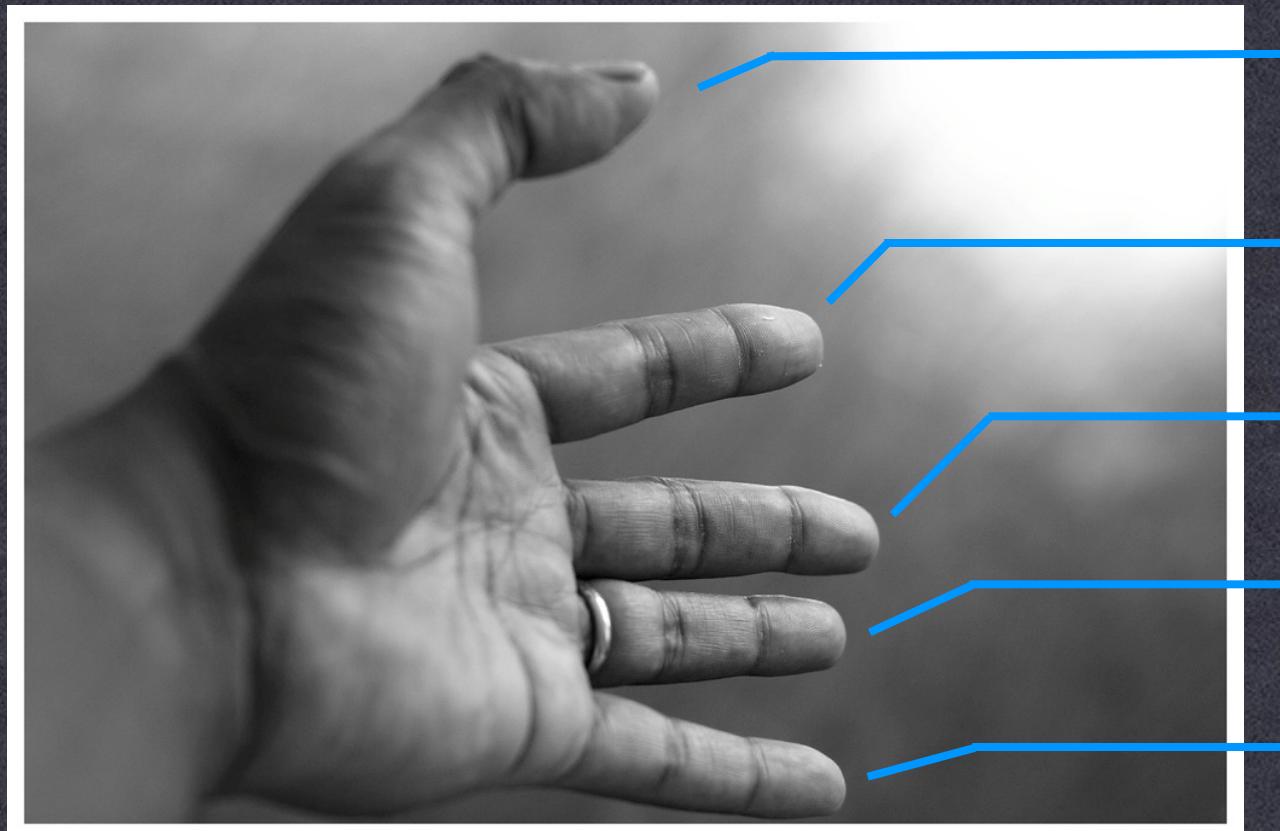


650.445: Practical Cryptographic Systems Software Vulnerabilities

Many, many slides graciously donated by Lucas Ballard!

