650.445/600.454: Practical Cryptographic Systems

Software Vulnerabilities

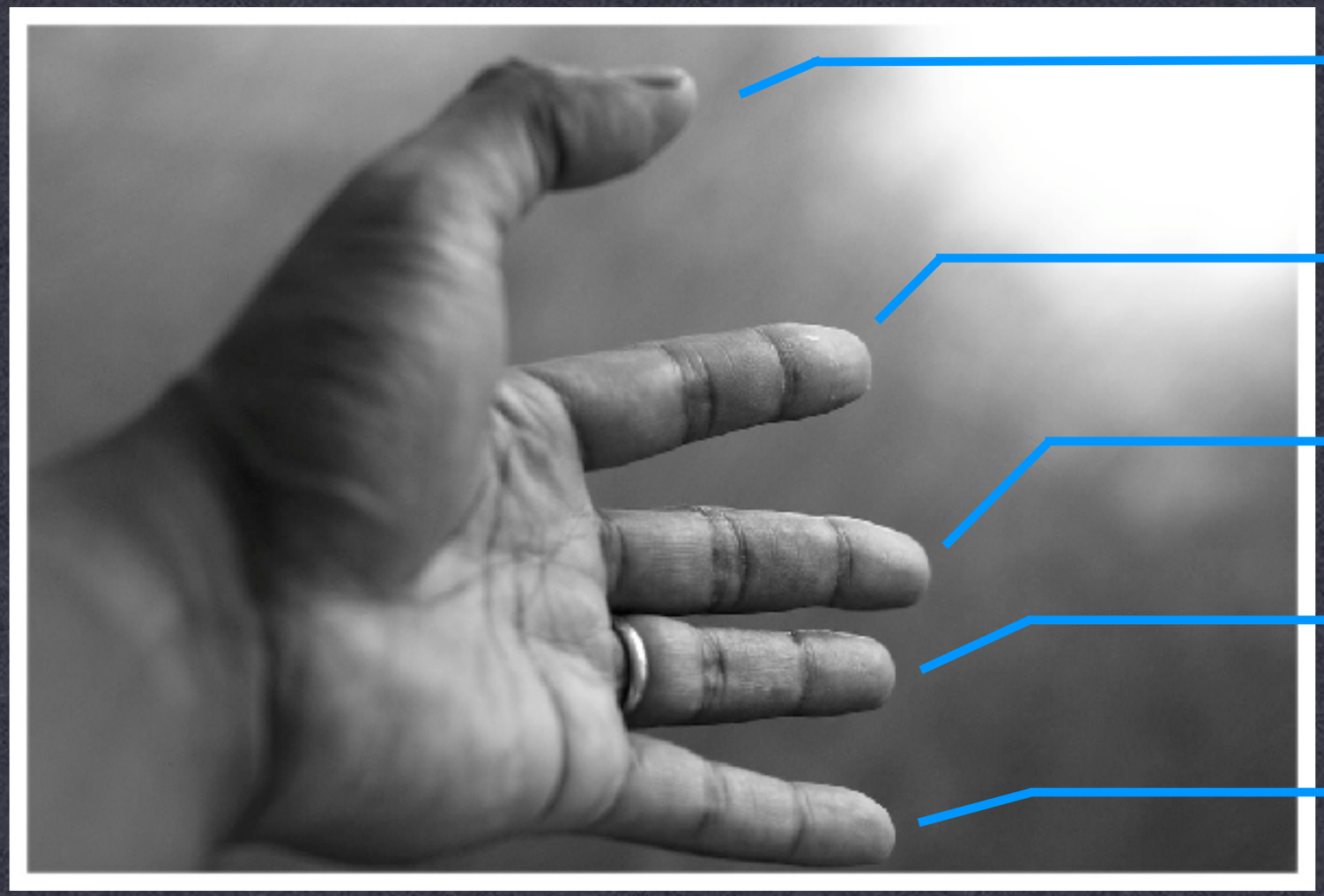
Many, many slides graciously donated by Lucas Ballard!

Instructor: Matthew Green

PKI & Certificates

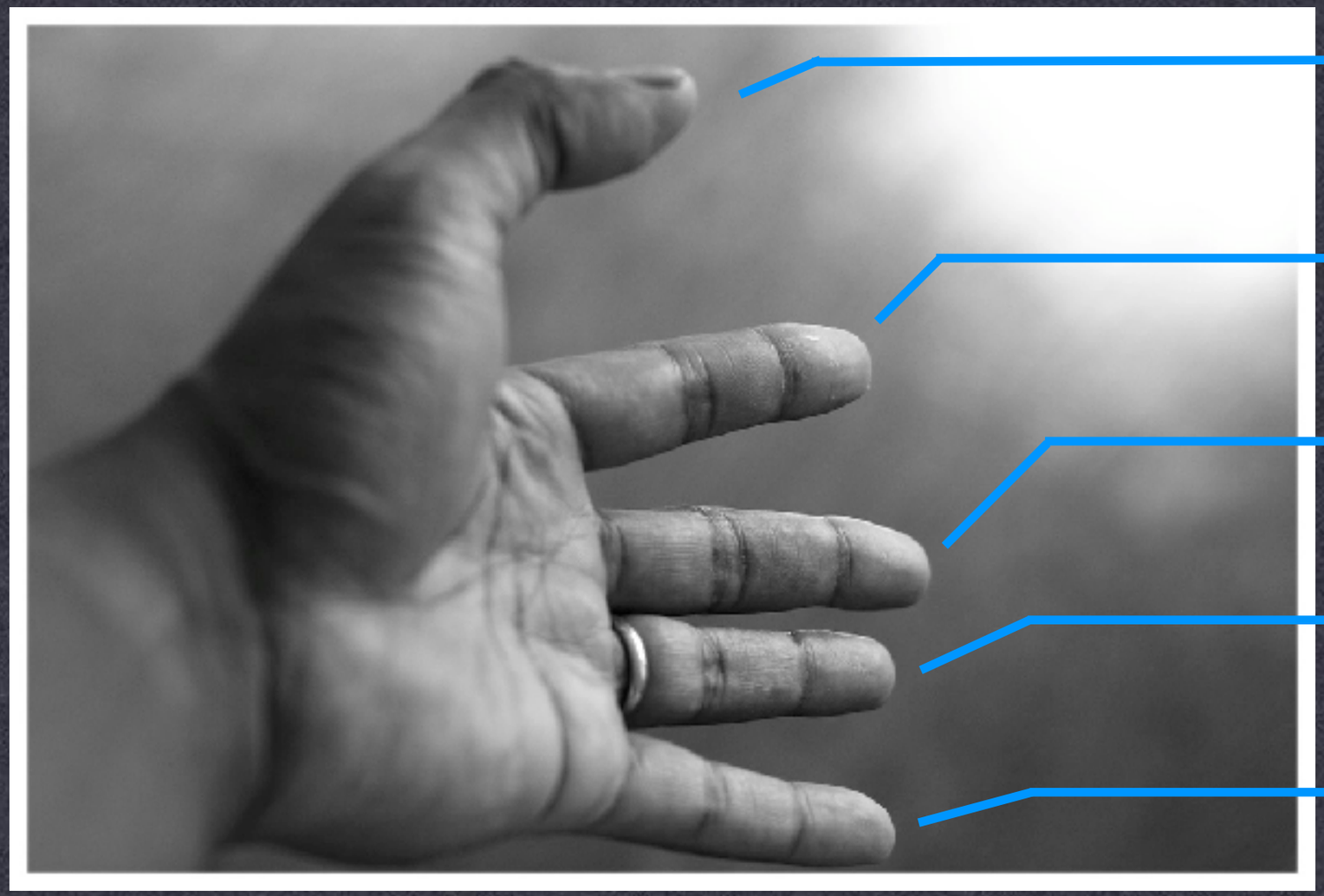
- How do I know to trust your public key?
  - Put it into a file with some other info, and get someone else to sign it!

