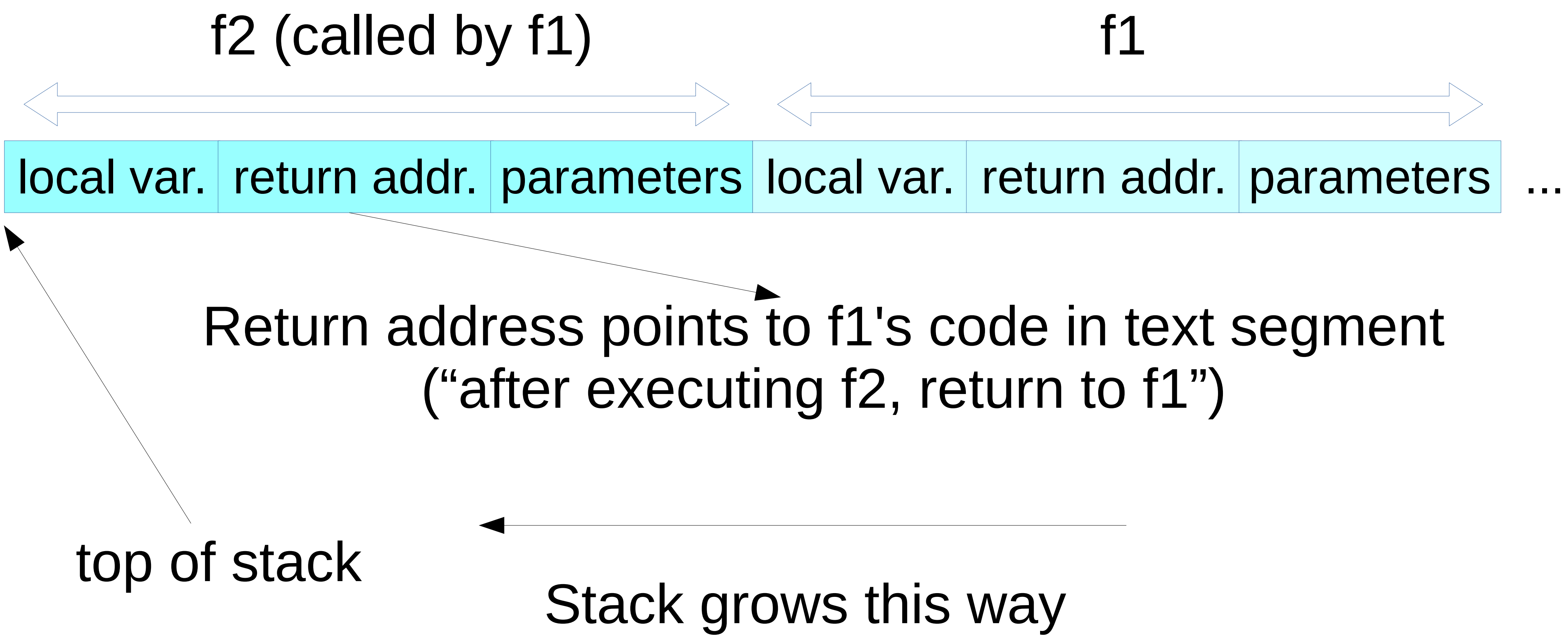
# Buffer overflow

Memory of C program process:



# Buffer overflow

A simplified function stack



# Buffer overflow

## A simplified function stack

```
void input_username(...) {
   char username[16];
   printf("Enter username:");
   gets(username);
   ...
}
```

gets does not check bounds!

[                    ] [7FA2]

| ... | username[16] | return addr. | Parameters |
|---|---|---|---|

(return address normally points to text segment, not stack)

# Buffer overflow

## A simplified function stack

```
void input_username(...) {
   char username[16];
   printf("Enter username:");
   gets(username);
   ...
}
```

If user types 22 A's...

```
[AAAAAAAAAAAAAAAA]  [4141]  [AAAA]
```

| | | | |
|---|---|---|---|
| ... | username[16] | return addr. | Parameters |

Upon function termination, return to "AAAA" (segfault)

But the attacker can be smarter

# Buffer overflow

A simplified function stack

`[execute evil code;]`

| another_buffer |
|---|

Malicious shell code can be written in the stack too

(shell code is assembly code that spawns a shell)

`[AAAAAAAAAAAAAAAA] [E4FF]`

| ... | username[16] | return addr. | Parameters |
|---|---|---|---|

This will cause the shell code to be executed!