Instructor: Matthew Green

Review



Concept

Primitives

Protocols

Implementation

Usage

Review



Concept

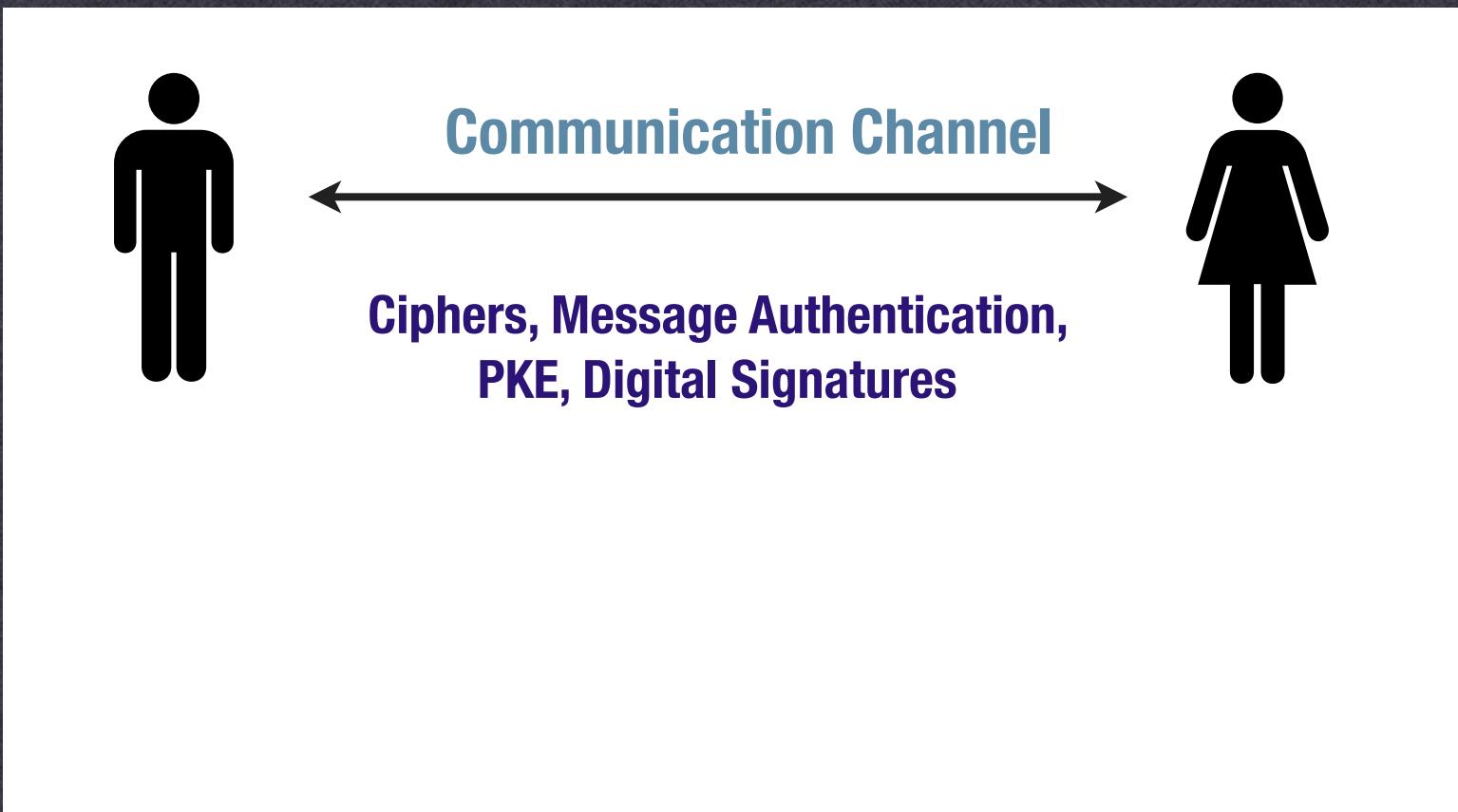
Primitives

Protocols

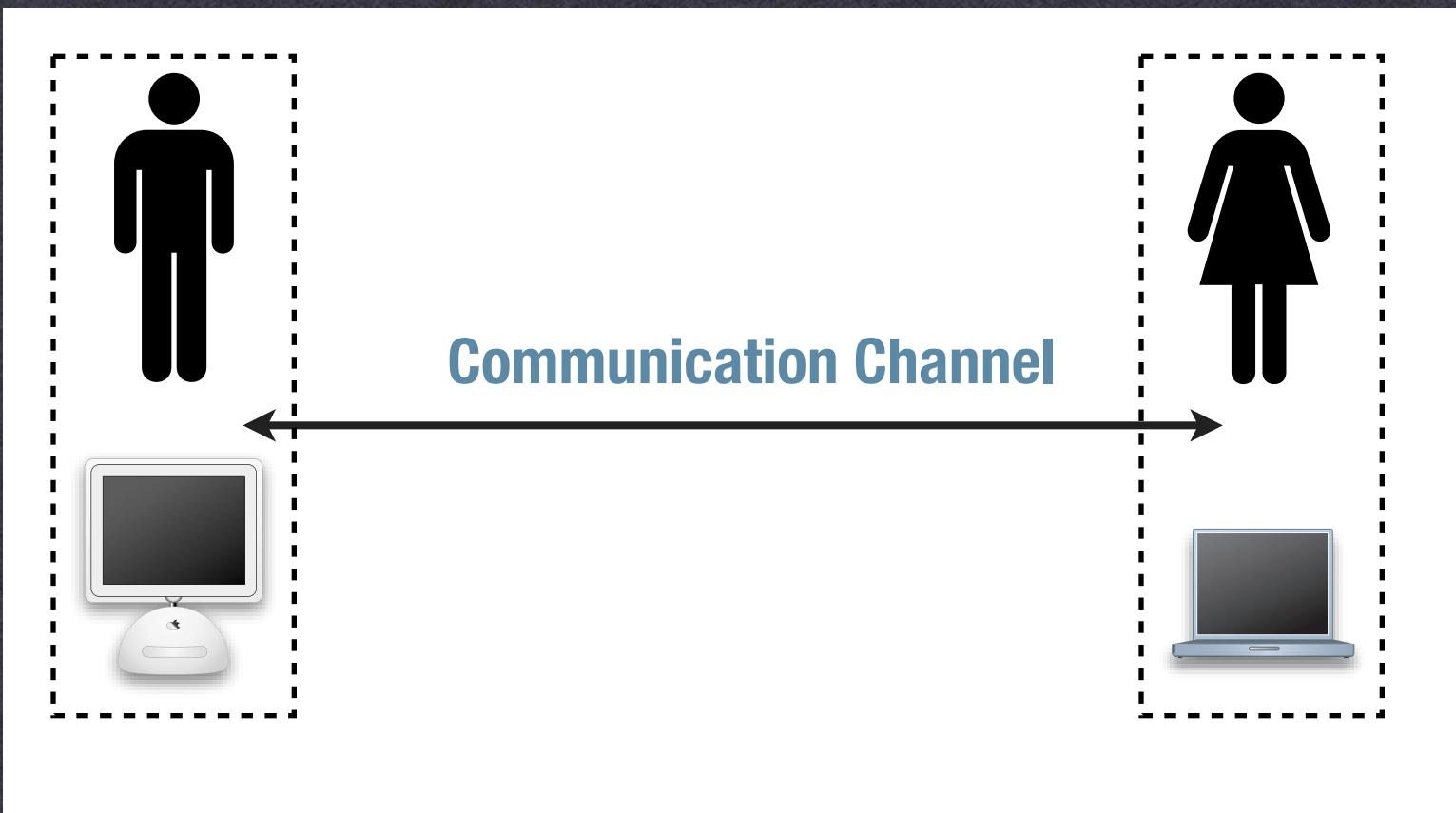
Implementation

Usage

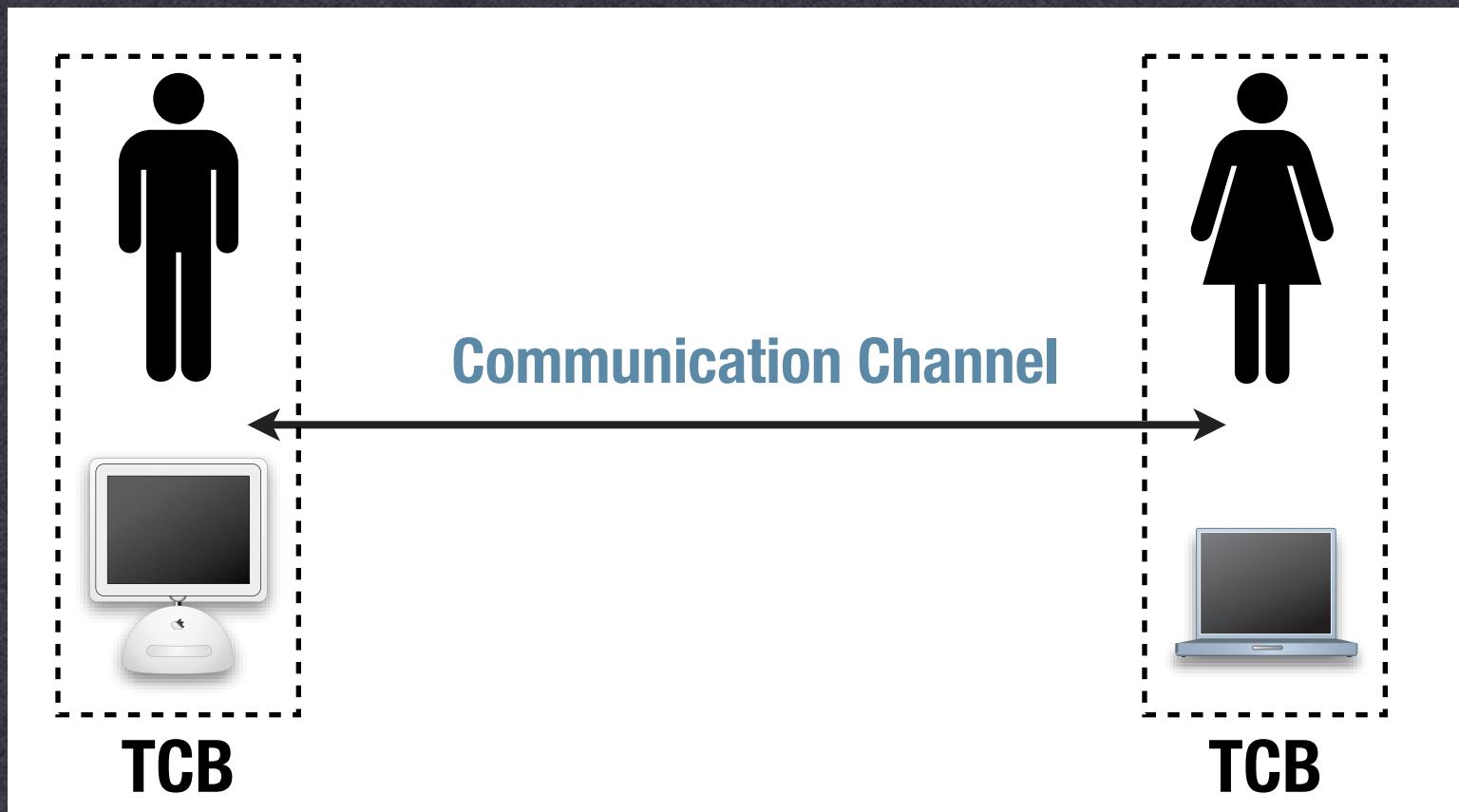
Review



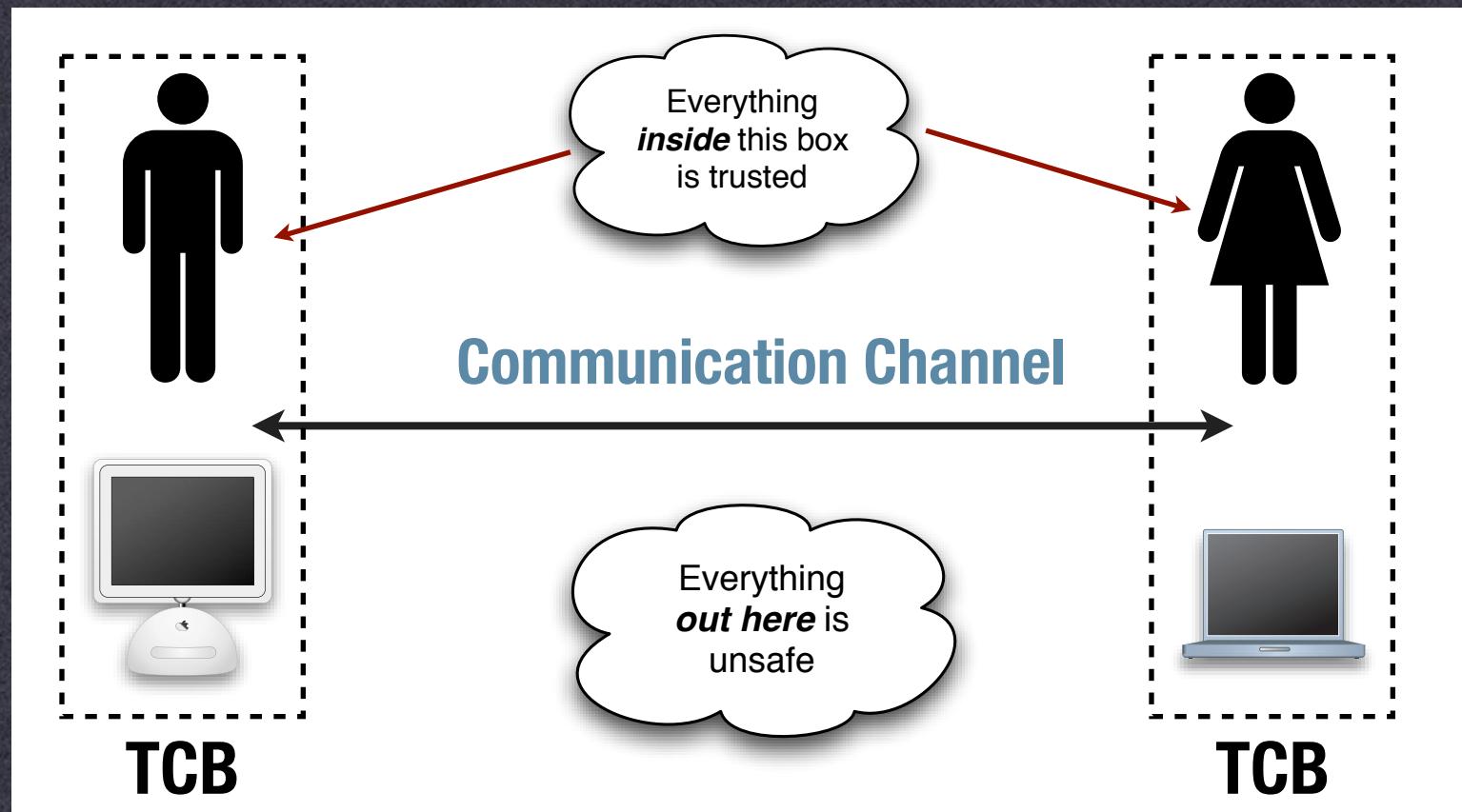
Review



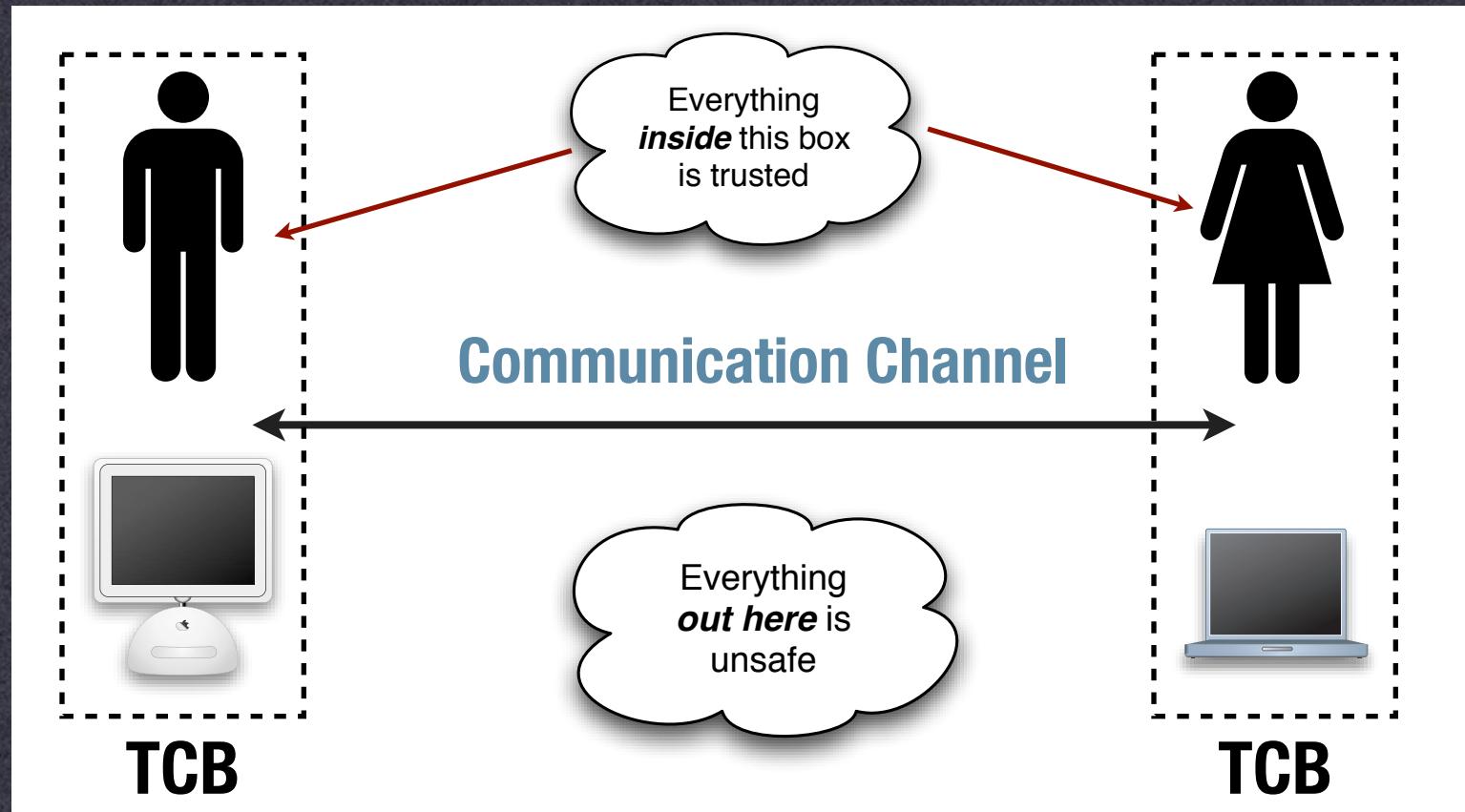
Review



Review



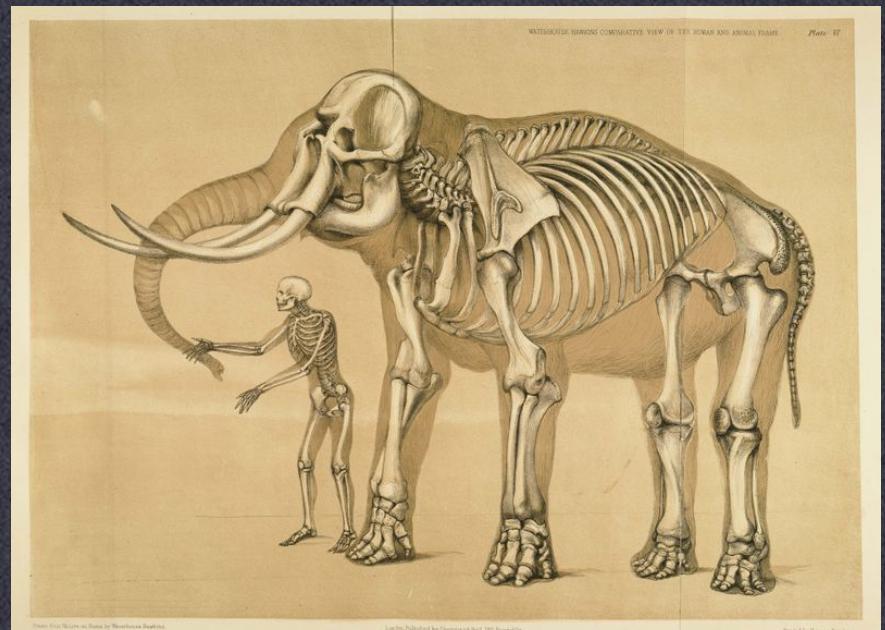
Review



- Just because something's “trusted”... doesn’t mean that it’s trustworthy.

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.

[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

September 6, 2002 6:39 AM PDT

Credit card theft feared in Windows flaw

By Joe Wilcox
Staff Writer, CNET News

