WORDPRESS INFECTIONS LEADING TO TESLACRYPT RANSOMWARE

by **Michael Mimoso** Follow @mike_mimoso

February 5, 2016 , 7:00 am

Website operators running sites on the WordPress platform need to be aware of a massive string of infections that as of Thursday were poorly detected by security products.

Researchers at Heimdal Security said the compromised sites redirect victims to other domains hosting the Nuclear Exploit Kit, a potent collection of exploits for vulnerable Adobe products (Flash, Reader, Acrobat), Internet Explorer and Microsoft Silverlight, that has in the past, and in this case, been dropping ransomware on infected computers.

Sucuri said the infections it saw were characterized by encrypted malicious code appended to the end of all legitimate JavaScript files. These infections hit only first-time visitors to the compromised sites and sets a cookie that expires within 24 hours and injects and invisible iFrame with “Admedia” or “advertising” in the path part of the URL, Sucuri said.

cs642

computer security

memory *protection*

adam everspaugh
ace@cs.wisc.edu

vulnerabilities

* buffer overflow
* stack smashing
* heap overflow
* function pointer overwrite
* double-free
* printf format string vulnerabilities

principles

Principles of Secure Designs

* Compartmentalization
  / Isolation
  / Least privilege

* Defense-in-depth
  / Use more than one security mechanism
  / Secure the weakest length
  / Fail securely

* Keep it simple
  / Economy of mechanism
  / Psychological acceptability
  / Good defaults

* Open Design

How can we defend against these attacks?