# Buffer overflow

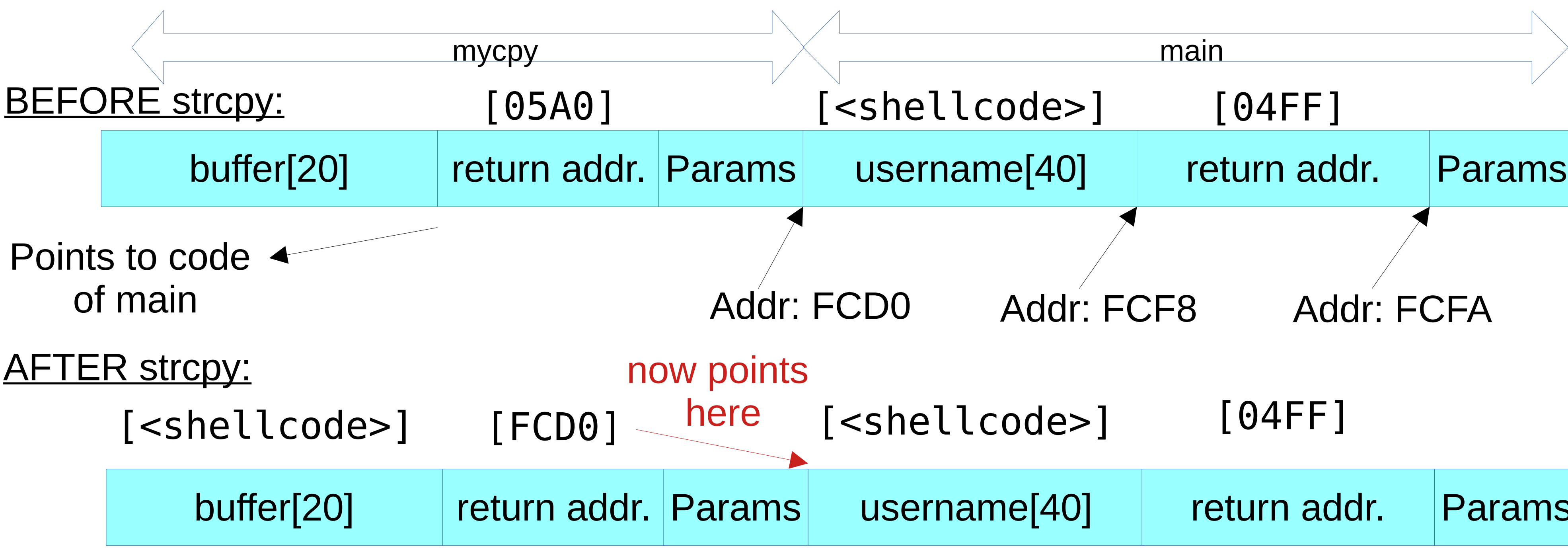
## Example

breakpoint

```
int mycpy(char* username) {
   char buffer[20];
   strcpy(buffer, username);
   return 0;}
void main(int argc, char** argv) {
   char username[100];
   fgets(username, 100, stdin);
   mycpy(username);}
```

Attacker inputs username:

<shellcode>FCD0



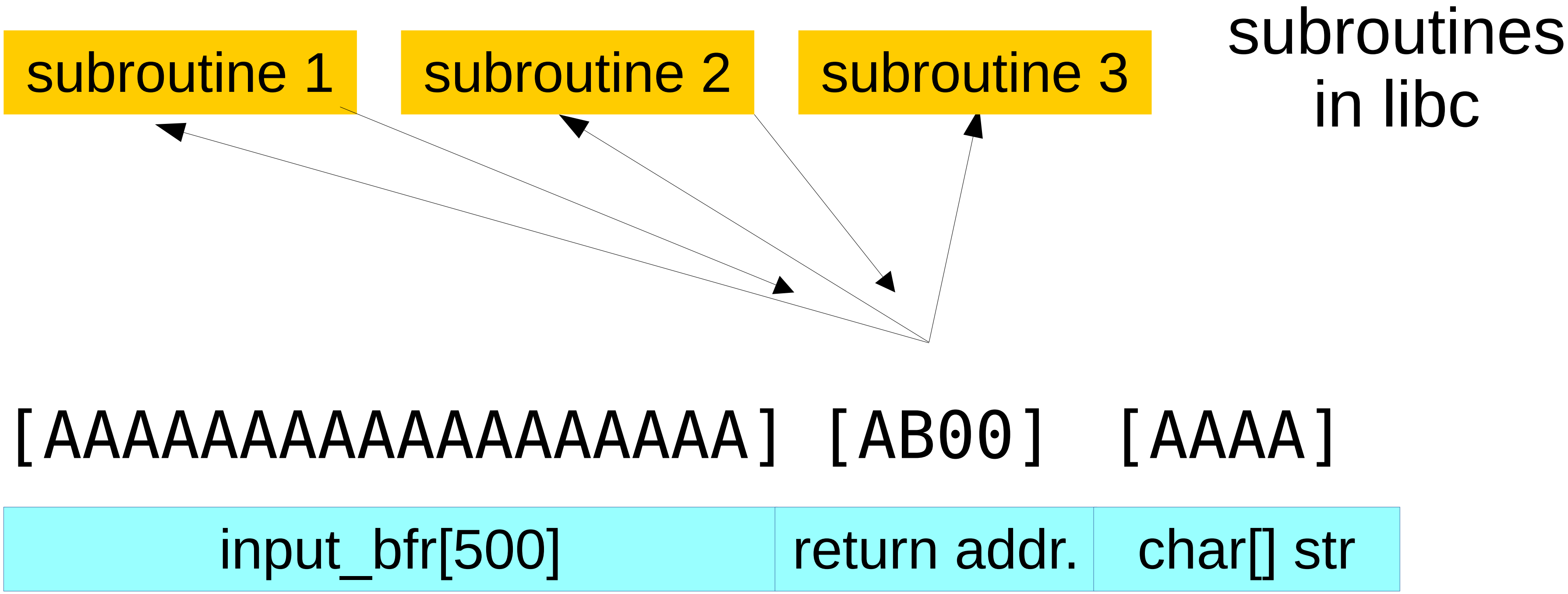
# Buffer overflow

## Defenses

- Never execute code on stack
  - W^X (write XOR execute), NX, or DEP
- Randomize stack
  - Address Space Layout Randomization
- Detect overflow
  - Canaries
- Don't use C

# Buffer overflow

## Return-Oriented Programming

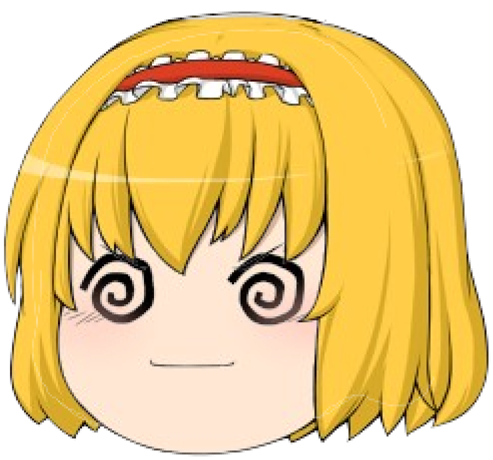



How to defeat W^X

# Buffer overflow

Majority of known software flaws are buffer overflows

- Very common (why?)
- Very powerful – gives root access
- Not much harder to exploit than to detect



# Integer overflow

- Integers are often stored in 32-bit
  - Sometimes 16-bit with specific systems
- When exceeding the maximum, the result is an error
  - Often, wrapping back to the lowest/negative number
- It is surprisingly easy to exceed the maximum!
  - e.g. What is 2^31 milliseconds?
  - e.g. Any multipliers that can be applied

# Format string vulnerability

- The following prints today's lucky number:

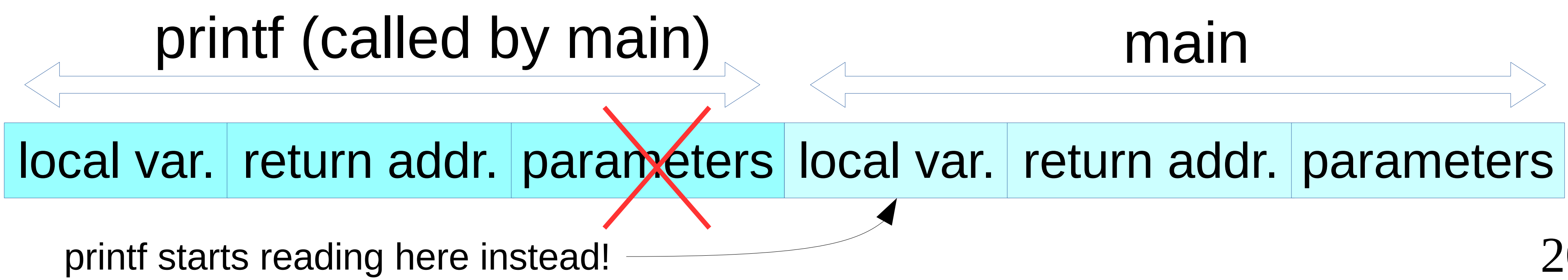
```
printf("Today's lucky number is %d", 18);
```

- What about the following?

```
printf("Today's lucky number is %d");
```

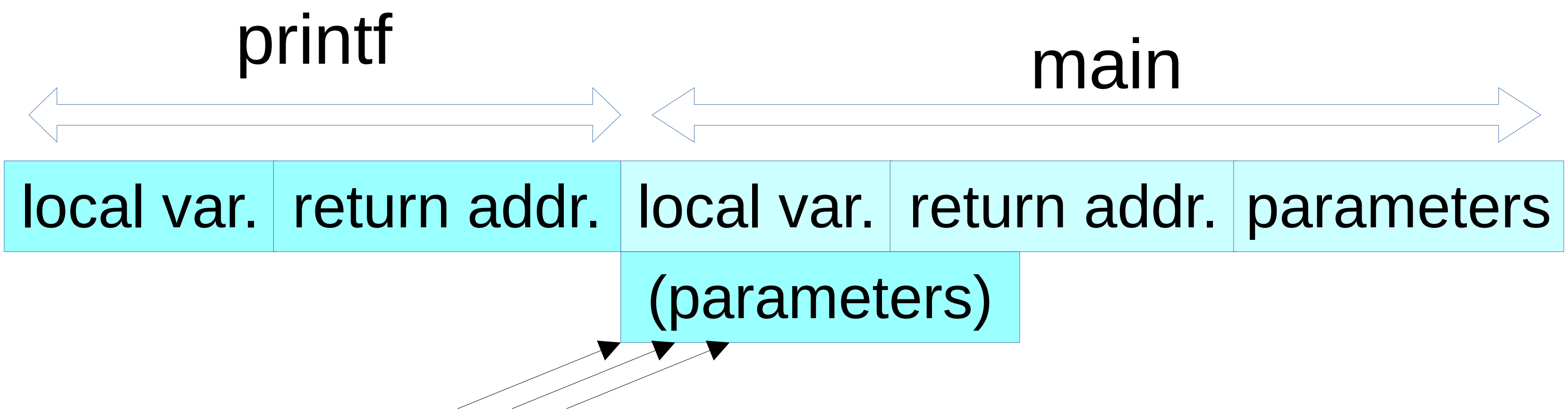
- What if the user has control over this string?

```
char uname[250];
fgets(uname, 250, stdin);
printf("Your username is: ");
printf(uname);
```



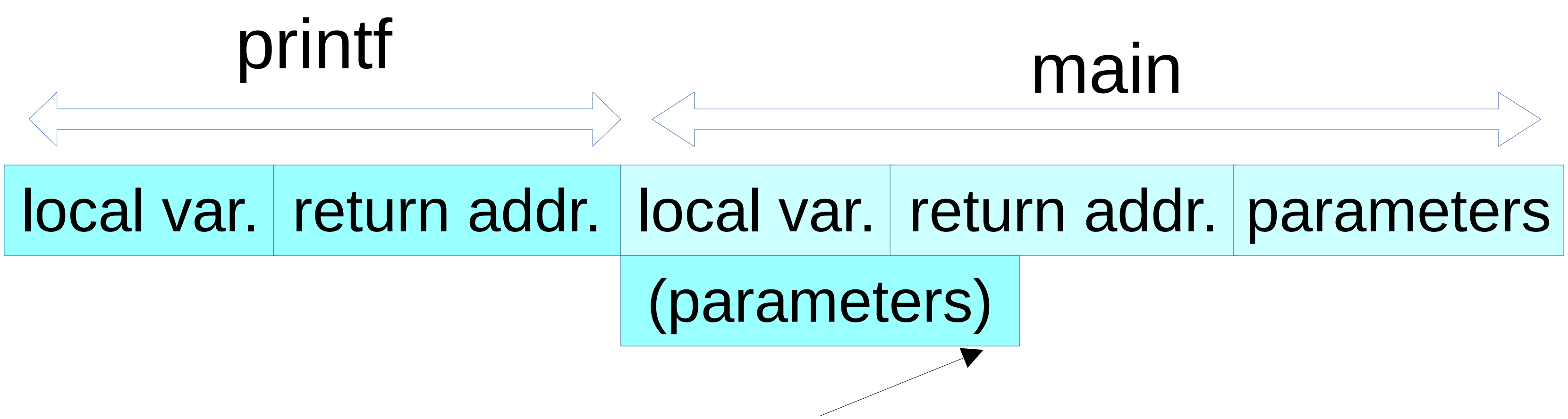
# Format string vulnerability

```
username = “%d %d %d”; printf(username);
```




Prints out the next 3x4-bytes as integers

```
username = “%18$d”; printf(username);
```




Prints out bytes 72 to 76 after the end of printf return addr

# Format string vulnerability

- %n: Counts the number of bytes written so far, writes it to the given variable

```
int len;
printf("This string length is%n...? ", &len);
printf("%d", len);
```

```
> This string length is...? 21
```

- What if len was not provided?
- If the user controls a format string, they can put a clever combination of %d and %n there to write whatever they want to an address!