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

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

Security News:

OpenSSL overflowing with buffer problems

By Edward Hurley, News Writers
01 Aug 2002 | SearchSecurity

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

 Post a comment

CONSOLE HACKING 2008: WII FAIL

Is implementation the enemy of design?

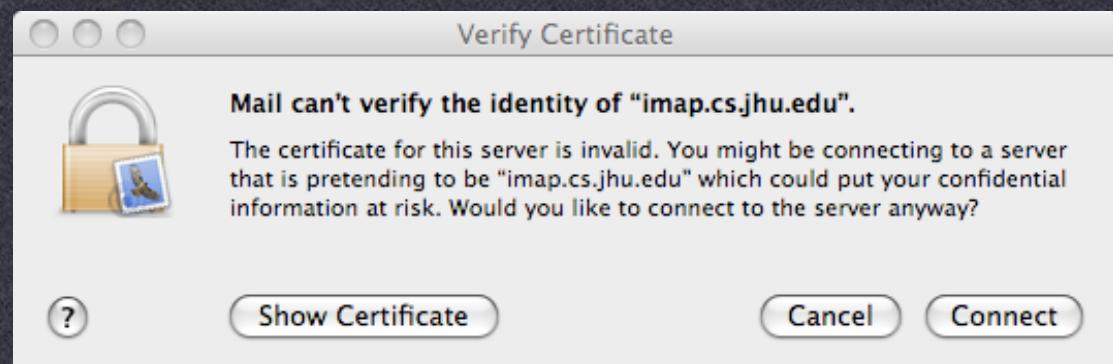
marcan and bushing
Team Twiizers

Today

- How software 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

$$N = p \cdot q$$

Signing

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

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

$$pk = (e, N)$$

$$sk = d$$

Algorithm implementation