Review



- Concept
- Primitives
- Protocols
- Implementation
- Usage

Review



Concept

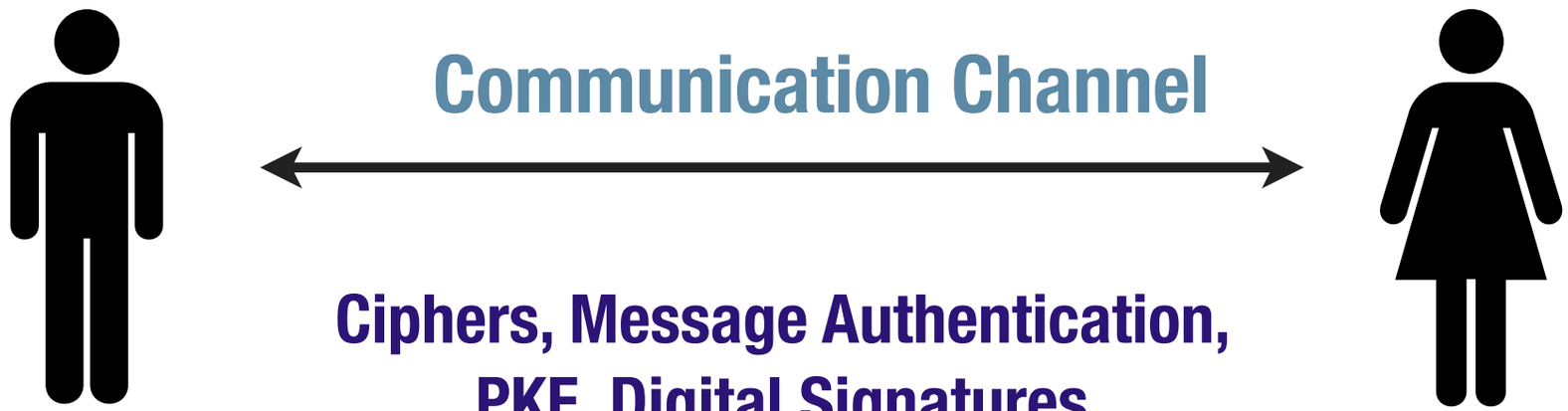
Primitives

Protocols

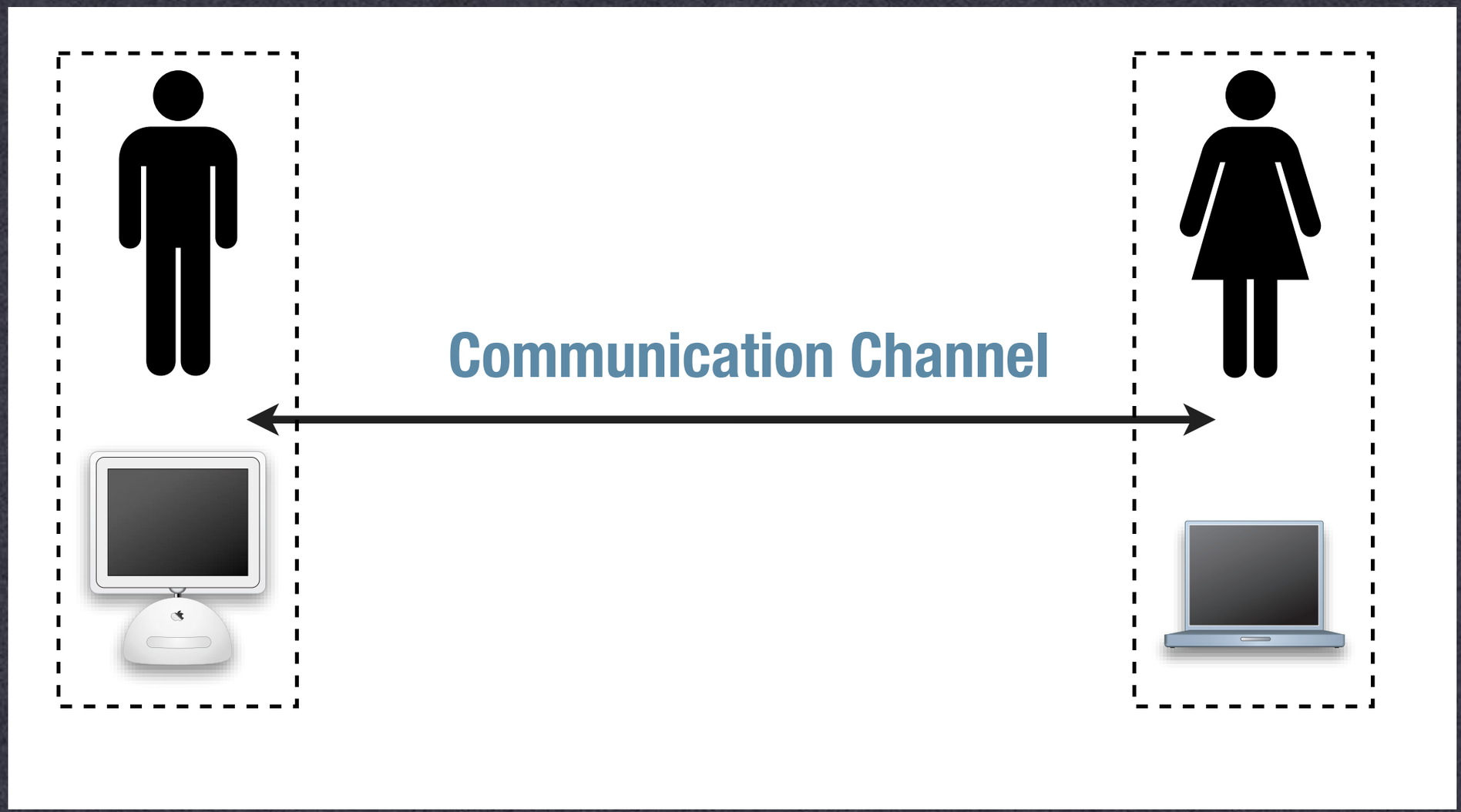
Implementation

Usage

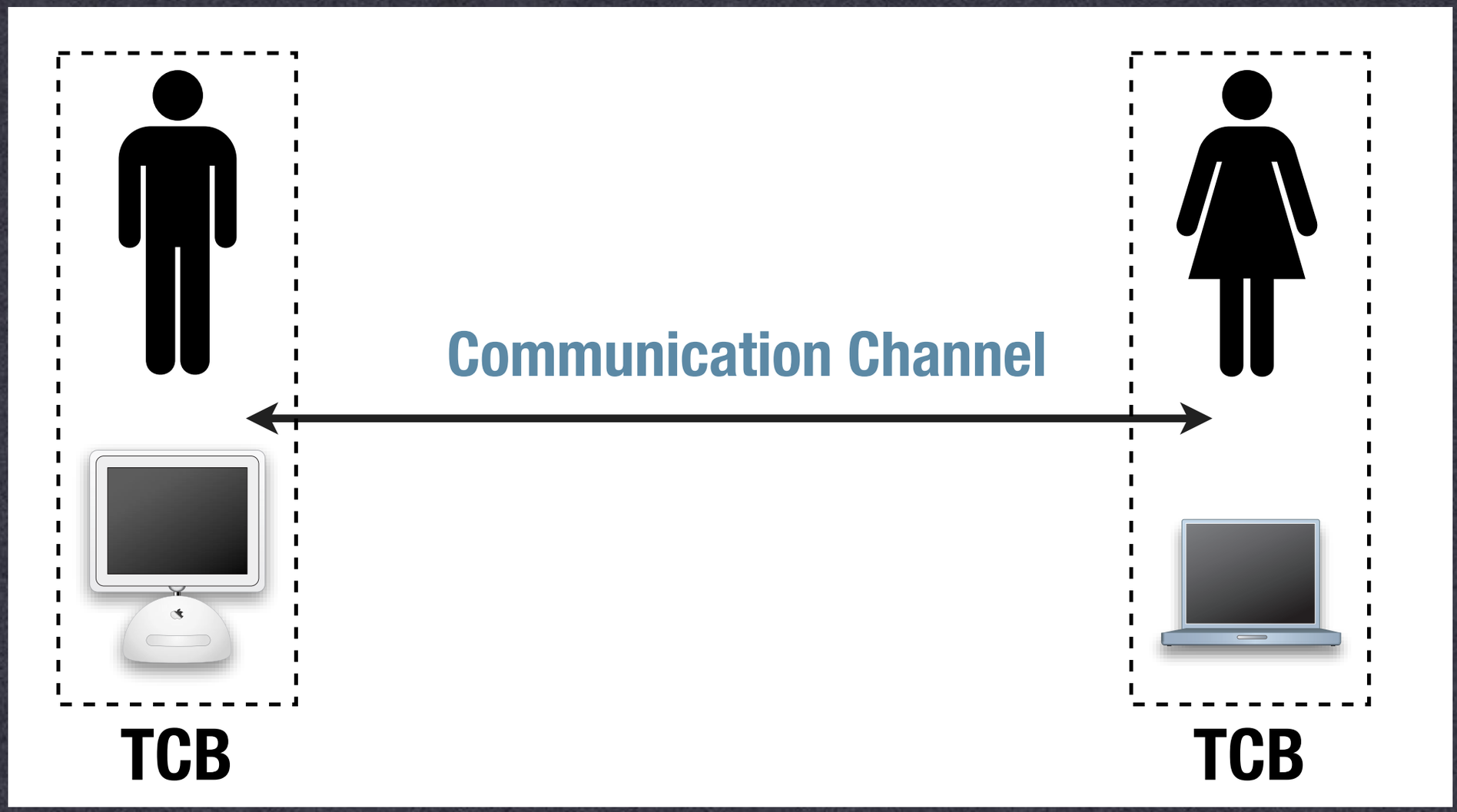
Review


Communication Channel
Ciphers, Message Authentication,
PKE, Digital Signatures

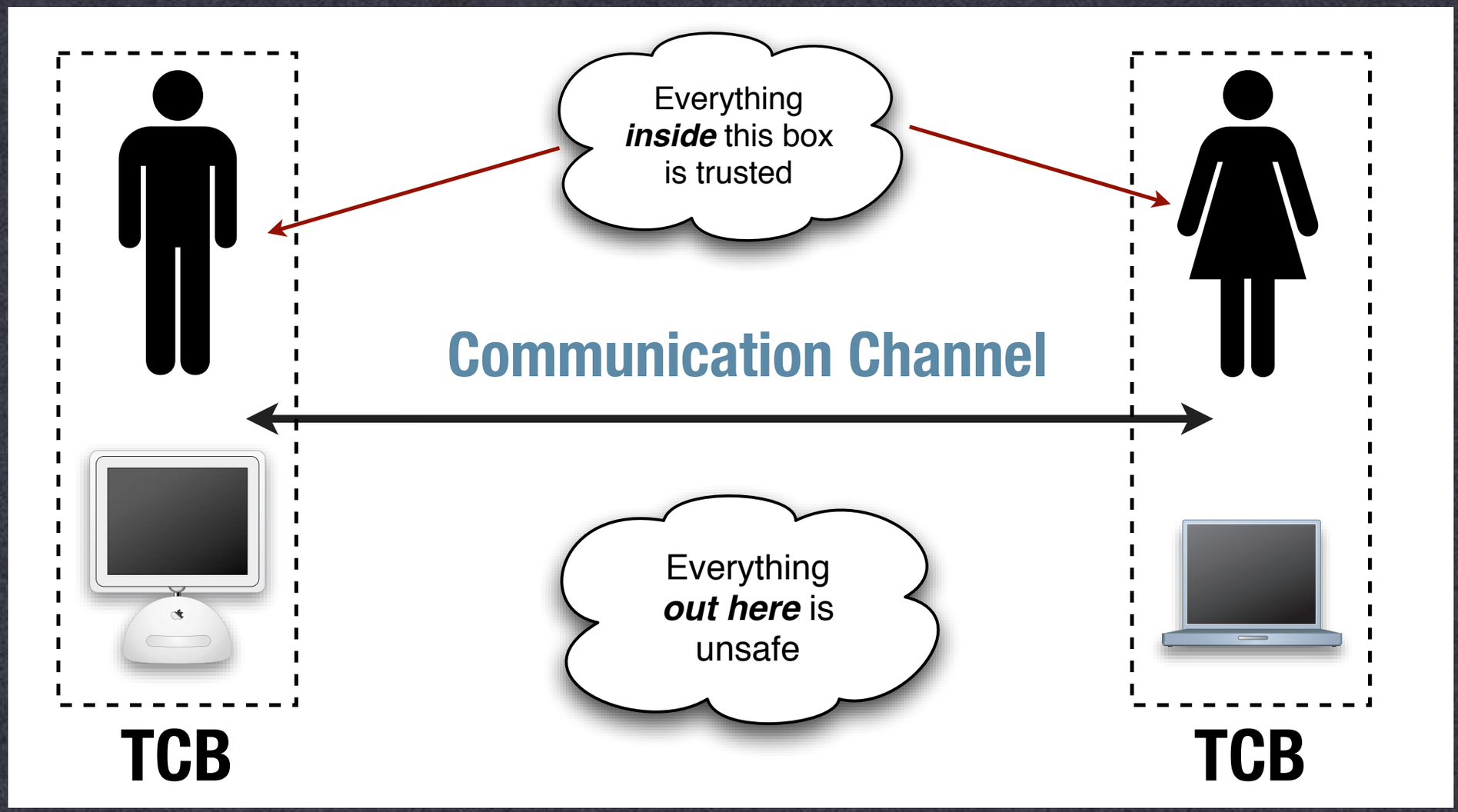
Review



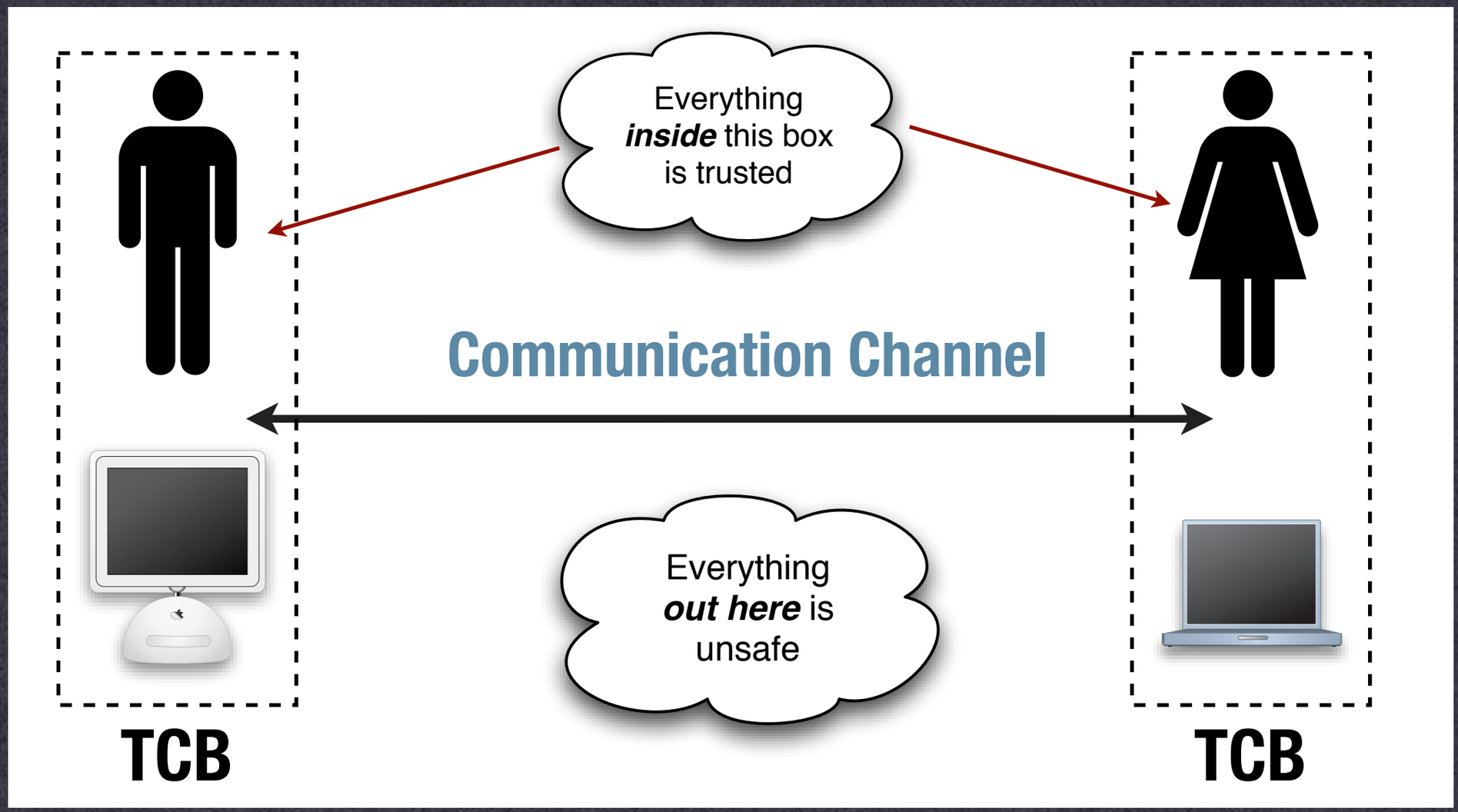
Review



Review


Everything
inside this box
is trusted
Communication Channel
Everything
out here is
unsafe
TCB
TCB

Review

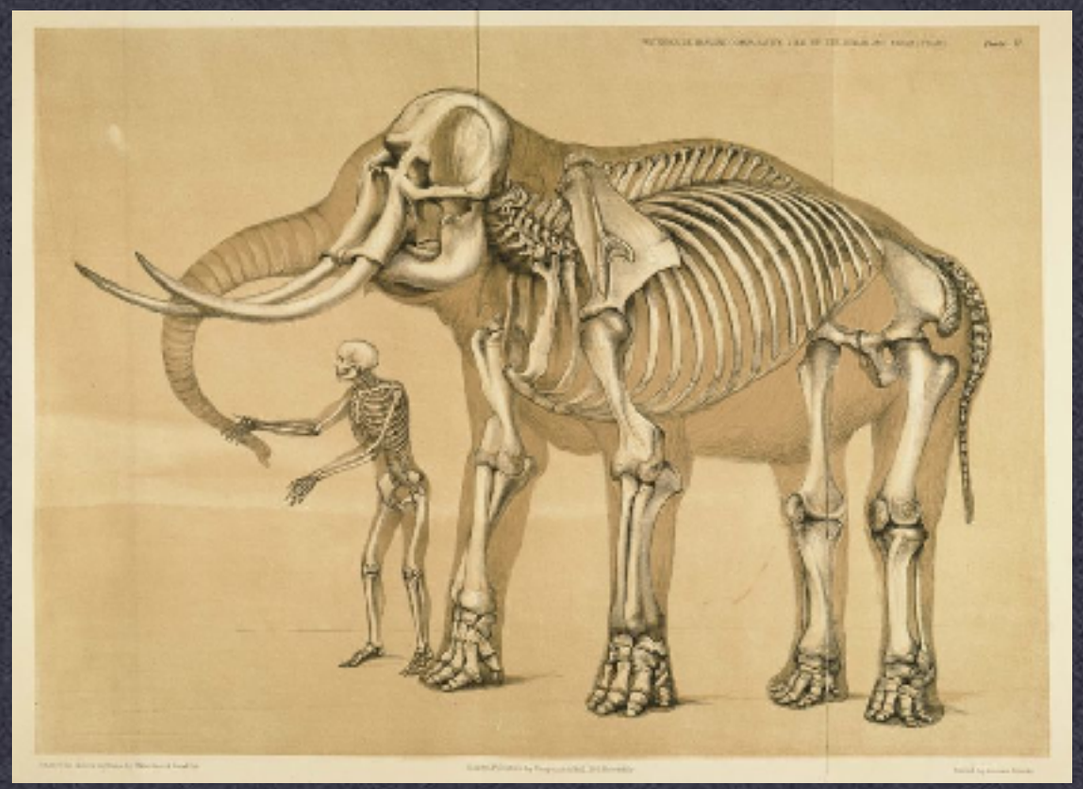


- **Just because something's "trusted"... doesn't mean that it's <u>trustworthy</u>.**

Today's Lecture

- The elephant in the room:
  - No matter what primitives we use...
  - No matter how carefully we design a protocol...
  - Somebody's going to implement it

  (and worse: in software)



Public domain image courtesy Wikipedia.

Oracle Security Alert #37
Created: 1 August, 2002
Updated: 5 August, 2002
Updated: 9 August, 2002
Updated: 24 September, 2002

OpenSSL Security Vulnerability

Description:

There are remotely exploitable buffer overflow vulnerabilities in OpenSSL versions prior to 0.9.6e.
These vulnerabilities may allow a remote attacker to execute arbitrary code or perform a denial-of-service (DoS) attack.

Security News:

OpenSSL overflowing with buffer problems

By Edward Hurley, News Writers
01 Aug 2002 | SearchSecurity

[Announce] GnuPG 1.4 and 2.0 buffer overflow

OpenSSL SSL_Get_Shared_Ciphers Off-by-One Buffer Overflow Vulnerability

Bugtraq ID: 25831
Class: Boundary Condition Error

Core Security Technologies Uncovers Vulnerability in Widely-Used Open Source Encryption Software

CoreLabs discovered that the scripts and applications using GnuPG are prone to a vulnerability involving incorrect verification of signatures. Unsuspecting users

September 6, 2002 6:39 AM PDT

Credit card theft feared in Windows flaw