# TOCTTOU

*A type of "race condition"*

- "Time of Check To Time of Use"
- Check: Should the user have privilege?
  - Access control, check ownership, etc.

What if something changes?

- Use: Do something for the privileged user
  - Read file, write to file, change permissions

# TOCTTOU

*passwd* example (pseudocode)

I want to change root password, but I am not root

> passwd new_password

passwd code:

```
check_access(password_file, user);

update_file(password_file, new_password);
```

What if you can change password_file in-between?

# TOCTTOU

*passwd* example (pseudocode)

> passwd new_password

passwd code:

attacker: set password_file to point to user_password

```
    check_access(password_file, user);
```

attacker: set password_file to point to root_password

```
    update_file(password_file, new_password);
```

(Attacker actions are on the OS, not part of the code)

# TOCTTOU

Attacker can increase chance of success by:

- Opening a file in a deep directory
- Opening a file in a remote network location
- Simply timing the attack well or keep retrying

<u>Prevention:</u>

- Locking the object under use
- Checking something that is immutable

# Cross-site Scripting (XSS)

Enter the following in your profile/biography:

```
</script> <script>
Please log in again!
<a href="http://attackerserver.com">
    <input type="text" placeholder="Enter
Username" name="uname" required>
    <input type="password" placeholder="Enter
Password" name="psw" required>
    <button type="submit">Login</button>
</a> </script>
```

If this works, that page has an XSS!

# Cross-site Scripting (XSS)

*XSS vulnerabilities occur when users can write code onto a web page*

- <u>Persistent XSS vulnerability</u>
  - User changes content of a page persistently
  - e.g. social media profile page
- <u>Reflected XSS vulnerability</u>
  - Malicious link that executes code as if it was part of the page's content
  - Person who clicks link doesn't know it's evil

www.bad-bank.com/login.php?username=<script>dobadthings</script>

  - e.g. Steal cookies, make fake login window, send messages to other users

# Cross-site Request Forgery (XSRF)

*In XSRF, a malicious forged link causes the user to make a request that harms herself*

<u>Example:</u>

If the victim is currently logged into bad-bank.com:

www.bad-bank.com/give_money.php?amount=10000&target=attacker

*Difference with reflected XSS:*

- *XSRF is itself a legitimate request for the website, though the website should not allow such a link to work*
- *Reflected XSS puts arbitrary code in the link, running a script that can be completely unrelated to the website*

# SQL injection

Poor SQL code with parsing vulnerability:

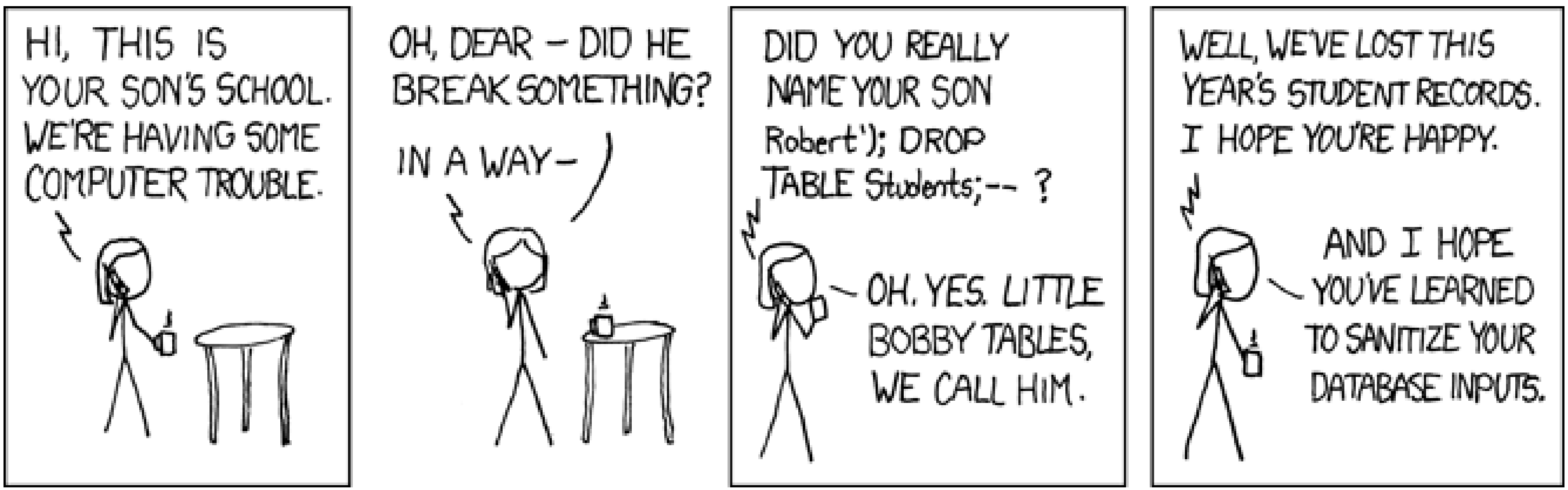
s = “SELECT uid FROM utable WHERE username =‘” + input_uname + “’AND password =’” + input_password + “’”

If uid is non-empty, then login is successful.

User inputs input_uname as:

‘ OR ‘1’ = ‘1'--

# SQL injection



# Parsing vulnerabilties

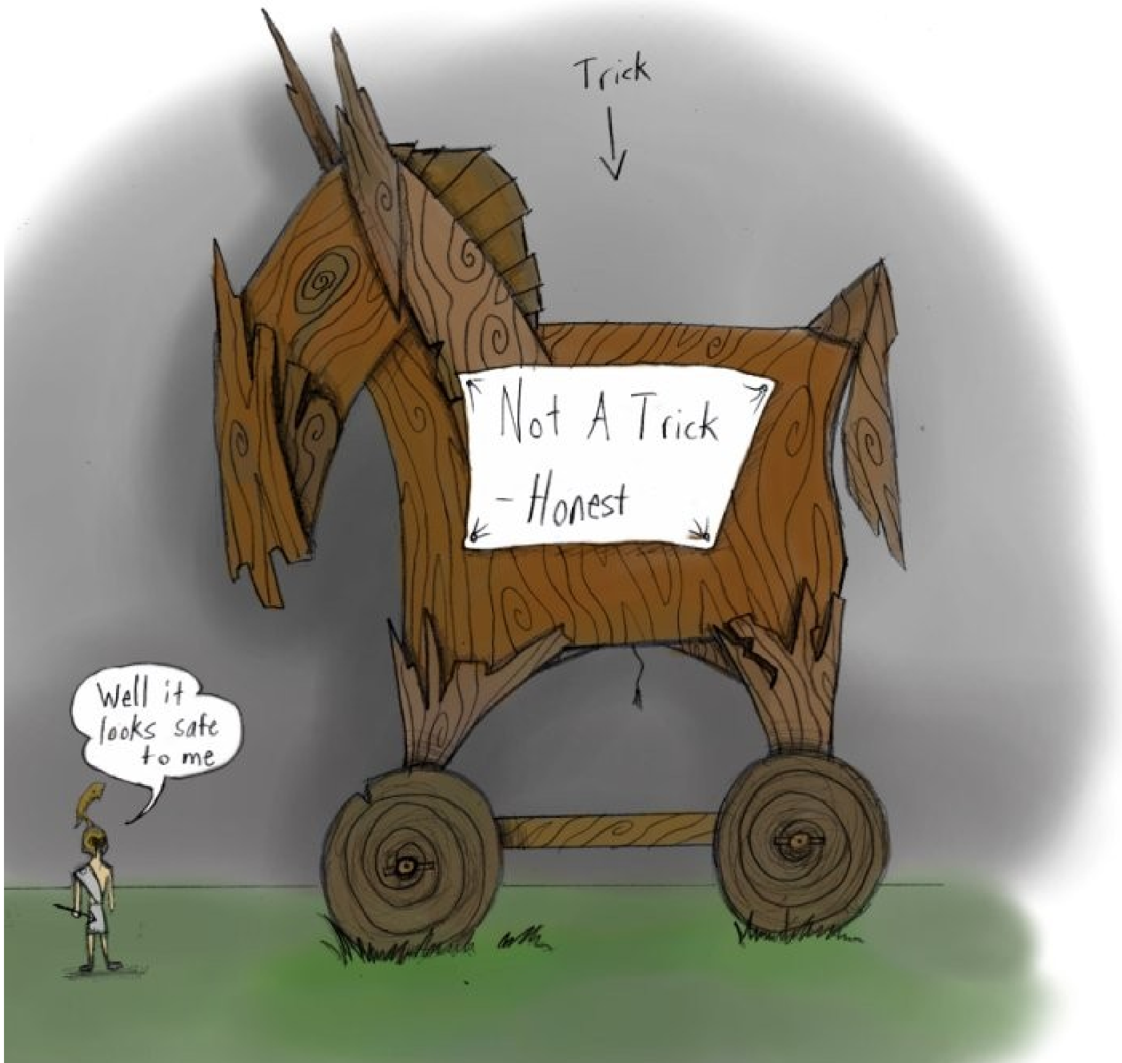
Characters and numbers may be parsed incorrectly:

- rlogin -l -froot attack allowed remote login as root
  - Target computer receives “login -f root”
- Canonicalization: Many ways to represent the same string; attacker chooses a way to avoid blocking/detection. Examples:
  - http://2130706433/
  - A trojan downloading a file with .exe%20 to avoid exe files being blocked
  - System allows access to /data/user/taowang, so you access data/user/taowang/../../../system/

# Classifying malware

- Malware consists of a *spreading mechanism* and a *payload*
- We can classify by method of spread
  - AKA infection vector
  - How does it get on your computer?
- Or by effect on system (payload)
  - What does it do to your computer?

# Trojan



# Trojan

*"Given a choice between dancing pigs and security, users will pick dancing pigs every time."*

–Gary McGraw and Edward Felten, "Securing Java"

# Trojan

A trojan is a piece of malware that spreads by tricking the user into activating/clicking it

- Packaged with useful software
- Looks like useful software (e.g. Android re-packaging)
- Scareware
- Spear phishing

*People often represent the weakest link in the security chain.*

— Bruce Schneier

# Trojan

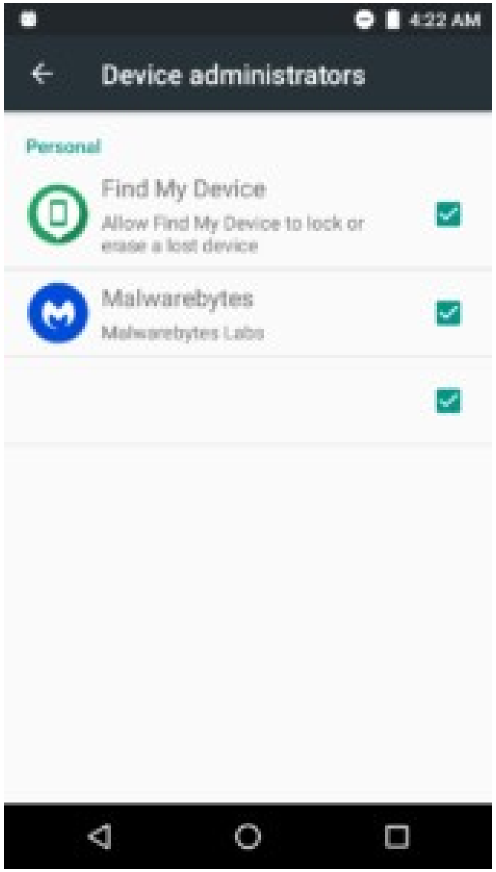
*ILOVEYOU (2000, Windows):*

- Malware in e-mail attachment: “LOVE-LETTER-FOR-YOU.txt.vbs”
- Destroys files on target system through replication
- Reads mailing list, sends files to them
- Downloads another trojan “WIN-BUGSFIX.EXE”
- Very easy to reprogram

# Trojan


Conficker Worm’s interface illusion

# Trojan




MobiDash’s interface illusion

# Removable media

*ByteBandit* (1987, Amiga):

- Spreads with an infected floppy disk
- Resides in memory, even after reboot
- Infects all inserted floppy disks
- After causing 6 infections, black screen!