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

0x00 0x01

Fixed Padding

0x00 H(Message)

~ 1024 bits (128 bytes)

Algorithm implementation

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

0x00 0x01

Fixed Padding

0x00 H(Message)

~ 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?



0x00 0x01 Fixed Pad 0x00 H(Message)

~ 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:

0x00 0x01

Fixed Padding

0x00

H(Message)

strcmp()

Oops

0x00 0x01

Fixed Padding

0x00

H(Message)

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:

0x00 0x01

Fixed Padding

0x00

H(Message)

`memcmp()`

0x00 0x01

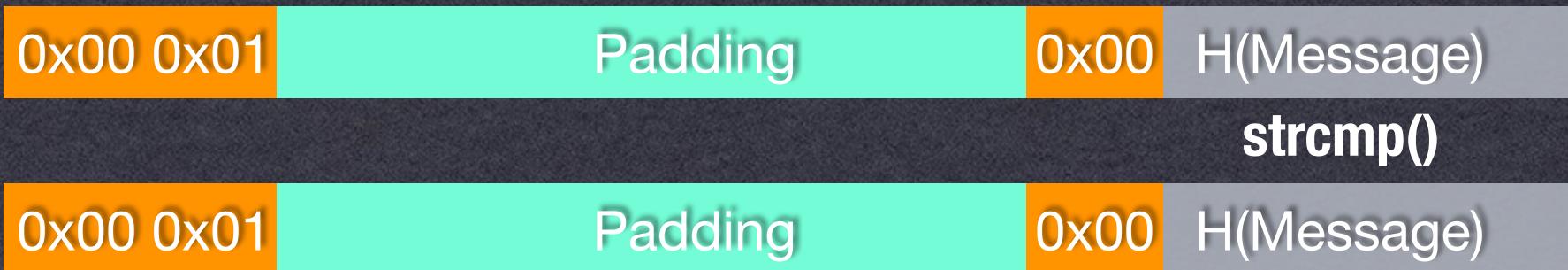
Fixed Padding

0x00

H(Message)

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?