By Joe Wilcox
Staff Writer, CNET News

Post a comment

Microsoft late Wednesday said that a flaw in its Windows operating system could allow hackers to gain unauthorized access to thousands of computers.

CONSOLE HACKING 2008: WII FAIL

Is implementation the enemy of design?

marcan and bushing
Team Twiizers

Vulnerability in Citrix Presentation Server could result in cryptographic settings not being correctly enforced

Today

- How <u>software</u> fails
  - Stupid humans
  - Poor algorithm implementation
  - Memory exploits
  - Bad key management/generation

-Database injection

  - Verbose error messages

Stupid humans

- Do you really trust your users?
  - Key generation, export & backup
  - Keys protected with user-supplied passwords
  - Overrides, fail open, fail closed:

Why Johnny Can't Encrypt:
A Usability Evaluation of PGP 5.0

Algorithm implementation

- **Digital signatures:**
  - **Authenticating public keys (X.509 certs)**
  - **Signed software updates**
  - **Example: RSA signature**

$$N = p \cdot q$$

$$\phi(N) = (p-1)(q-1)$$

$$pk = (e, N)$$

$$sk = d$$

Signing

$$s = m^d \ mod \ N$$

Algorithm implementation

- **RSA PKCS #1 v1.5 Signing:**
  - **First hash the message (e.g., SHA1)**
  - **Add structured padding**
  - **Append hash**

~ 1024 bits (128 bytes)

Algorithm implementation

- **RSA PKCS #1 v1.5 Verify:**
  - **Check padding structure**
  - **Make sure hash is right justified**
  - **Compare hash**

~ 1024 bits (128 bytes)

Algorithm implementation

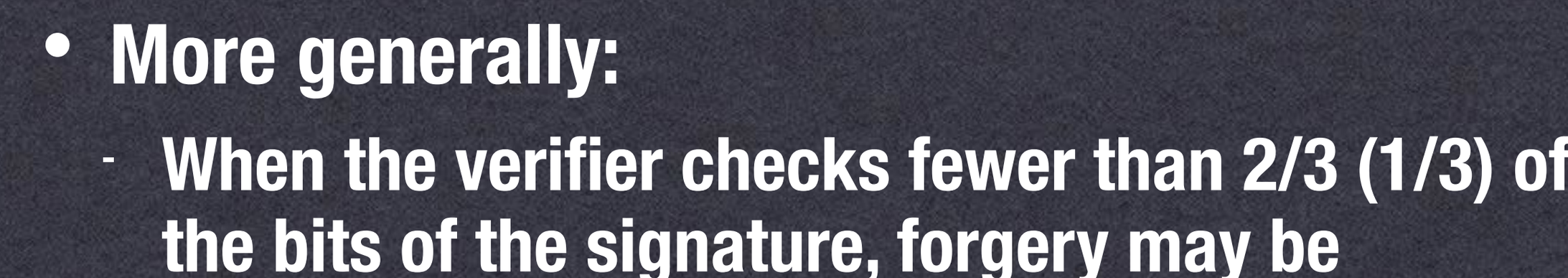
- Bleichenbacher's PKCS #1 v1.5 signature vulnerability
  - Applies to implementations with e=3
  - Some implementations don't check that digest is right justified
  - Why is this a problem?

~ 1024 bits (128 bytes)

Algorithm implementation

- More generally:
  - When the verifier checks fewer than 2/3 (1/3) of the bits of the signature, forgery may be possible
- How to fix it?
  - Check all signature bits!