# Network

Malware that spreads through packets requires *no user action*

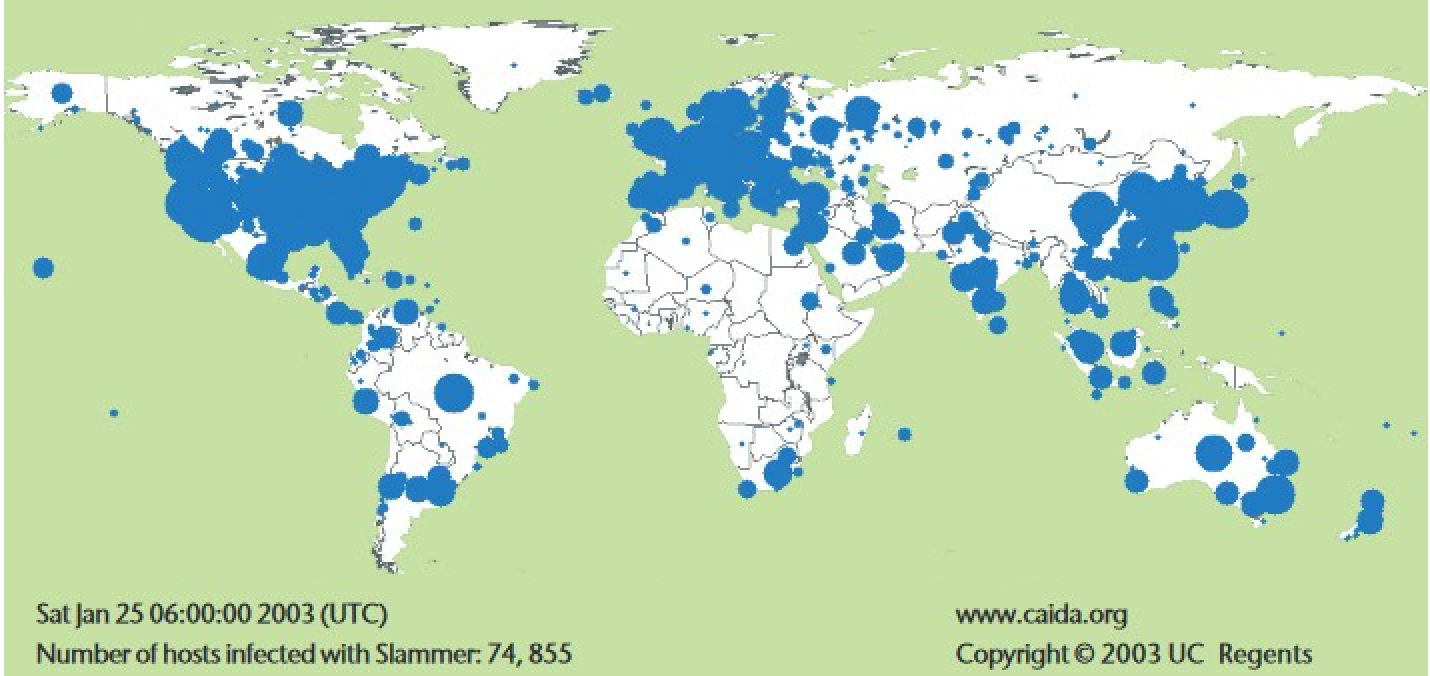
- Infects network-facing background programs (daemons) to spread
- Can be very fast – infection and spread can be automatic, exponential
- Malware spreading explosively can cause worldwide internet outage, and are called “worms”

# Network

Slammer Worm (2003, Microsoft SQL Server):

- Exploits SQL Server buffer overflow using a packet
- Patch had existed after Blackhat warning
- Generate random addresses, sends itself by UDP
- Infection doubled every 8.5 seconds, reached 90% of all vulnerable systems in 10 minutes
- “Warhol worm” - Andy Warhol “In the future, everyone will be world-famous for 15 minutes”
- No payload

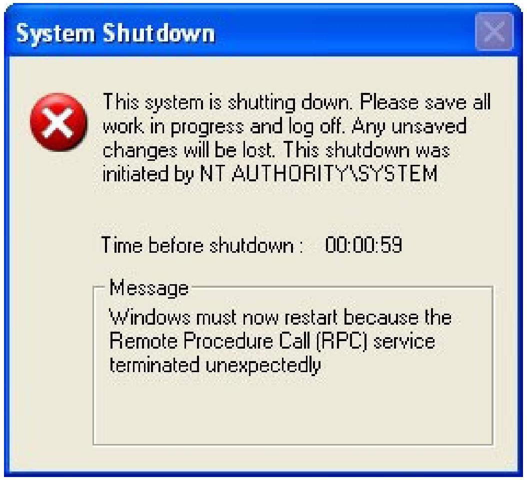
# Network


Sat Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Slammer: 74, 855
www.caida.org
Copyright © 2003 UC Regents

# Network

Blaster Worm (2003, Windows):

- Exploits RPC buffer overflow
- Payload: DDoS windows update site
- Earlier warnings, patches were not installed
- (Unintentionally) shut down computers
- Welchia is a “helpful” worm that removes Blaster and force-installs patches



# Planted malware

Installed intentionally by an attacker who has (temporary) control over the system:

- Employee
- Espionage
- From other malware

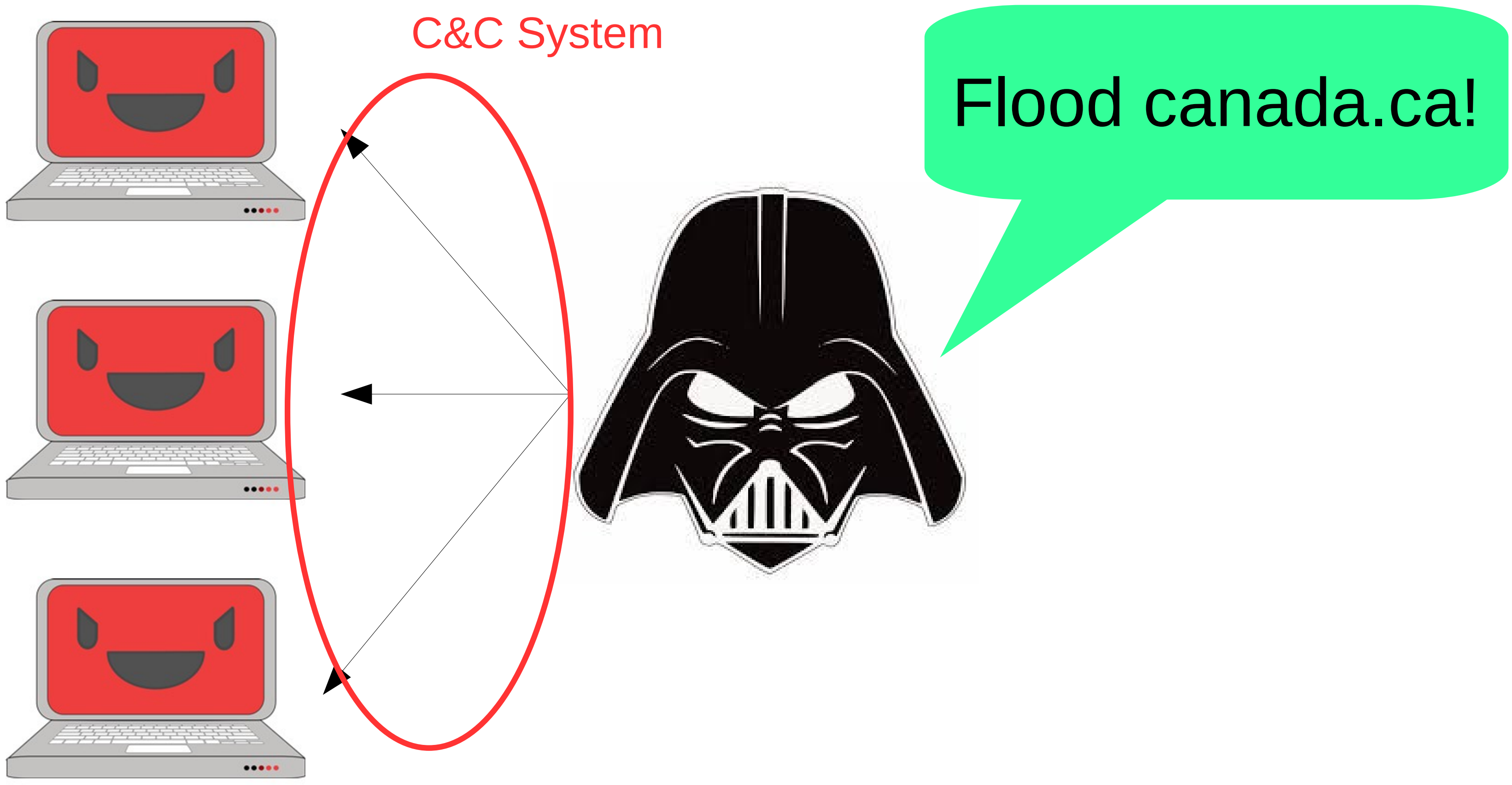
Sometimes the payload is a logic bomb: Malicious code set off by specific conditions

- After some amount of time
- If an employee is fired

# Classifying malware

- Malware consists of a *spreading mechanism* and a *payload*
- We can classify by method of spread
  - AKA infection vector
  - How does it get on your computer?
- Or by effect on system (payload)
  - What does it do to your computer?

# Botnet




Computers owned by
different users

# Botnet

- Consists of three components:
  - A Master
  - A large number of infected devices (“bots”)
  - A Command and Control structure
- Useful for:
  - Hiding attack source/identity
  - Sybil attacks
  - Malware spreading
  - Spam

# Backdoors

- Allows unexpected access to system
- Could be created on system because:
  - Left for testing (intentional non-malicious flaw)
  - Installed by malware
  - Demanded by law

# Rootkits

- A rootkit is a piece of malware for maintaining command & control over a target system (root)
- It changes the behavior of system functionalities to hide itself/some other malware
- Hard to remove
- User rootkits can change files, programs, libraries, etc.
- Kernel rootkits can change system calls

# Rootkits

Sony XCP (2005)

- **Rootkit** by Sony
- Garbles write-output of XCP disk
- Hides all files and folders starting with “sys”
- Eventually, Sony released an uninstaller due to pressure

# Zip bombs, compiler bombs

- Destructive payloads usually used in the context of a trojan
- Zip bombs: Unzipping the bomb creates a very large file
- Compiler bombs: Compiling the bomb creates a very large file
- Besides destruction, can be used to break certain scans

# Spyware


zuck
Instagram HQ
500M likes
zuck Thanks to everyone in our community for helping us reach this milestone!
kevin @zuck

# Spyware

- Secretly collects data about the user

Pegasus (2016):

- Spyware for iOS and Android
- Developed by software company NSO Group
- Reads text messages, traces the phone, can enable microphone and camera, etc.
- Uses three zero-days, including Use After Free

# Trackers (Spyware)

- **Cookies** store information about you
- Third-party cookies allow your actions on site A to be collected and sent to site B (blocked on some browsers)
- Web beacons on websites make a request for you to a third-party (ad) server, which can also automatically send your cookies for that server
- Beacons in multiple sites often link to the same ad server

<img src="http://ad.doubleclick.net/ad/pixel.quicken/NEW" width=1 height=1 border=0>

# Keylogging

Several kinds of keyloggers:

- Application-specific keyloggers
- Software keyloggers
- Hardware keyloggers

Each can be installed covertly

Some keylogging malware steals your credentials (e.g. “bankers”)