CONSOLE HACKING 2008: WII FAIL

Is implementation the enemy of design?

marcan and bus
Team Twiizers



WiiFreeloader

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

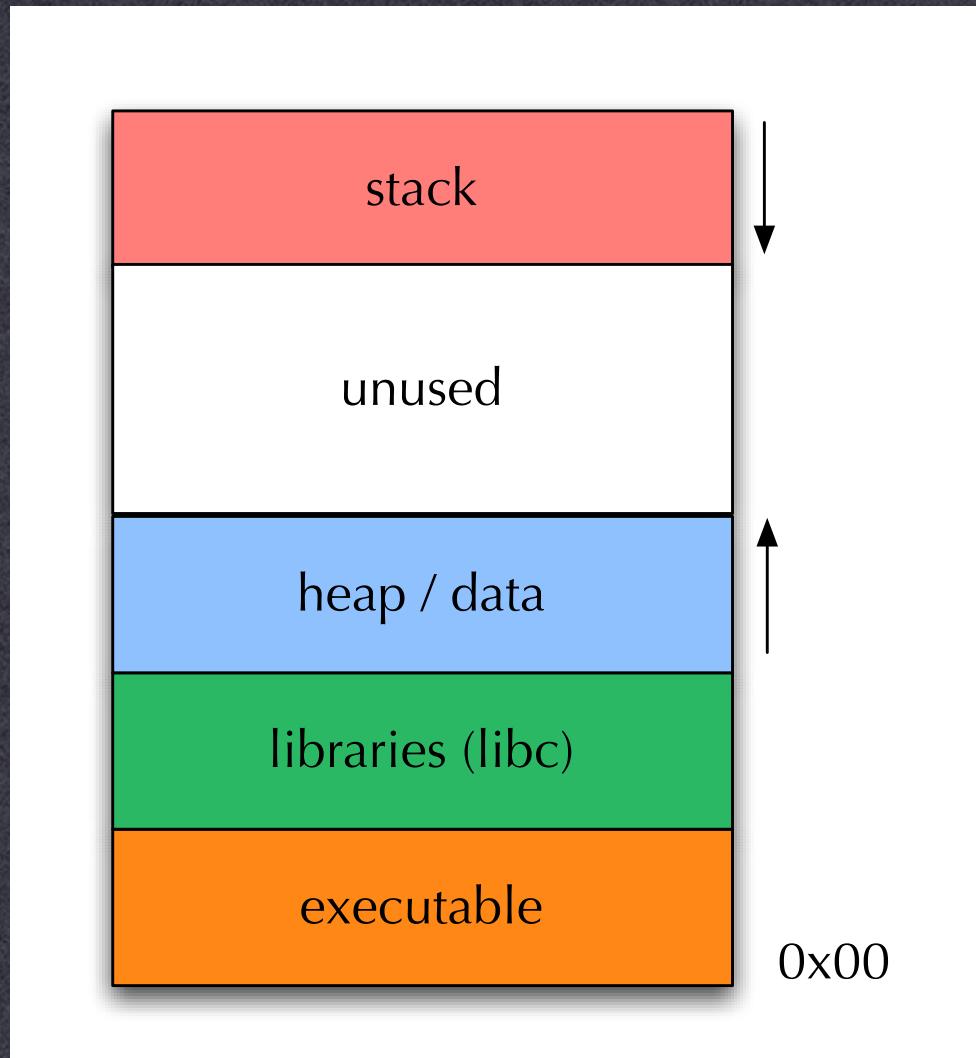
- Most cryptographic software written in C/C++
 - C++: efficiency > security
 - No runtime bounds checking, limited type safety
 - Results in memory exploits
 - These can be exploited remotely!

Stack-based Exploits

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



Memory



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: BUFFER 1
		Your social security number: 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
Rachelle Posner Deacons-Smith		Your social security number
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		2(a)
a. \$26,000 for married filing jointly		2(b)
b. \$13,000 for single		2(c)
c. \$17,000 for single head of household		
3. Number of dependents, not including spouse		3