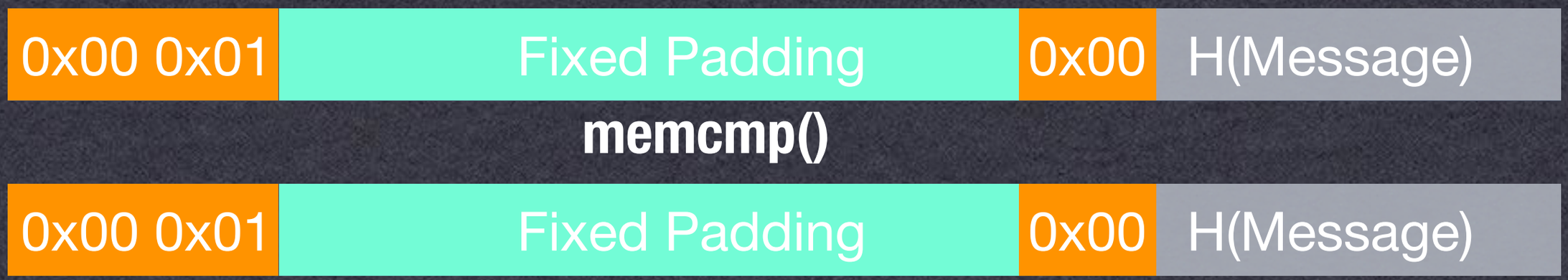
Reconstruct from scratch:



Algorithm implementation

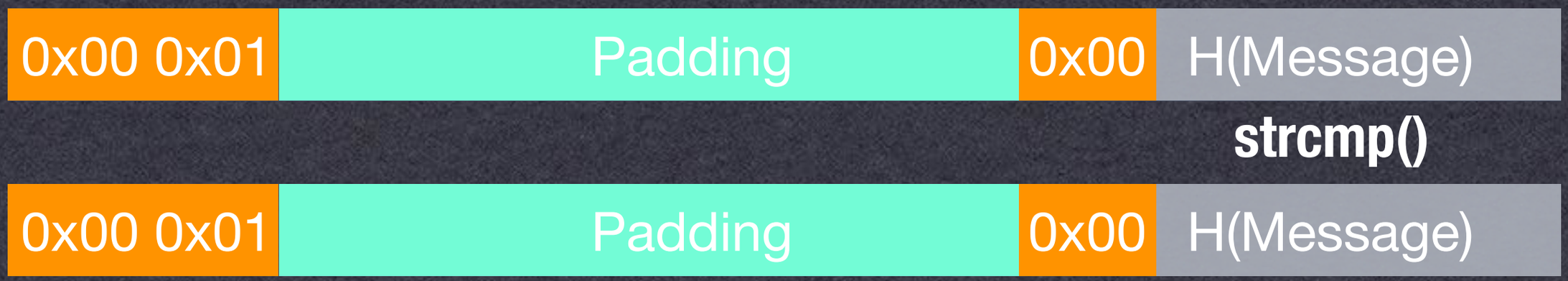
- **More generally:**
  - When the verifier checks fewer than 2/3 (1/3) of the bits of the signature, forgery may be possible
- **How to fix it?**
  - Check all signature bits!

Reconstruct from scratch:



Algorithm implementation

- Wii software patching:
  - Used strcmp() instead of memcmp()
  - Only checked the hash!
  - Comparison ends at the first 0 byte in the hash
  - The attack?



Algorithm implementation

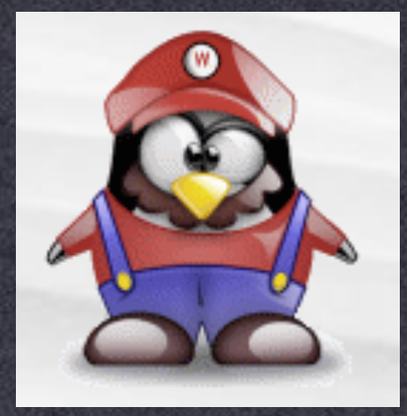
- Wii software patching:
  - Used strcmp() instead of memcmp()
  - Only checked the hash!
  - Comparison ends at the first 0 byte in the hash
  - The attack?

A digression

- De-incentivize your attacker:
  - Never get in the way of users' ability to play free games :)

-(Probably no alternative for Nintendo)

  - Never get in the way of users' ability to install Linux on their game system



Memory Exploits

- Most cryptographic software written in C/C++
  - C++: efficiency > security
  - No runtime bounds checking, limited type safety
  - Results in memory exploits
  - These can be exploited remotely!

Stack-based Exploits

- Stack-Smashing (Axe)
- return-to-libc (Steak Knife)
- format string (Scalpel)

Slide courtesy of: Lucas Ballard

Memory

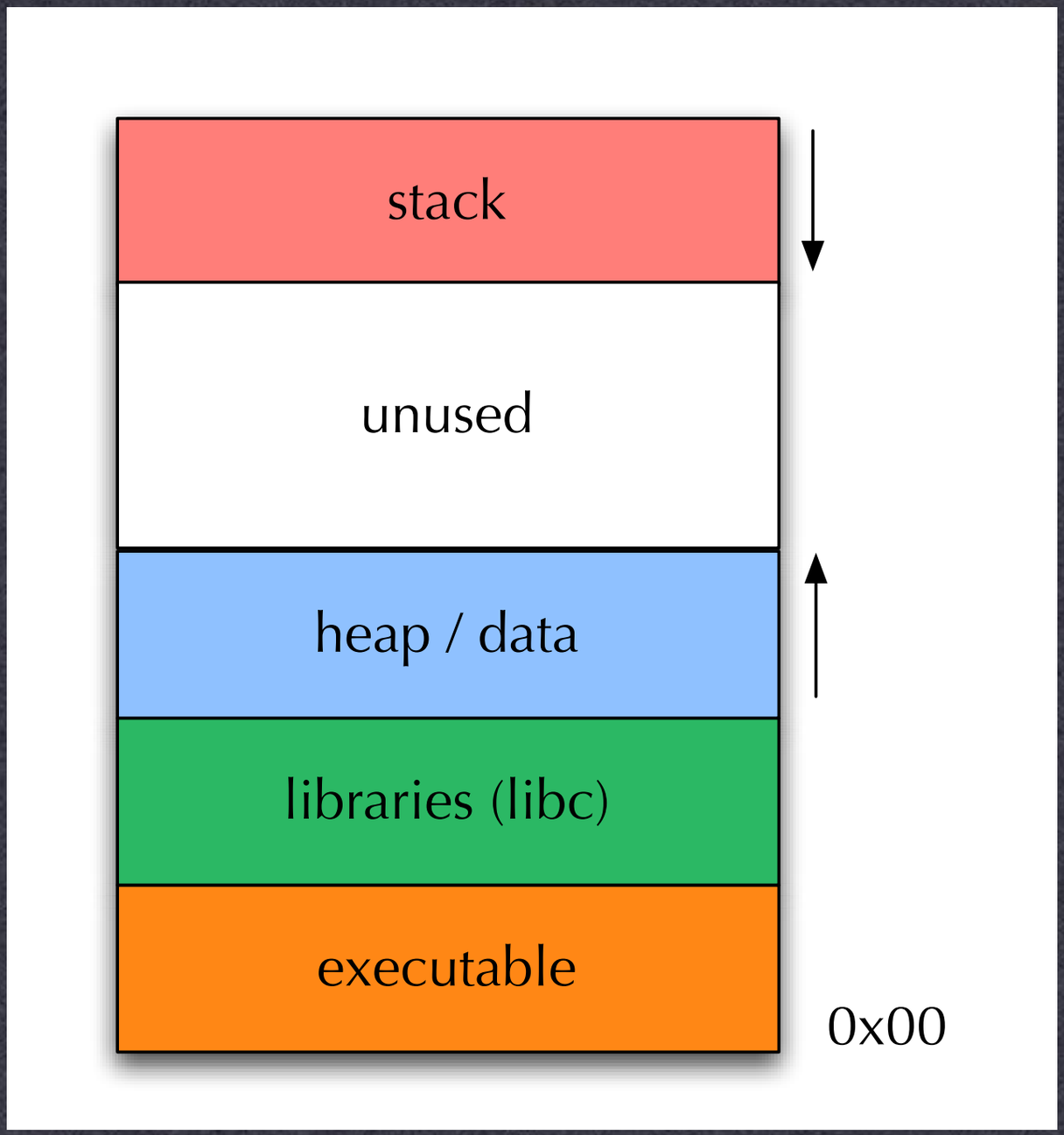


Image by Lucas Ballard, used with permission

Buffer Overflow

- Buffer is an area of computer memory
  - Has a defined size
  - Usually near other important things in memory
- Example:

Form 1 Individual Wage Tax 1997

Your first name and initial (if joint return, also give spouse's name and initial) Last name	Your social security number
BUFFER 1	BUFFER 2
Present home address (number and street including apartment number or rural route)	Spouse's social security no.
BUFFER 3	BUFFER 4
City, Town or Post Office, State and ZIP Code	Your occupation Spouse's occupation

1. Wages and Salary	1	
2. Taxpayer exemptions		
a. $26,000 for married filing jointly	2(a)	
b. $13,000 for single	2(b)	
c. $17,000 for single head of household	2(c)	
3. Number of dependents, not including spouse	3	

Buffer Overflow

- **Buffers have limited size, hence:**
  - **If you put too much into one, it overflows**
  - **Usually into some other buffer**

Form 1 Individual Wage Tax 1997

Your first name and initial (if joint return, also give spouse's name and initial) Last name	Your social security number
Rachelle Posner Deacons-Smith	
Present home address (number and street including apartment number or rural route)	Spouse's social security no.
City, Town or Post Office, State and ZIP Code	Your occupation
	Spouse's occupation

1. Wages and Salary	1	
2. Taxpayer exemptions		
a. $26,000 for married filing jointly	2(a)	
b. $13,000 for single	2(b)	
c. $17,000 for single head of household	2(c)	
3. Number of dependents, not including spouse	3	

Example

```c
#include <stdio.h>
#include <string.h>

bad_code(char *str)
{
  char buffer[20];

  strcpy((char*)buffer, str);
  printf("%s\n", (char*)buffer);
}

main(int argc, char **argv)
{
  bad_code(argv[1]);
}
```

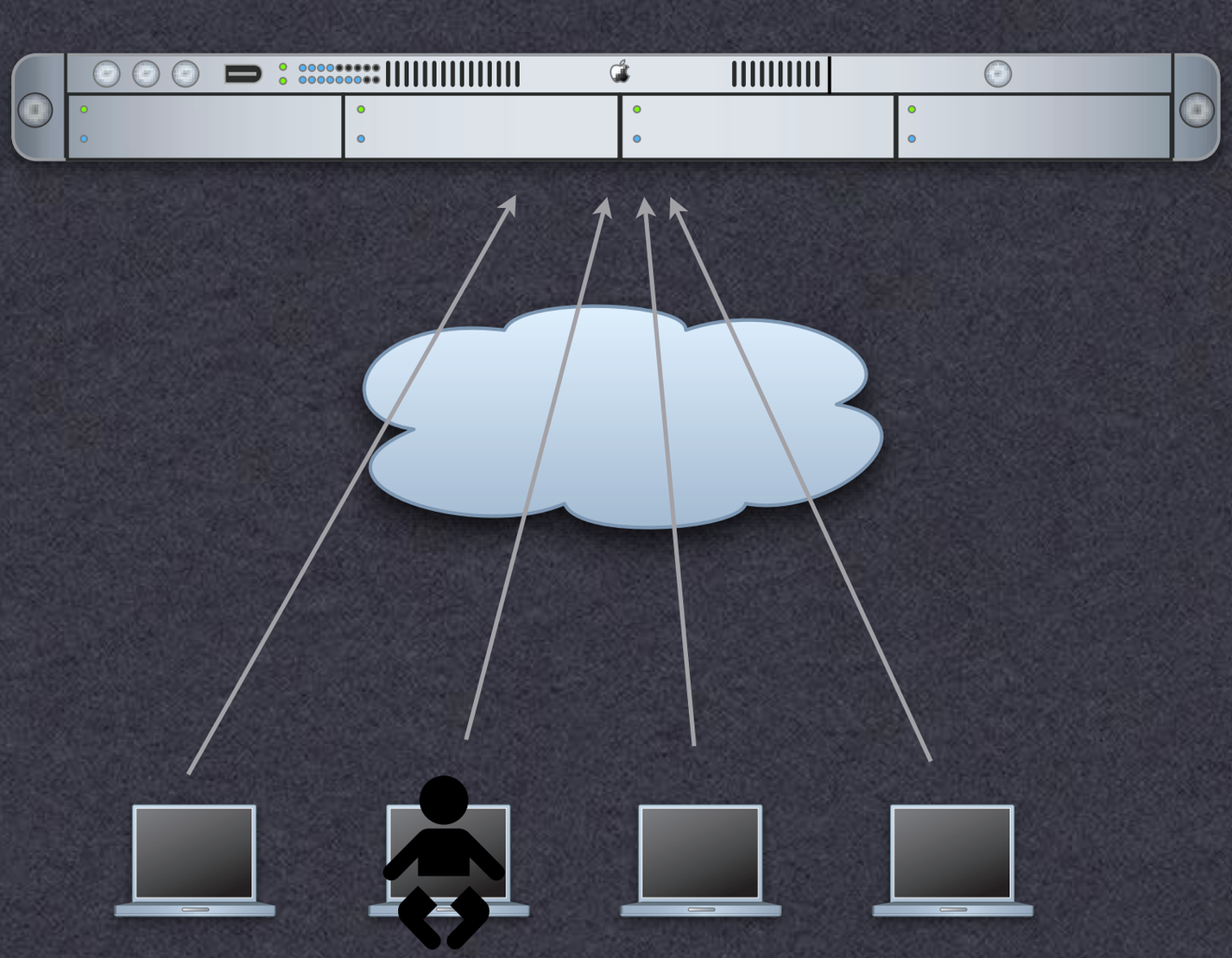
```
bash-3.2$ ./test nothing_wrong
nothing_wrong
```

```
bash-3.2$ ./test something_is_very_wrong_with_this_one
something_is_very_wrong_with_this_one
Segmentation fault
```