# Ransomware


CryptoLocker: Estimated $3 million extorted

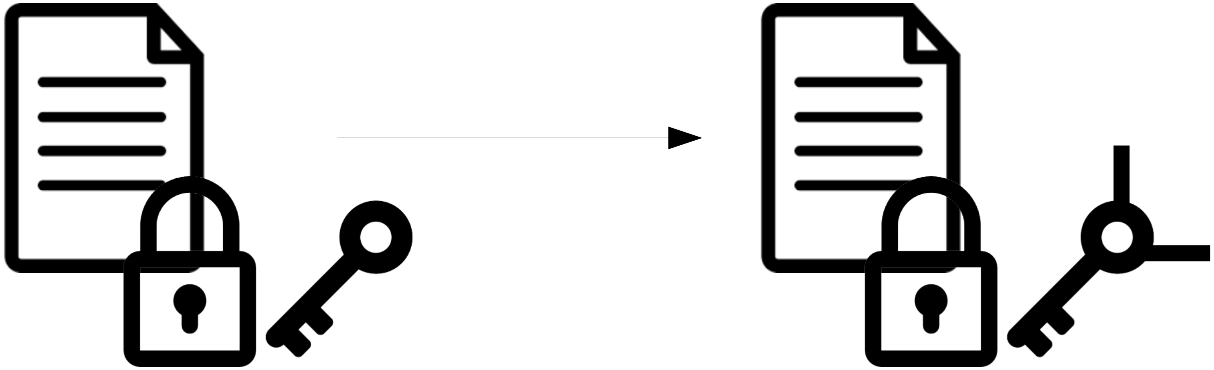
# Ransomware

- General technique: encrypt disk, then demand ransom to decrypt it
- Disk is encrypted using public key, private key is on attacker’s own server
- Attached storage media will also be encrypted
- Little recourse once files are encrypted
- A number of attacks fail to release keys

# Stealth techniques

To avoid detection:

- Polymorphic code
- Hide in memory, disguise file patterns
- Interrupt scanning techniques



*Code polymorphism*

# Advanced Persistent Threats

- Combination of multiple infection vectors and spreading strategies
- Focused, long-duration attack
- Achieves political/industiral goal

# Advanced Persistent Threats

Stuxnet (2011)

- Spreads by network and USB
- Uses four zero-day attacks
- Does nothing in almost any machine
- But it wrecks a specific type of
  Iranian nuclear reactor centrifuge controller
- Speculated to be government-sponsored

# Advanced Persistent Threats

# Advanced Persistent Threats

Flame (2012)

- Spyware: records keystrokes, camera, screen, sends to remote server
- Behavior determined by your antivirus
- Uses a fake certificate obtained by attacking a Microsoft server's weak cryptorgaphy
- Very large (20MB)
- Attempted to erase itself when discovered

# Covert Channels

Covert channels are resources (not intended for communication) that are used by an attacker to communicate information in a monitored environment *without alerting the victim*

- To retrieve stolen data
- To receive commands
- To update malware

Examples: TCP initial sequence number, size of packets, timing, port knocking

# Side Channels

Side channels leak information in unintended ways

- Power analysis
- Timing analysis
- EM wave analysis
- Acoustic analysis

Defenses: air gap, Faraday cage, etc.

# Side Channels

Spectre (2017)

Side channel attack on microprocessors

1) CPU branch prediction can be trained by attacker-controlled data

2) A branch mis-prediction can read process memory and affect processor cache

3) Processor cache contents can be exposed using timing attacks

=> This can potentially leak any process memory

# Side Channels

Spectre (2017)

Example (Kocher et al.):

```
1    if (x < array1_size)
2        y = array2[array1[x] * 4096];
```

- The attacker can make the CPU “expect” that the check in line 1 will pass, and predictively execute line 2
- If the CPU runs line 2 on x larger than array1_size, it is a buffer overread
- This affects the processor cache and what it reads can be guessed with a timing attack

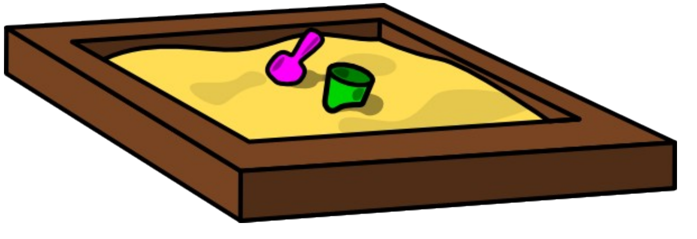
# Defensive strategy

How do we defend against software flaws?

- Blocking access from attackers: Scanning, ...
- Writing good code: code review, change management, testing
- Fixing bad code: code analysis, patching

# Malware scanning

- Signature-based:
  - Scans for virus “signatures”
  - Scans memory, registry, program code
- Behavior-based (“heuristics”):
  - Detects system irregularities
  - May have false positives
- Sandboxing
  - Run potentially malicious code in controlled environment
  - Often used with honeypots



# Code analysis

*Look for vulnerabilities/bugs in code*

- Static code analysis

  *Examine code for vulnerabilities*

- Dynamic code analysis

  *Test code by running it on input*

- Formal verification

  *Prove that code follows a specification*

# Code analysis

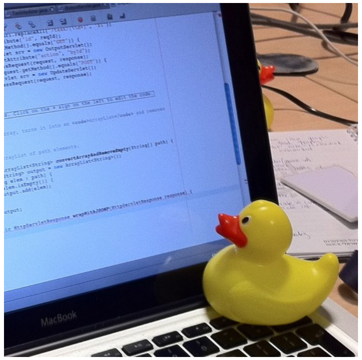
## sel4: Formally verified OS

- Contains 8,700 lines of C, 600 lines of assembly
- Proof of correctness: 200,000 lines of code
- Can have “unintended features”
- Bugs that are not in the specification could still exist (e.g. timing attacks)

# Software testing

- Unit testing (test small units one at a time)
- Integration testing (test integration of units)
- Fuzz testing (test with random input)
- Black-box testing (test unknown system)
- White-box testing (test known system)
- Regression testing (test if update causes bugs)

# Code review



- Formal inspection
  - Programmer explains code to panel
- Pair programming
  - Programmer explains code to an observer
- Rubber duck programming
  - Programmer explains code to themselves
- Change management
  - System for recording and managing code changes

# Patching

# Error 503 Service Unavailable

Service Unavailable

### Guru Meditation:

XID: 1995750753

*Varnish*

Having a good error message helps!

# Patching

Several unresolved problems:

- Vulnerable users don’t install patches
- Patches cause further issues
- Patches don’t resolve underlying issues

Microsoft’s “Patch Tuesday” forces patches to be installed and makes it easier for system administrators to fix issues

# Summary

Unintentional flaws

- Buffer overread, buffer overflow, TOCTTOU
- XSS, XSRF
- Exploited by malware: viruses, worms, trojans

Intentional malicious flaws

- Planted malware, rootkits

Intentional non-malicious flaws

- Covert channels, side channels

Defensive strategy

- Scanning, code analysis, testing, review, patching