Example

```
#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]);
}
```

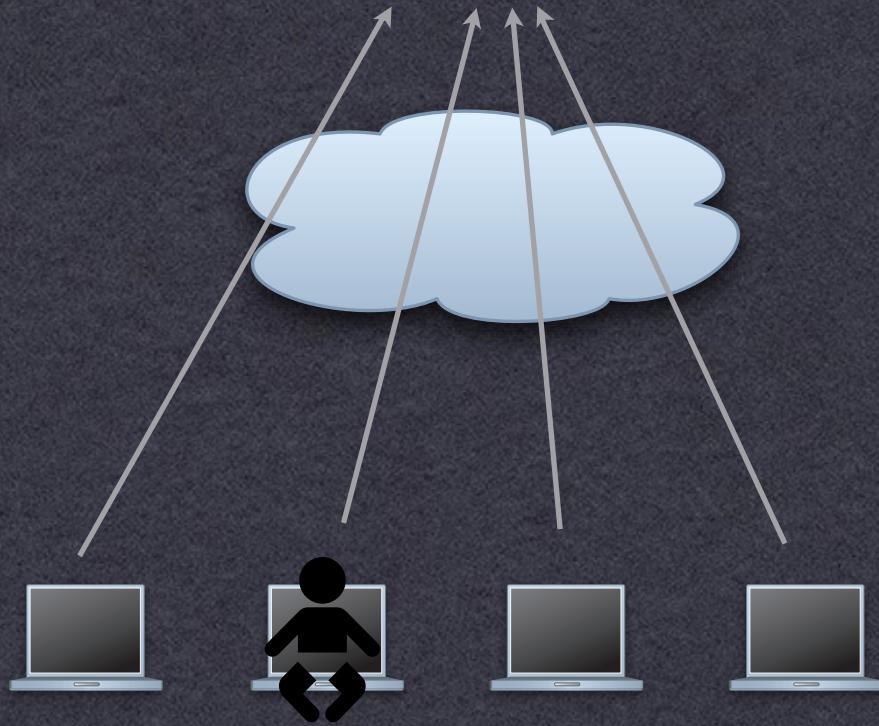
```
bash-3.2$ ./test nothing_wrong
nothing_wrong
```

```
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...?



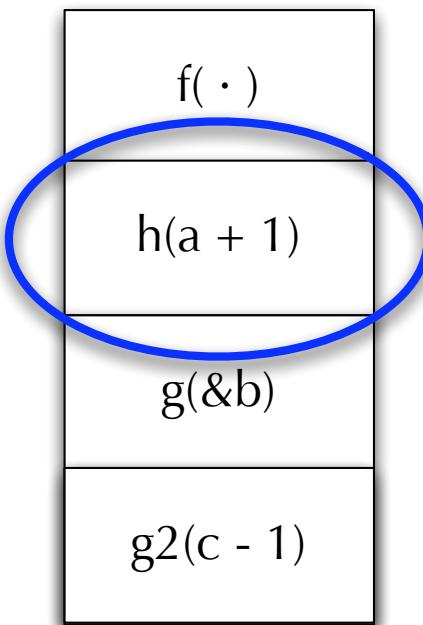
The Stack

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

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

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

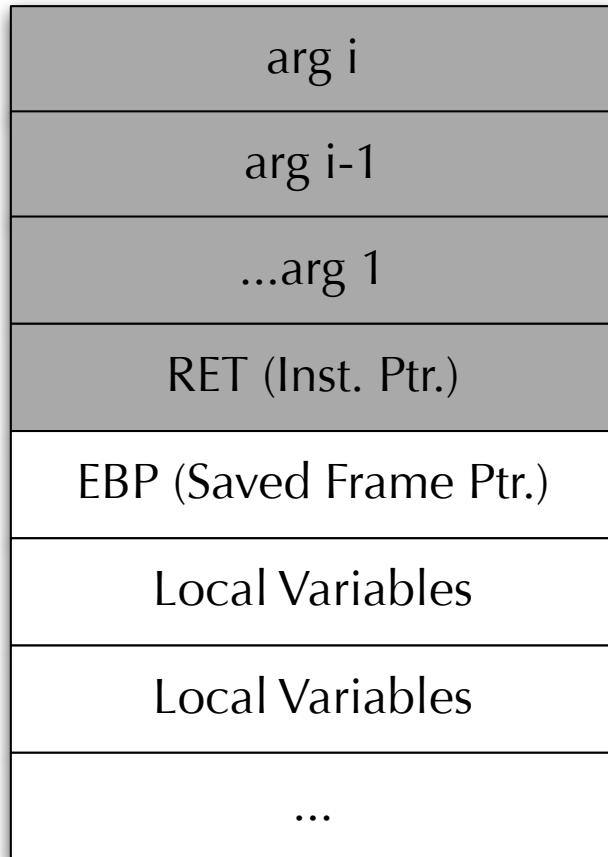
Stack Frames



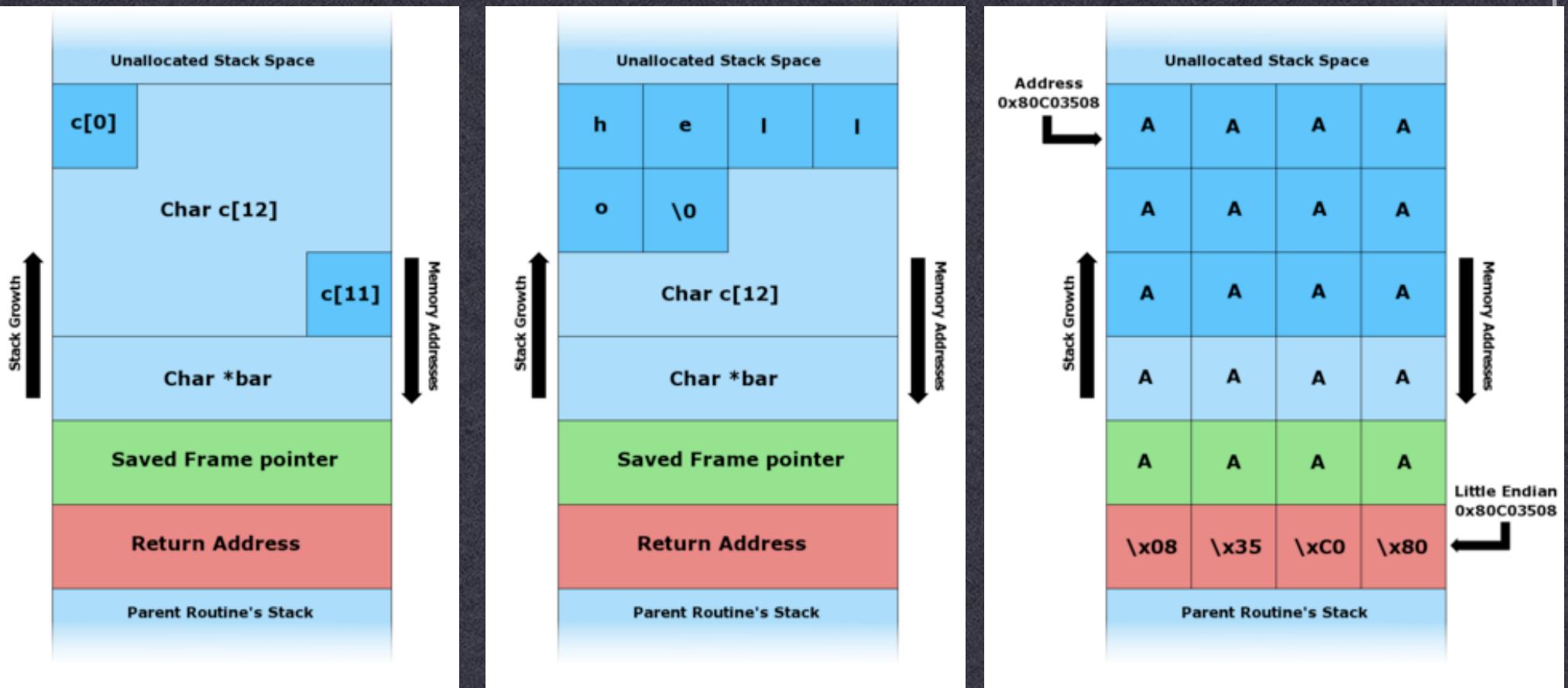
Stack Frame

Caller

Callee

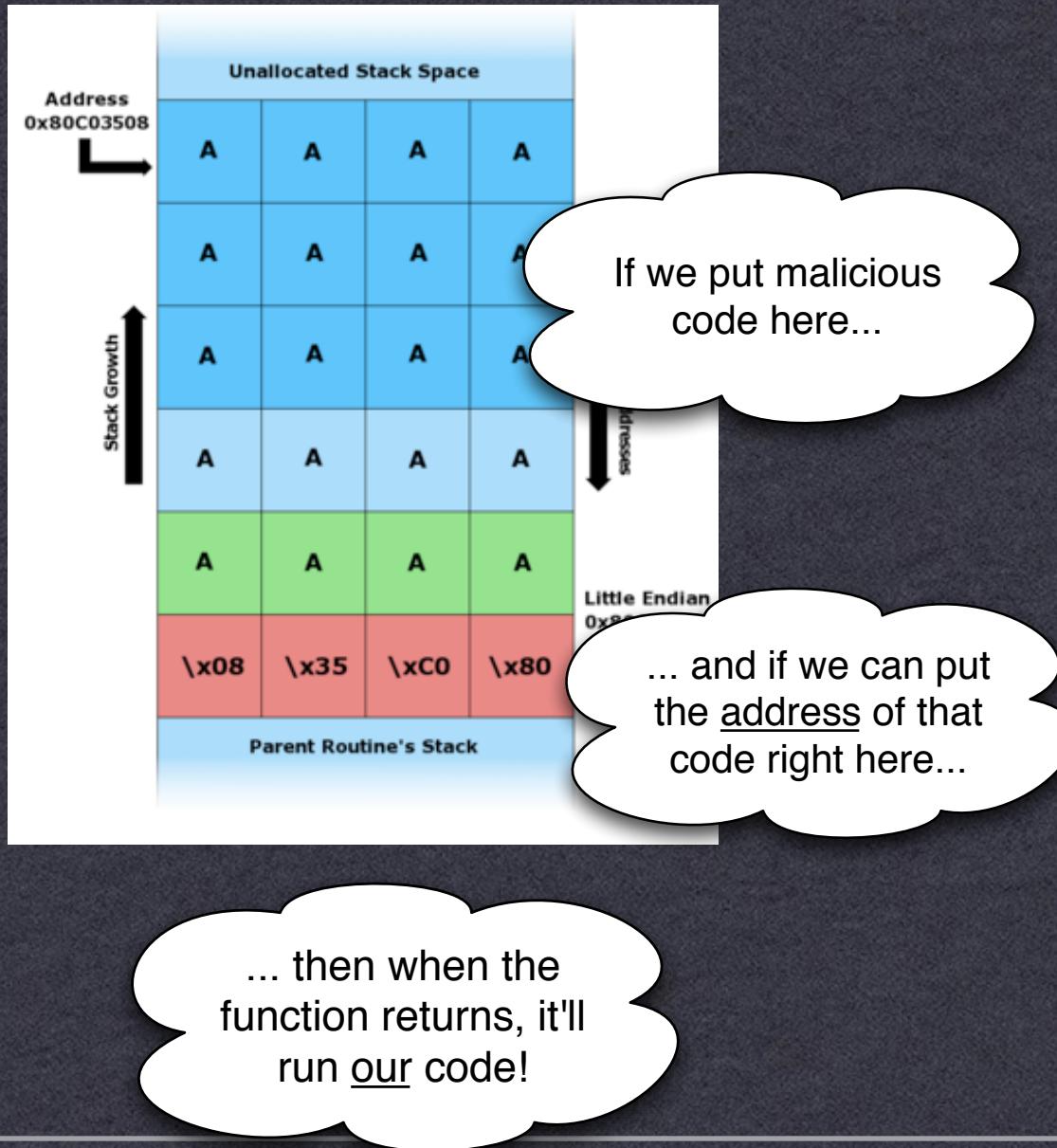


“Stack smashing”



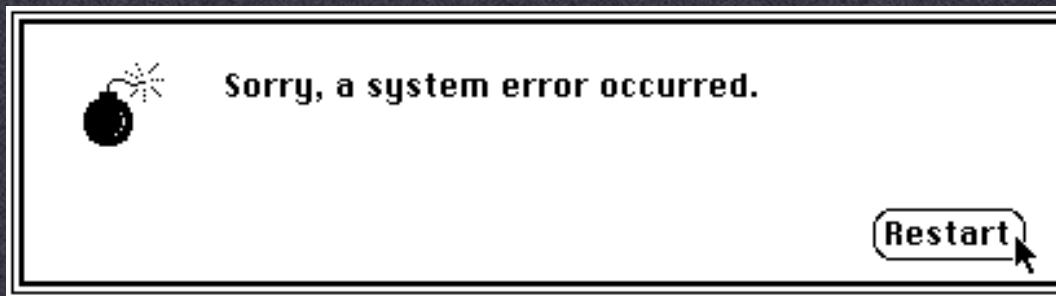
Public domain image courtesy Michael Lynn, Wikipedia (Stack Overflow).

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



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)

bullets - 2
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 backdoor
pass a command to the shell

C Library (libc, -lc) , exec* ()

• `<stdlib.h>` Modify “arguments” in addition to

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(...); ...`

Format strings

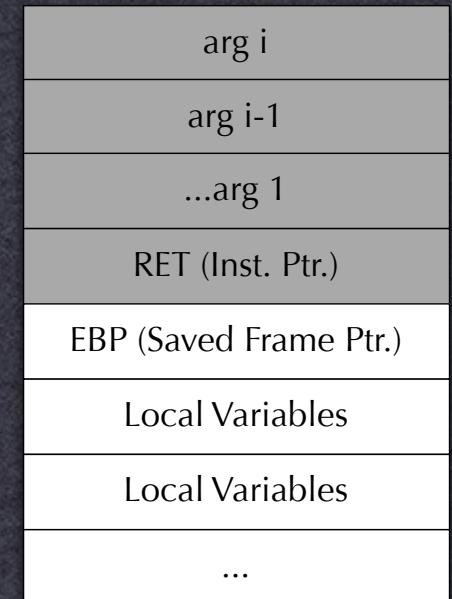
- Standard functions (**printf**, **sprintf**, **snprintf**)
 - Accept “format strings”