Typical Result

```
bash-3.2$ ./test something_is_very_wrong_with_this_one
something_is_very_wrong_with_this_one
Segmentation fault
```

- **But if we're careful:**
  - **Overwrite memory with chosen values**
  - **Include (malicious) code in the input**
  - **But how do we get our code to run...?**




OpenOffice


cruzer
micro
4 GB

The Stack

```
void f(int a){
  h(a + 1);
}

void h(int b){
  g(&b);
}

void g(int *c){
  g2(c);
  g2(c - 1);
}
```

Stack Frames

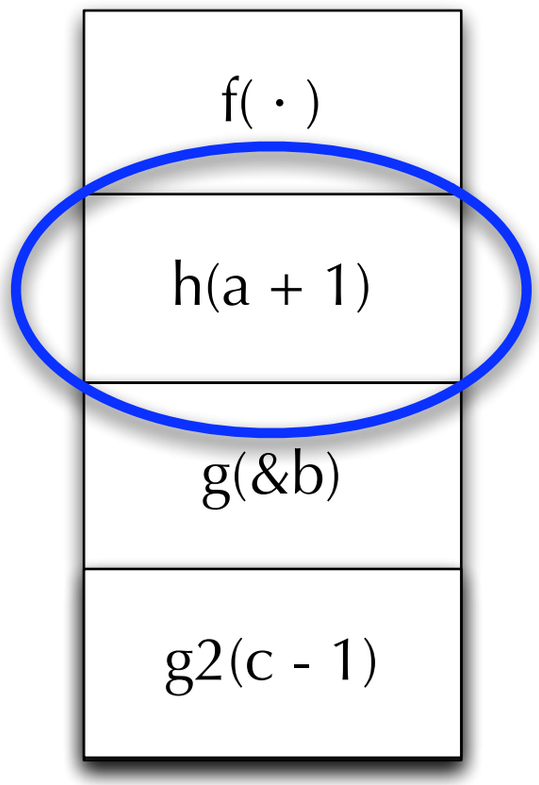
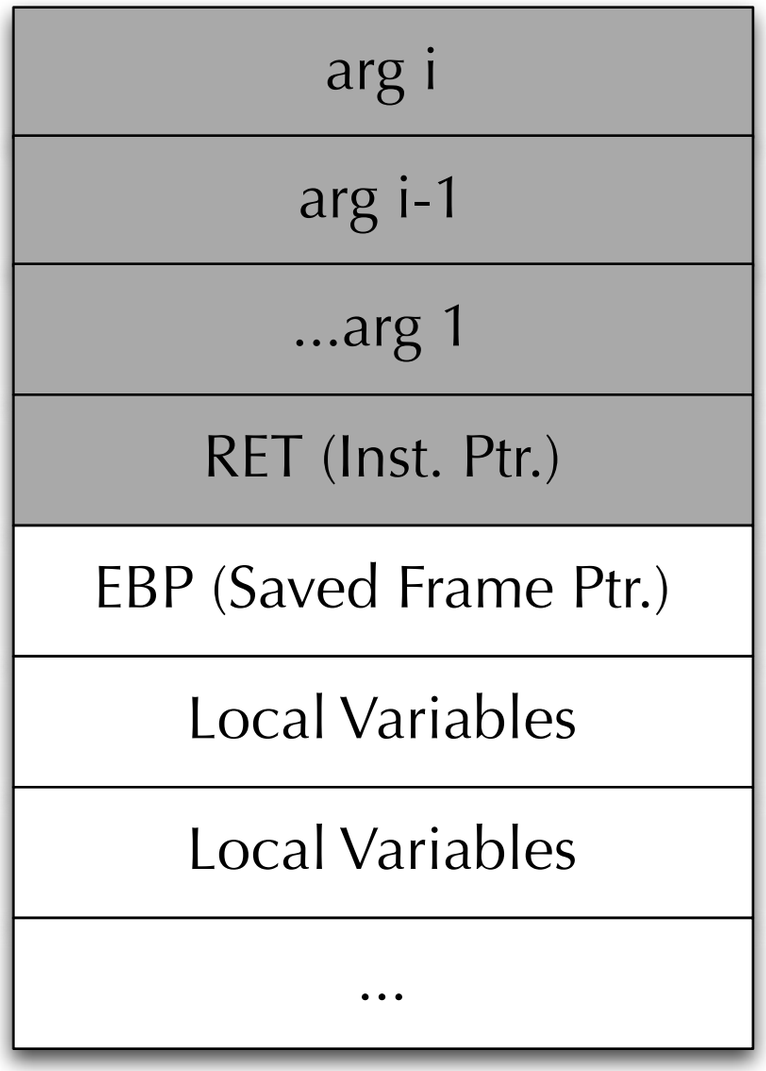