```
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 ("%os \n", "A string");
```



Reading the stack

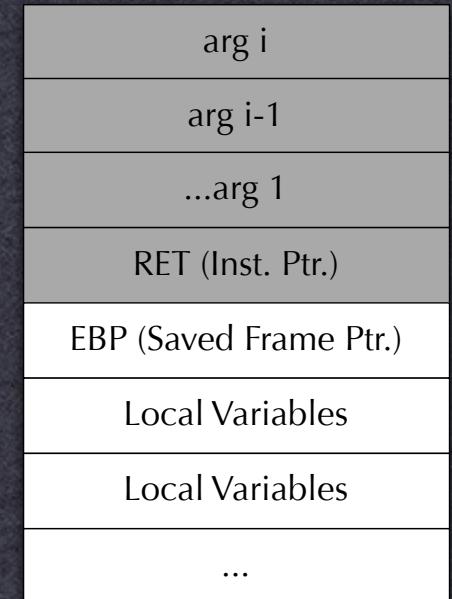
```
int parse_user_supplied_buffer(buffer)
{
    printf(localbuf, buffer);
    ...
}
```



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

Reading arbitrary memory

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

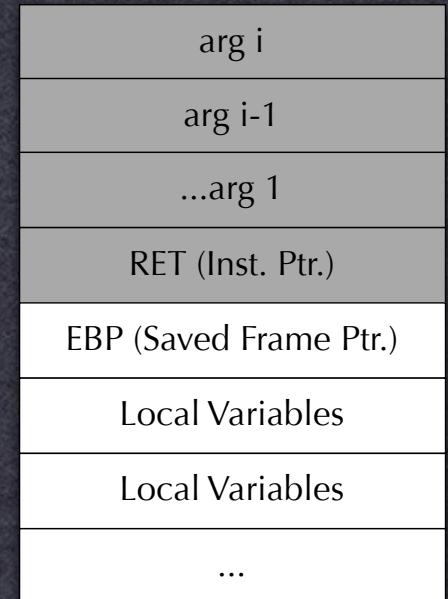


```
> ./program
“\x23\x22\x0A\x43_%08x.%08x.%08x.%08x.%08x|\%s|”
hello 023uhdfh..a,sCRYPTOGRAPHICKEYkaduueuj
```

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

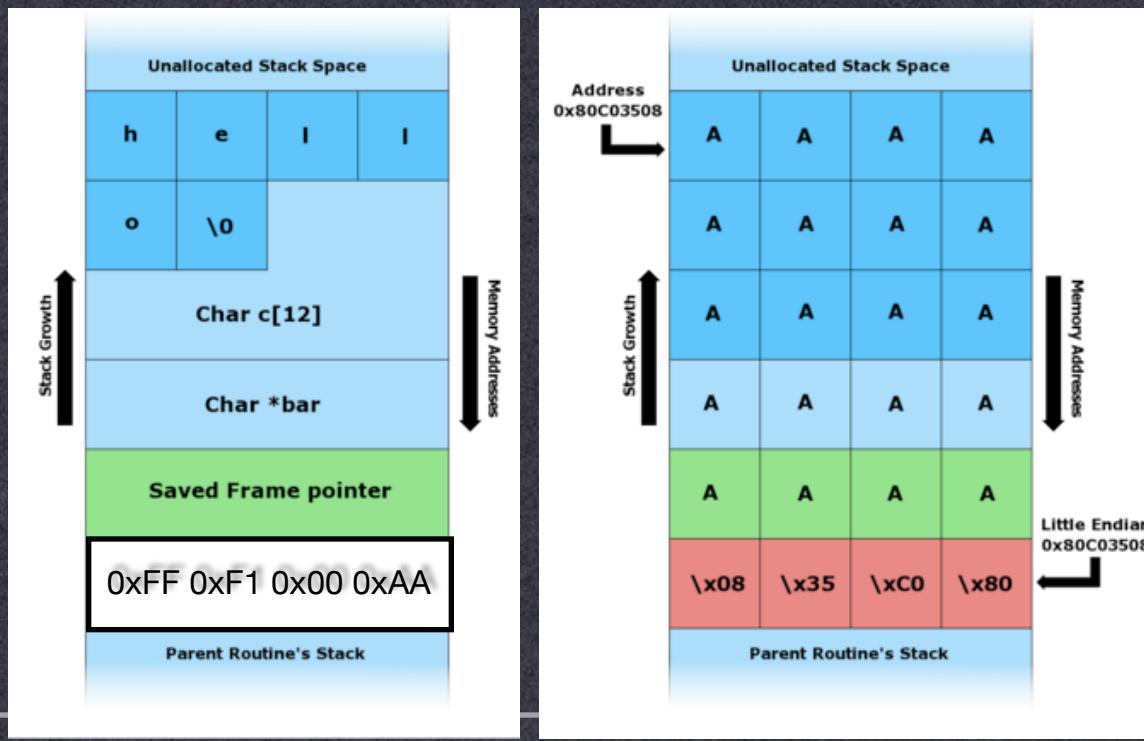
Unsigned
usage

How to fix this?

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

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!!!

Shadow Stacks

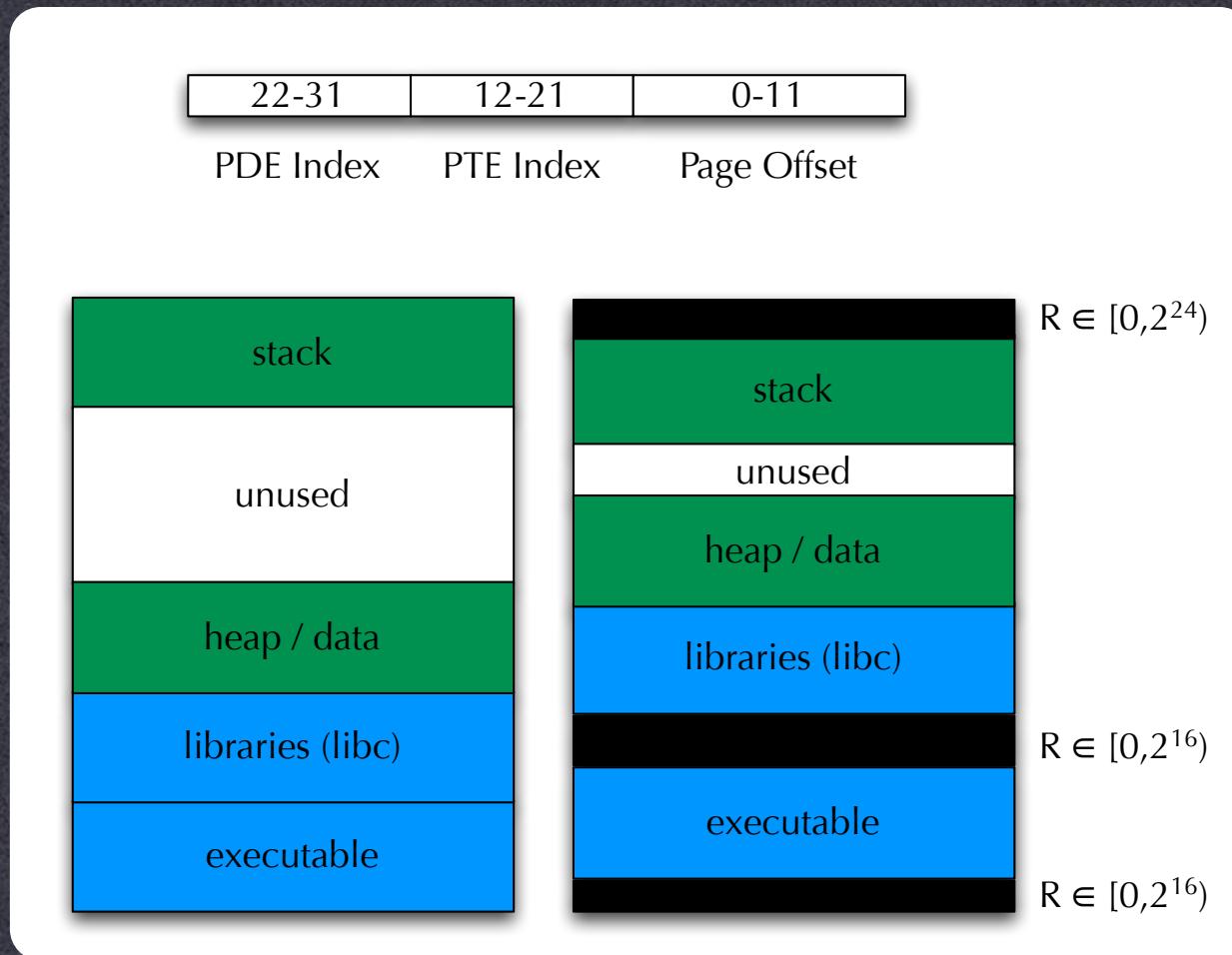
- Keep an extra copy of return address in Kernel Memory
 - Only return if addresses match up
 - Doesn't protect other memory, registers, etc.

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



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

Solutions

- Use a safe language...?
 - Java, Ruby, etc.
 - Enforce bounds checking, garbage collection
 - Type safety
 - Don't ever 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!!



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!