Image by Lucas Ballard, used with permission

Stack Frame

Caller

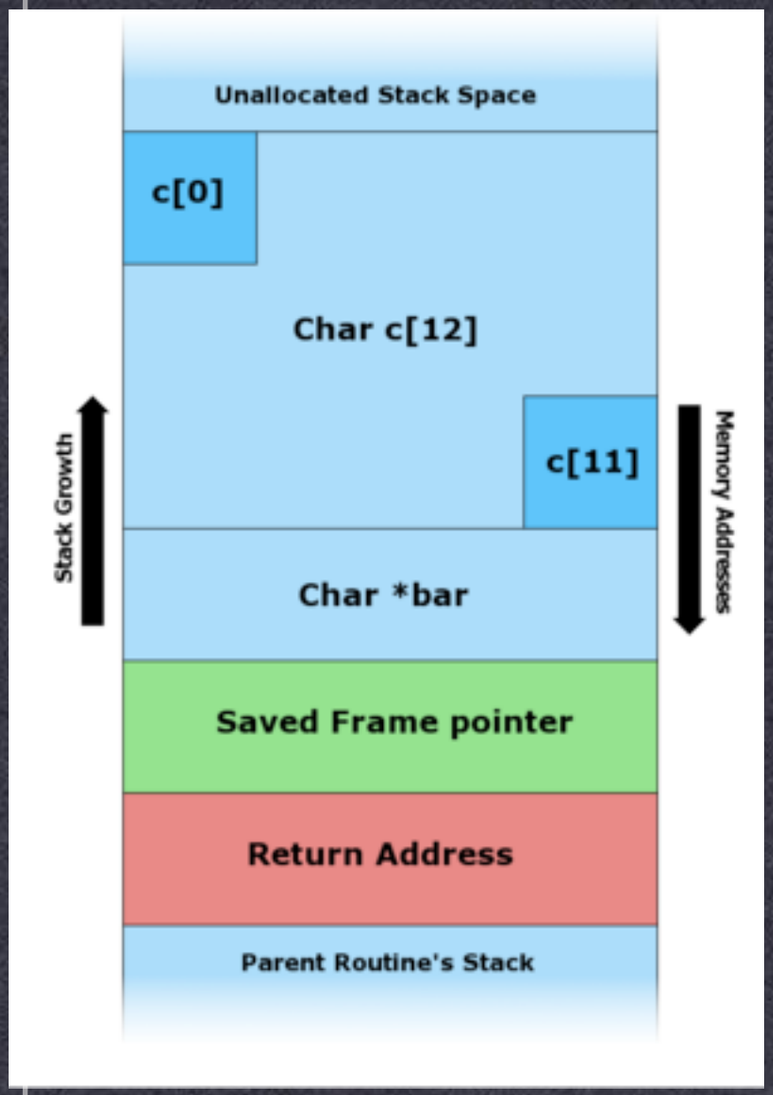
Callee

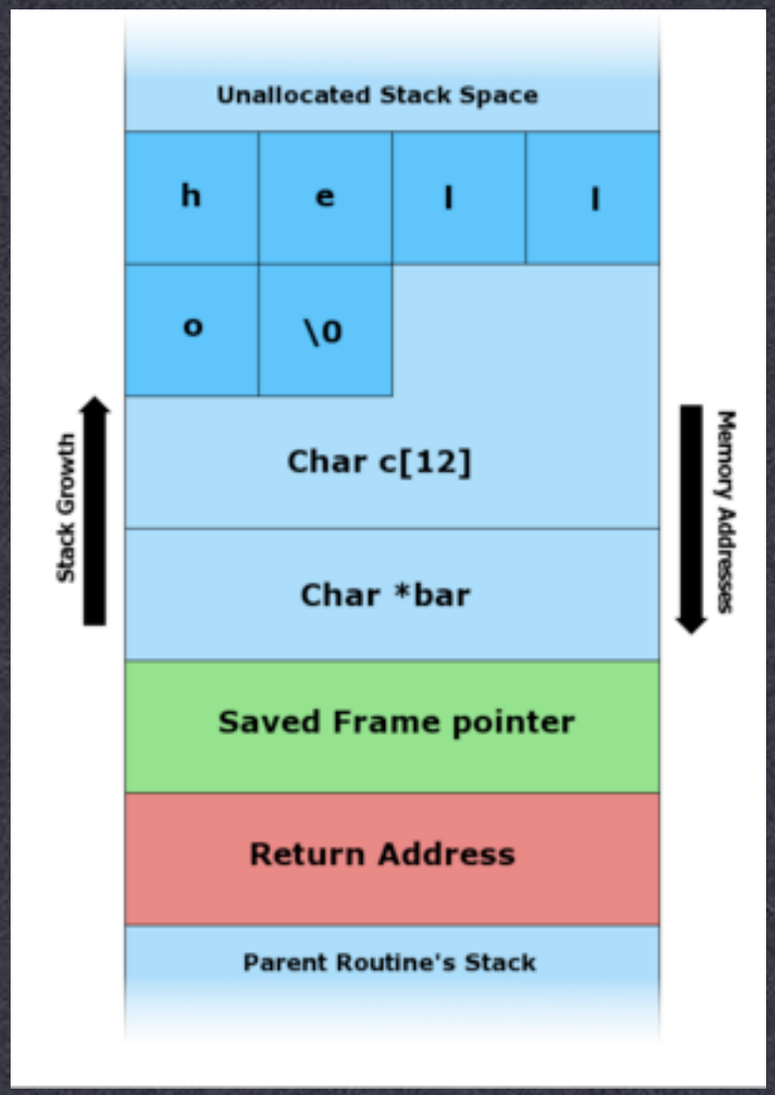


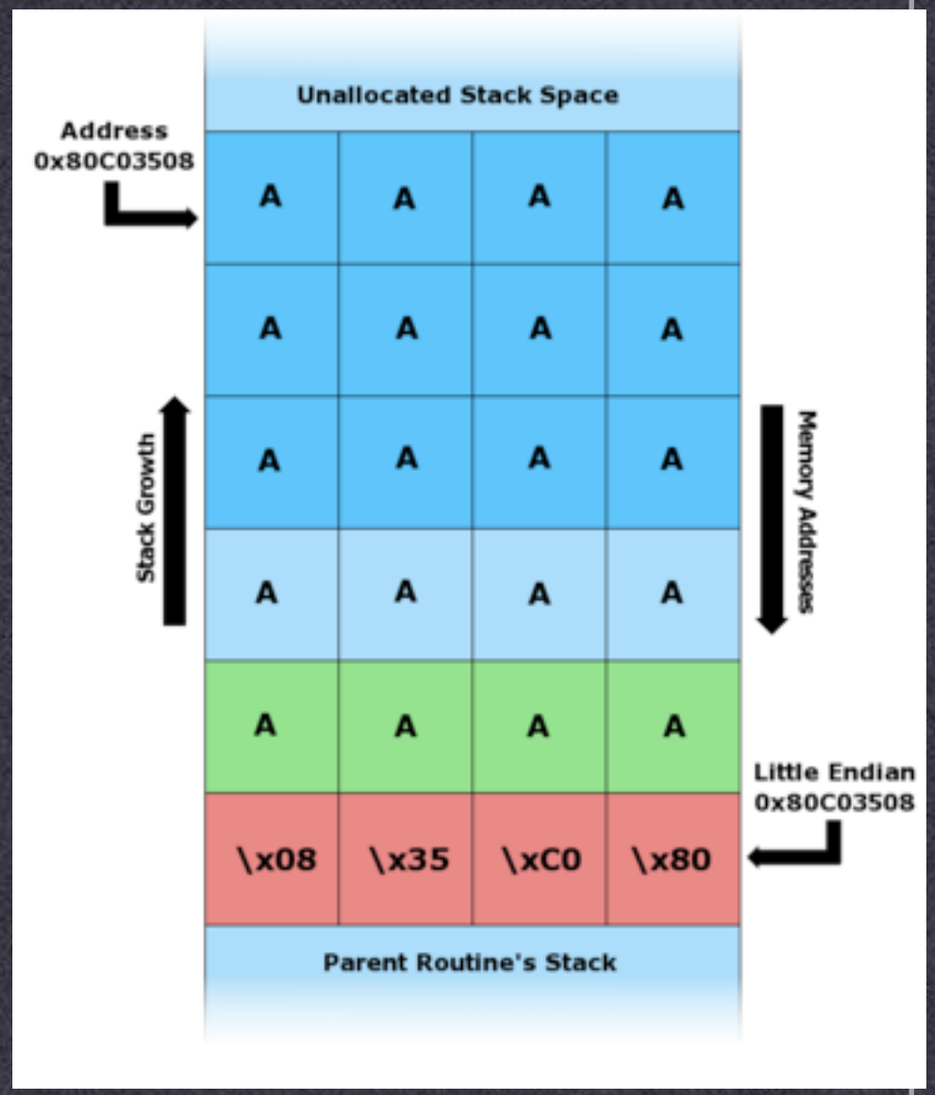
(Inst. Ptr. points to executable)

Slide courtesy of: Lucas Ballard

“Stack smashing”

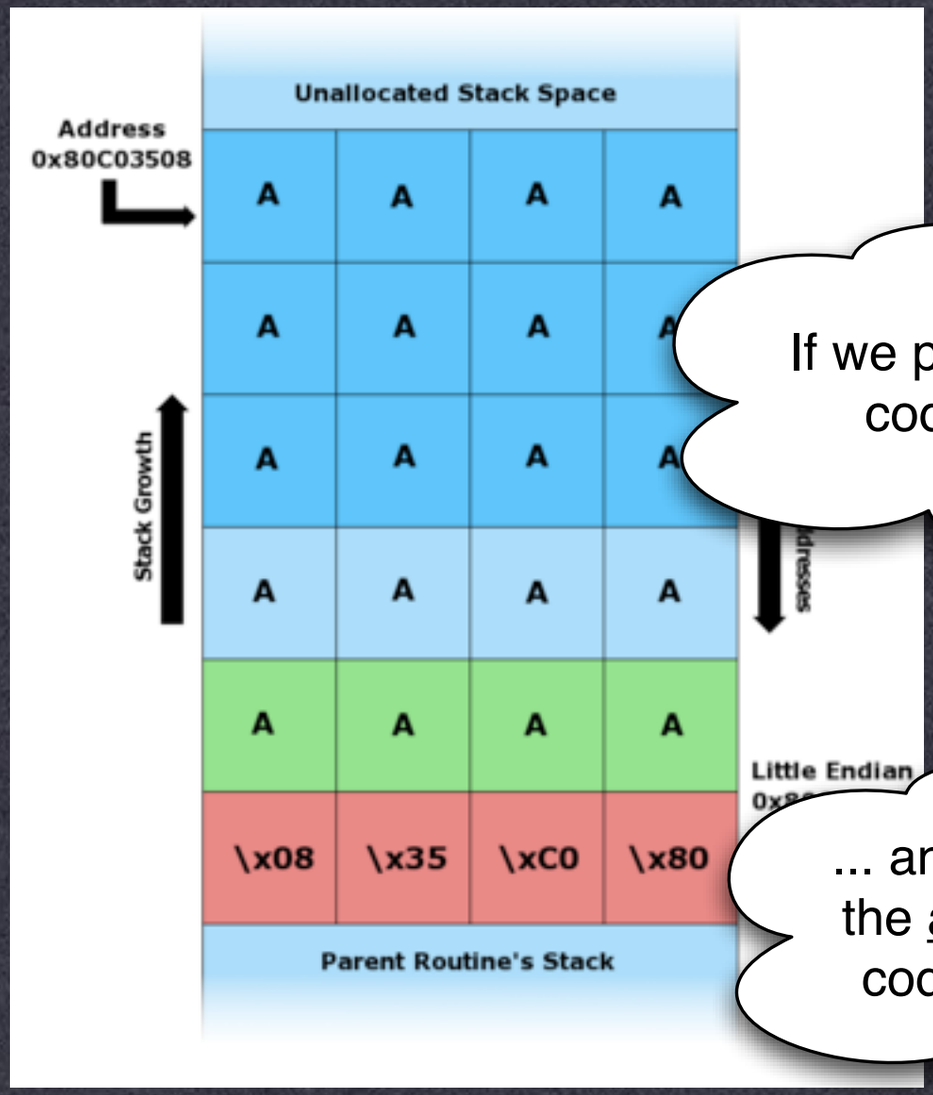






Public domain image courtesy Michael Lynn, Wikipedia (Stack Overflow).

Stack smashing



Stack smashing

- Caveats:
  - Need enough room to inject useful shellcode
  - If it's a string--- can't use 0x00 bytes

-But we can work around this

  - Must know the exact address where our shellcode will be located
  - May only get one "shot" at it

Slide courtesy of: Lucas Ballard

Return-to-libc

- **Why provide our own shellcode?**
  - **OSes already have lots of useful code sitting in libraries. For example:**

```
SYSTEM(3)                BSD Library Functions Manual                SYSTEM(3)

NAME
     system -- pass a command to the shell

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdlib.h>

     int
     system(const char *command);

DESCRIPTION
     The system() function hands the argument command to the command inter-
```

- Idea: point to libc instead of back stack
  - system(), exec*()
- Modify “arguments” in addition to
- Depending on situation, can “chai
  - setuid() system

Return-to-libc

- Idea: point to libc instead of back into the stack

  `system(), exec*()`

- Modify “arguments” in addition to ret. addr.

  - Depending on situation, can “chain” calls:
    - setuid(...); system(...); ...

Slide courtesy of: Lucas Ballard

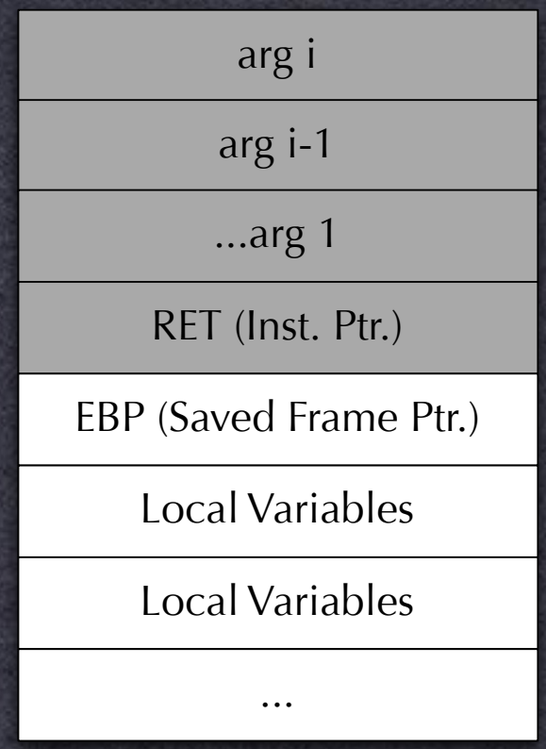
Format strings

- **Standard functions (printf, sprintf, snprintf)**
  - **Accept “format strings”**

```
printf ("Characters: %c %c \n", 'a', 65);
printf ("Decimals: %d %ld\n", 1977, 650000L);
printf ("Preceding with blanks: %10d \n", 1977);
printf ("Preceding with zeros: %010d \n", 1977);
printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100,
100, 100, 100);
printf ("Width trick: %*d \n", 5, 10);
printf ("%s \n", "A string");
```

Reading the stack

```
int parse_user_supplied_buffer(buffer)
{
   printf(localbuf, buffer);
   ...
}
```

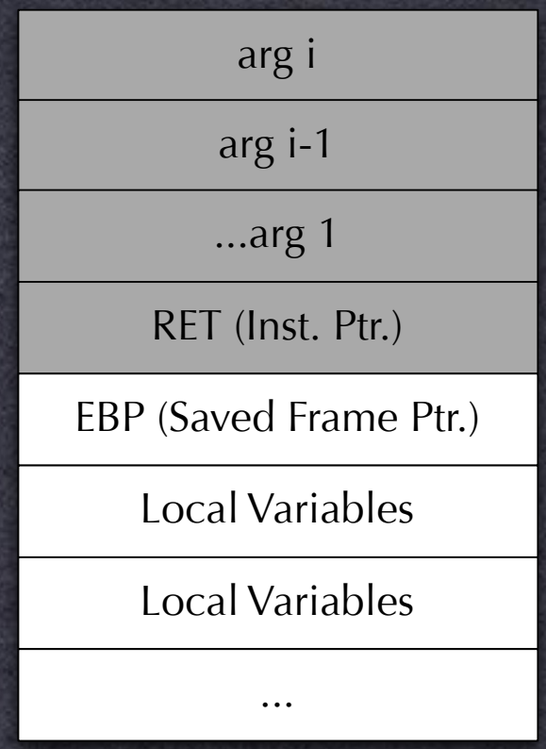



```
> ./program “hello %08x.%08x.08x”
hello 02ab34fe.3ed8273d.836abfed
```

Slide courtesy of: Lucas Ballard

Reading arbitrary memory

```
int parse_user_supplied_buffer(buffer)
{
   printf(localbuf, buffer);
   ...
}
```

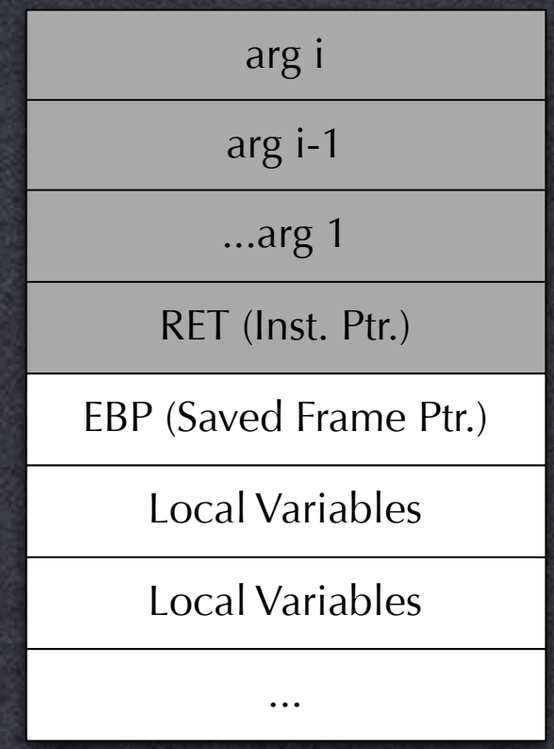



```
> ./program
“\x23\x22\x0A\x43_%08x.%08x.%08x.%08x.%08x|%s|”
hello 023uhdfh..a,sCRYPTOGRAPHICKEYkaduueuj
```

Overwriting memory

```
int len;

printf ("Print out some stuff %n\n", &len);
printf ("I printed %d bytes\n", len);
```



- **Good news: Disabled in some modern compilers**
  Bad news: Not all of ‘em

Integer overflow

```
call(char* buf1, int user_supplied_length) {
   /* Check for malicious value */
   if (user_supplied_length > MAX_LENGTH) {
      return -1; /* Sneaky user! */
   }

   /* It's ok!  We can trust the value. */
   memcpy(buf1, buf2, user_supplied_length);
}
```

Integer overflow

```
call(char* buf1, int user_supplied_length) {
  /* Check for malicious value */
  if (user_supplied_length > MAX_LENGTH) {
    return -1; /* Sneaky user! */
  }

  /* It's ok!  We can trust the value. */
  memcpy(buf1, buf2, user_supplied_length);
}
```

Signed comparison

Unsigned usage

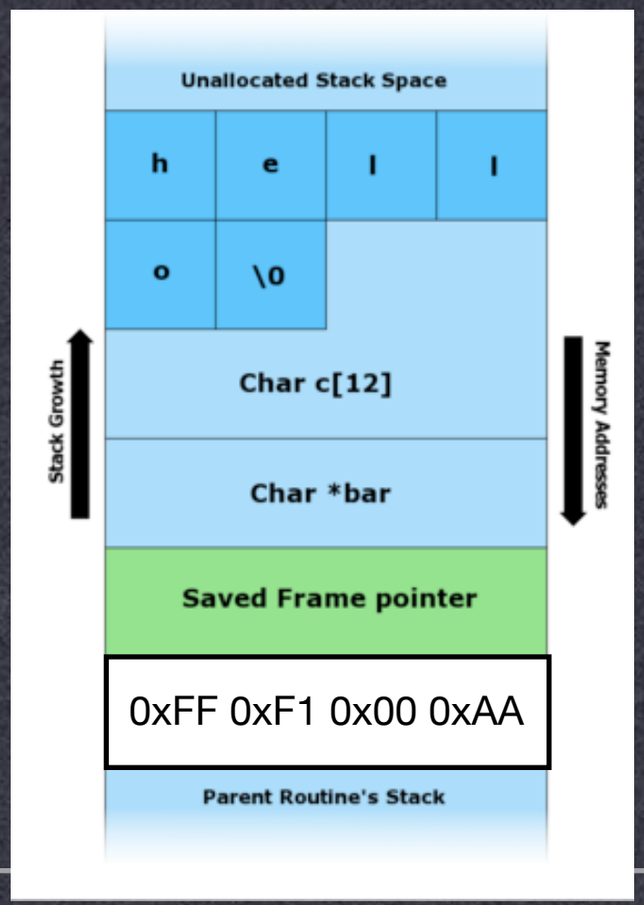
How to fix this?

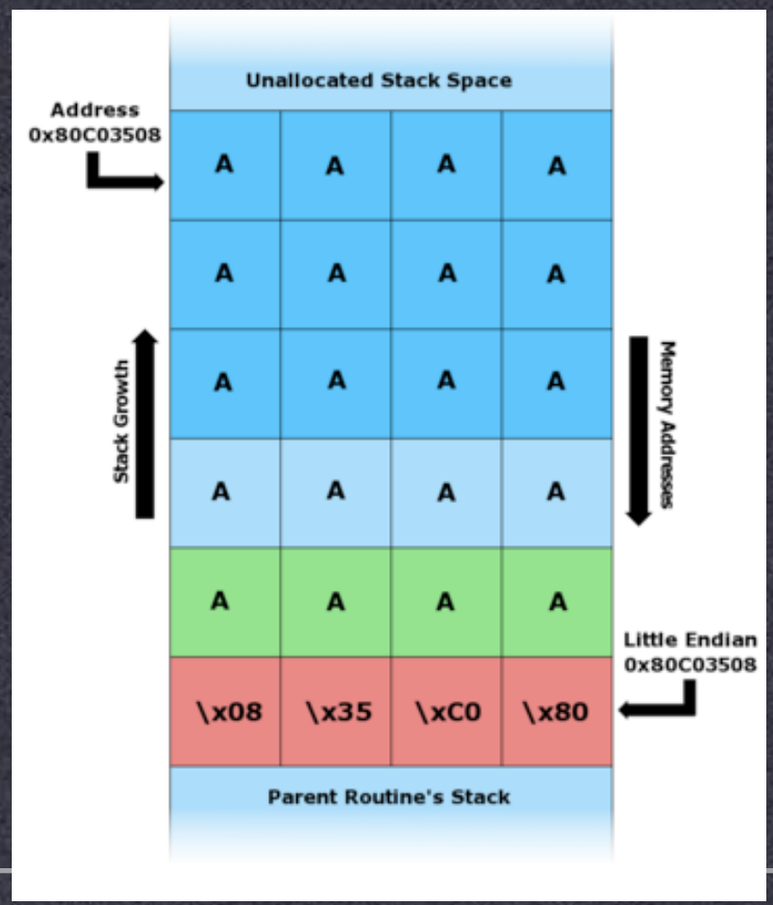
- Handful of approaches:
  - Stack canaries
  - Shadow stacks
  - W⊕X Pages
  - Write better code (machine analysis, safe calls)

Some material courtesy of: Lucas Ballard

Canaries

- Place structured values onto stack
  - Check them before we return
  - But what about format string attacks?





W⊕X Pages

- Simple idea:
  - Executable code is basically static
  - Shouldn't be written to during execution (at least, after libraries loaded)
  - So mark each page as either writeable or executable (can make the decision dynamically)

  - But this still doesn't solve the problem!!!

Some material courtesy of: Lucas Ballard

Shadow Stacks

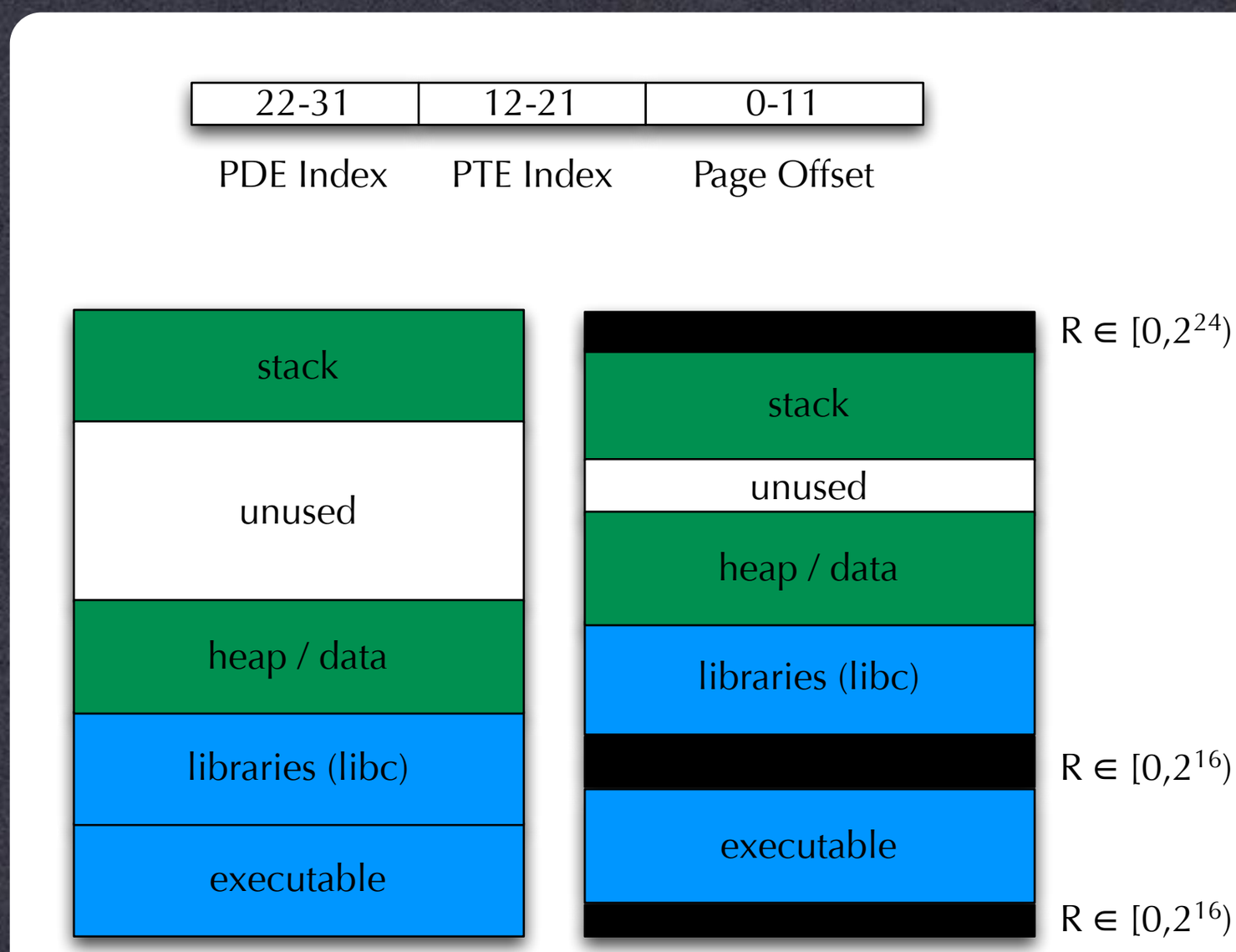
- Keep an extra copy of return address in Kernel Memory
  - Only return if addresses match up
  - Doesn't protect other memory, registers, etc.

Some material courtesy of: Lucas Ballard

Address Obfuscation

- Randomize address space
  - Place stack, buffers at random location
  - Thus, attackers won't know precise address to point control flow
  - E.g., PaX ASLR, Windows Vista, etc.

PaX ASLR



Some material courtesy of: Lucas Ballard

Limitations of ASLR

- Several limitations:
  - 16 bits not a huge number
  - Doesn't re-randomize on fork()
  - So just try over and over again until you guess randomization
  - 64-bit addressing helps a lot here
  - Sometimes libraries (e.g., DLLs) aren't randomized

Some material courtesy of: Lucas Ballard

Solutions

- Use a safe language...?
  - Java, Ruby, etc.
  - Enforce bounds checking, garbage collection
  - Type safety
  - Don’t <u>ever</u> let programmers near the memory!
  - We can even run untrusted code in a sandbox

Solutions

- Use a safe language...?
  - Vulnerabilities in JVM
  - Thread issues
  - Load malicious libraries
  - Efficiency?

Microsoft Identifies Eight JVM Vulnerabilities

by Nate Mook

December 12, 2002, 9:54 PM

In a single security bulletin issued late Wednesday, Microsoft disclosed eight new security vulnerabilities discovered in its java virtual machine. Build 5.0.3805 and older are at risk, containing one flaw rated "critical," two "important," two "moderate" and three classified as "low" severity.

Solutions

- Interpreted languages aren't always our friend
  - The new frontier is finding bugs in VMs
  - If you can run arbitrary "safe" code, then it gets lots of chances to work its way out

Crazy stuff

- Mark Dowd's Flash Attack:
  - Cause malloc() to fail (produces NULL)
  - Luck: control write offset from NULL pointer
  - Damage table used to validate Javascript integrity
  - Provide malicious script code
  - Win!!

Dowd's Whitepaper: http://documents.iss.net/whitepapers/IBM_X-Force_WP_final.pdf

Key Management

- Achilles heel of most crypto libraries:
  - Keys stored on the heap
  - You'll find them in the swapfile
  - Finding keys is an art in and of itself:

Key management

- Worse:
  - Putting your keys in a database
  - Especially when it's shared...
  - This actually happens!

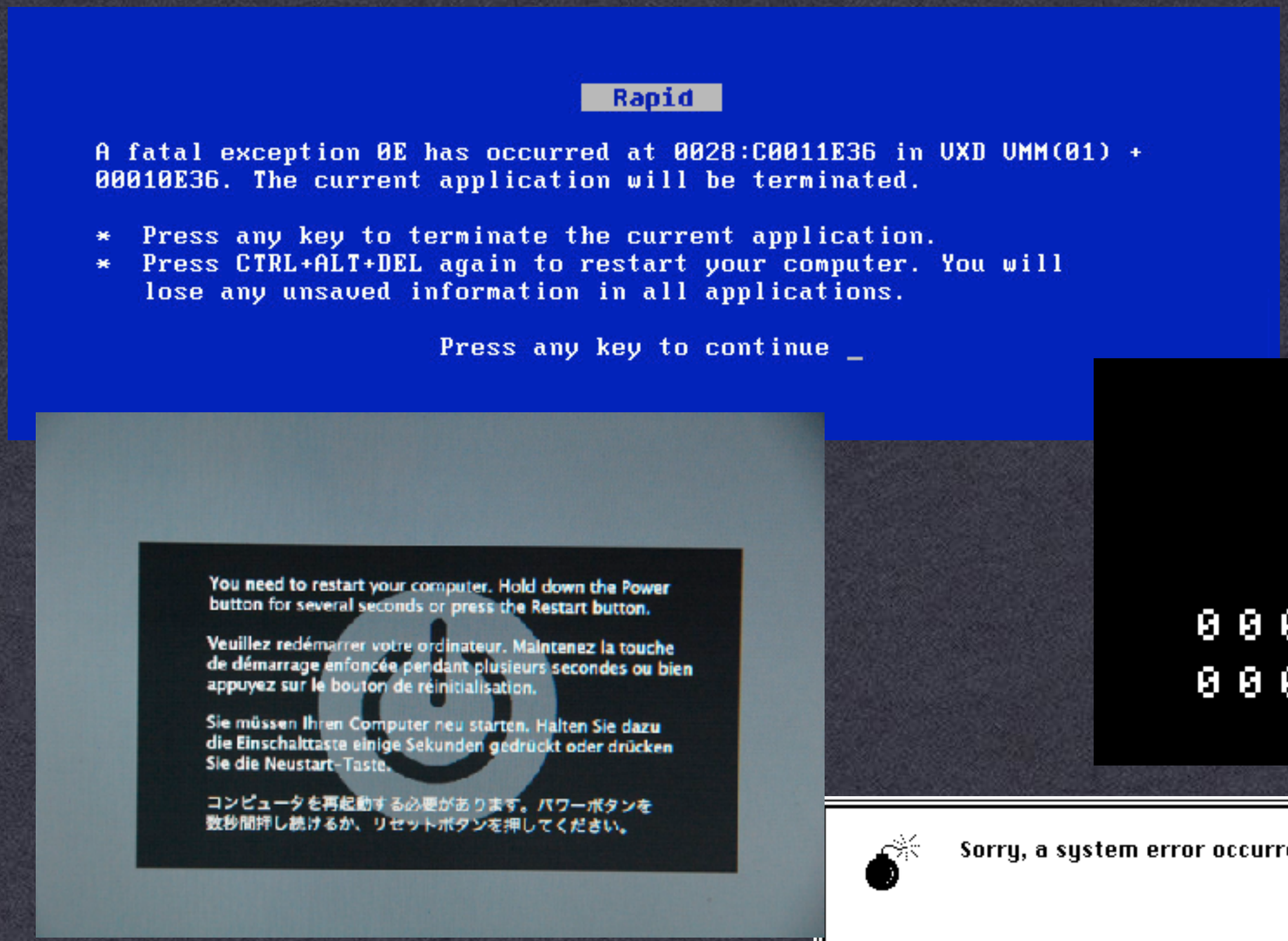
Next time:

- Hardware Security
  - Physical security
  - Tamper-resistance
  - Tamper-evidence
  - Emissions security & side-channel attacks

Older systems & embedded devices

Buffer Overflow

- Buffers have limite


Rapid
A fatal exception 0E has occurred at 0028:C0011E36 in VXD VMM(01) +
00010E36. The current application will be terminated.
* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue _

You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。


0 0 0 0 0 0 0 F
0 0 0 0 0 0 0 3

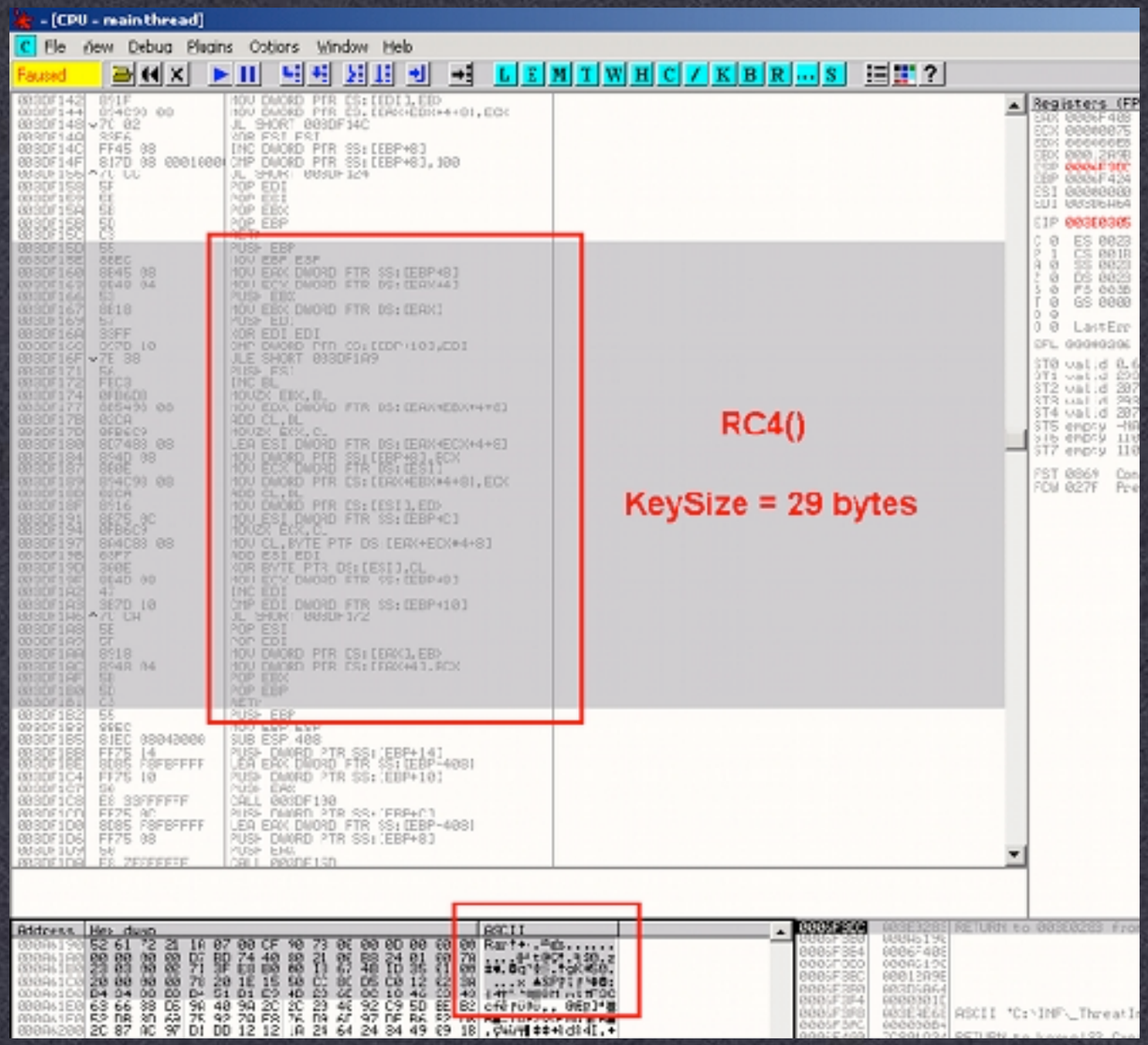

Sorry, a system error occurred.
Restart

Worms

Current Event

- Downadup (aka Conficker, Kido)
  - 9 million infected hosts (1/2009)
  - Spread by network or removable storage
  - Polymorphic components (RC4 encryption)
  - Domain generator (PRNG)
  - Software update mechanism (signatures?)

Source: Symantec. https://forums.symantec.com/t5/Malicious-Code/Downadup-Small-Improvements-Yield-Big-Returns/ba-p/381717


- [CPU - main thread]
File View Debug Plugins Options Window Help
Paused
RC4()
KeySize = 29 bytes