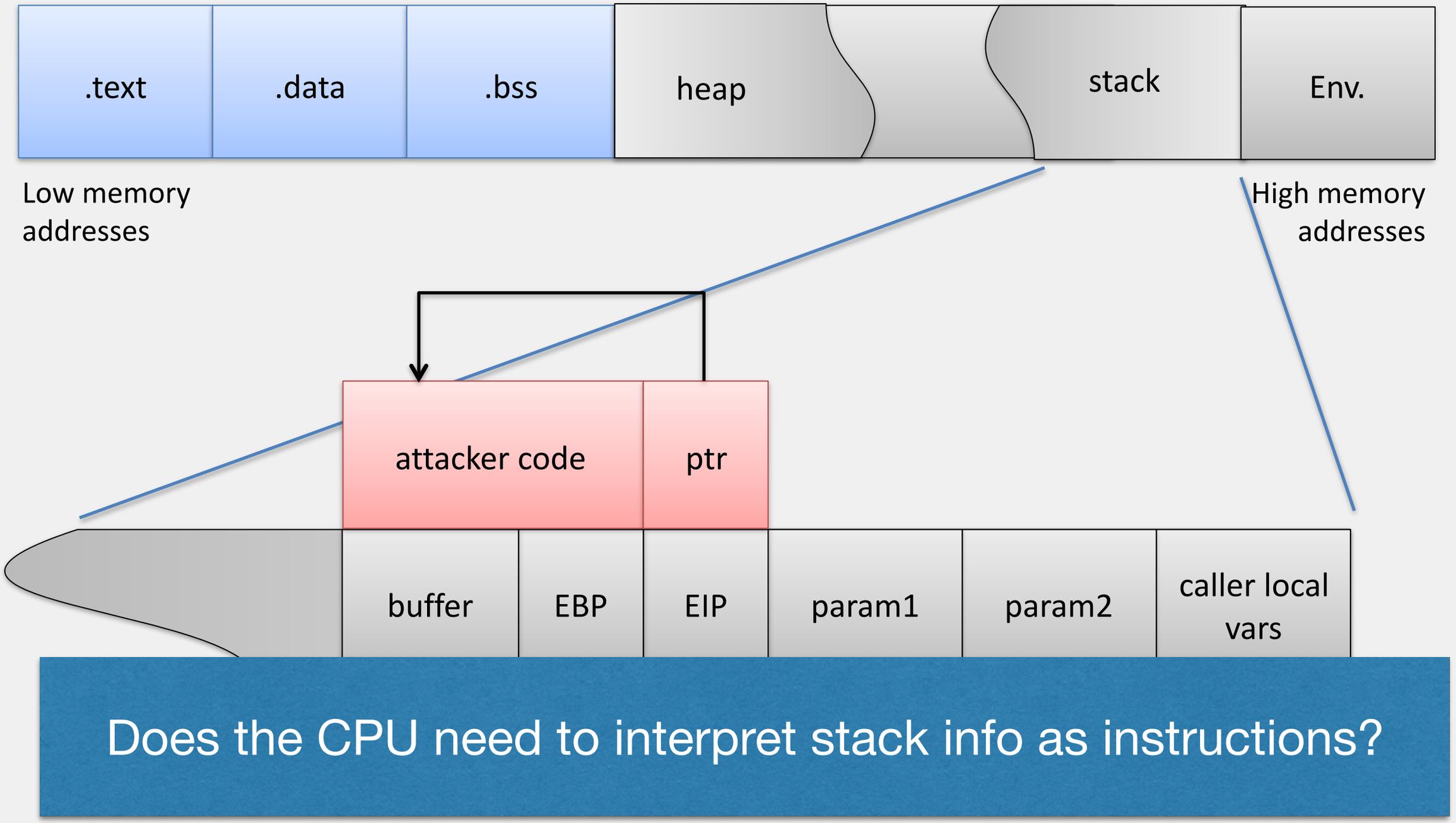
today

* Defenses (and counter-attacks)

* Data execution prevention
  / return-into-libc
  / return-oriented programming

* Address space layout randomization
  / and counter-attacks

* Sandboxing



eip overwrite exploit

dep

* Data Execution Prevention (DEP)
  / Mark memory pages containing writable data
    as **"no execute"**

* Which pages?
  / Data pages in: heap, stack, .bss, .text, env, pages

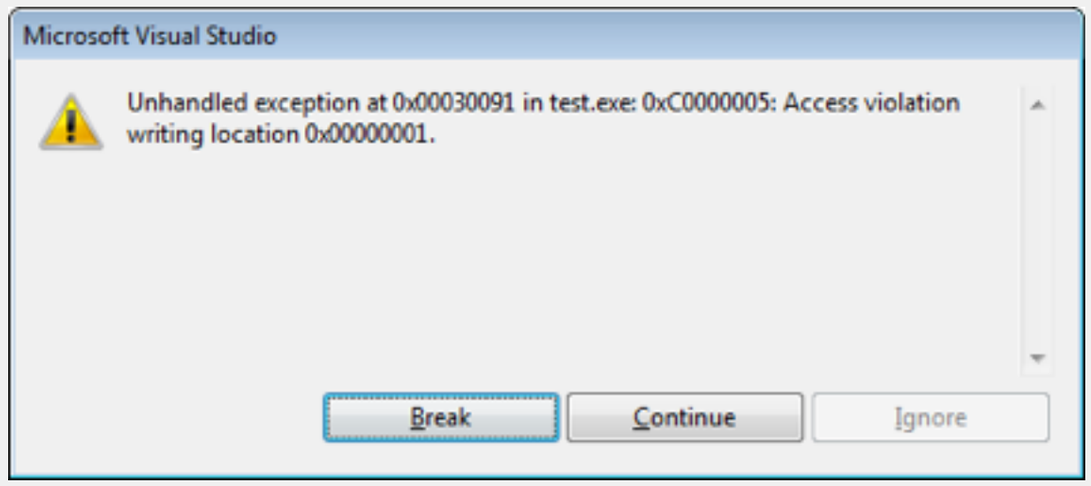
* Hardware support -- extra bit in page table entry
  / Intel x86 -- XD bit (execute disabled)
  / AMD x86 -- NX bit (no execute)
  / ARM -- XN bit (execute never)

dep limitations

* Problems?
  / Breaks some existing applications and libraries

* Just-in-time (JIT) compilers
  / Microsoft's .NET framework
  / Java

* If any dependent library in an application crashes with DEP enabled, what's the app developer going to do?



timeline

* Solaris/Sparc non-executable stack: 1997
* AMD NX bit - Athlon/Opteron - 2003
* Intel XD bit - Pentium 4 - 2004
* Windows XP Service Pack 2 supports DEP - August 2004
* 19 years later -- DEP is quite effective

effectiveness

think-*pair*-share

Does DEP prevent:

* Overwriting EBP/EIP on stack? No
* AlephOne's stack smashing attack? Yes
* Stack smashing that points to shellcode in heap or Env? Yes
* Double-free exploit with shellcode in same location? Yes

defeating dep

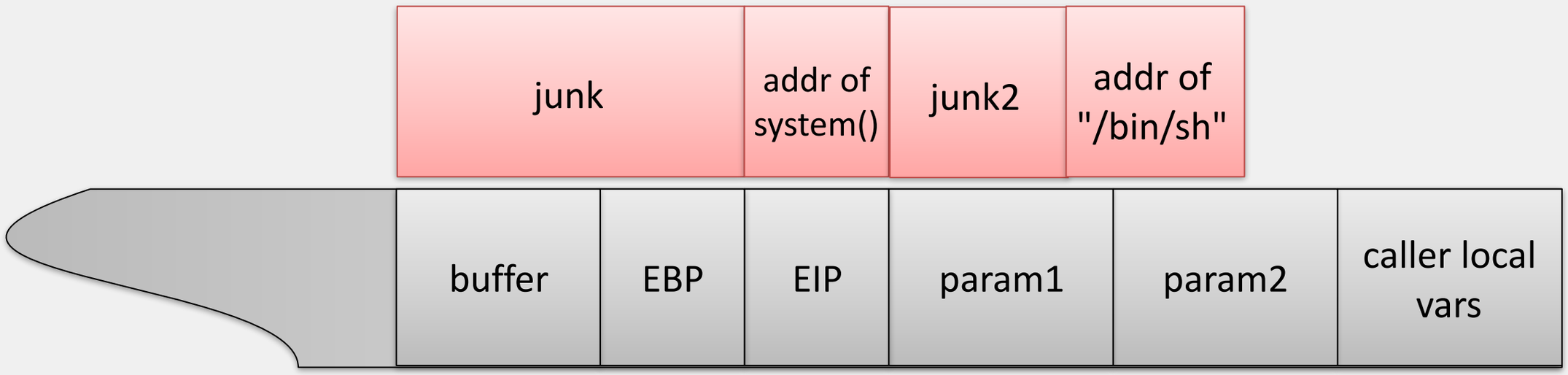
How does an adversary defeat DEP?

Must use existing memory pages containing code

libc -- standard c library

included in all processes

Contains system() -- runs commands



Overwrite EIP with address of system() function

junk2 => nonsense EBP/EIP "saved" on stack

parameter to system() is ptr to "/bin/sh"

Problem: system() drops privileges first.

return-into-libc

rop

* Return-oriented programming (ROP)
* Chaining together multiple return-into-libc calls
* Enables arbitrary code execution

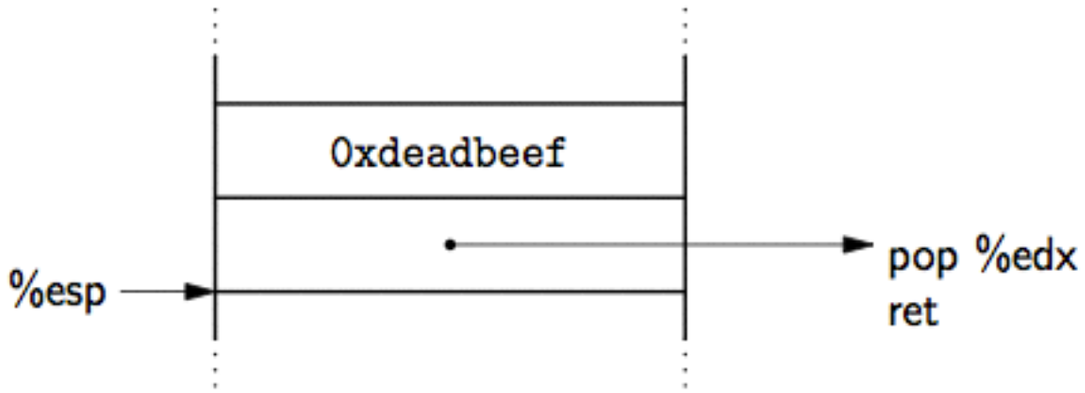


Figure 2: Load the constant `0xdeadbeef` into %edx.

[*The Geometry of Innocent Flesh on the Bone*, Schacam]

rop

[not covered in lecture - fyi only]

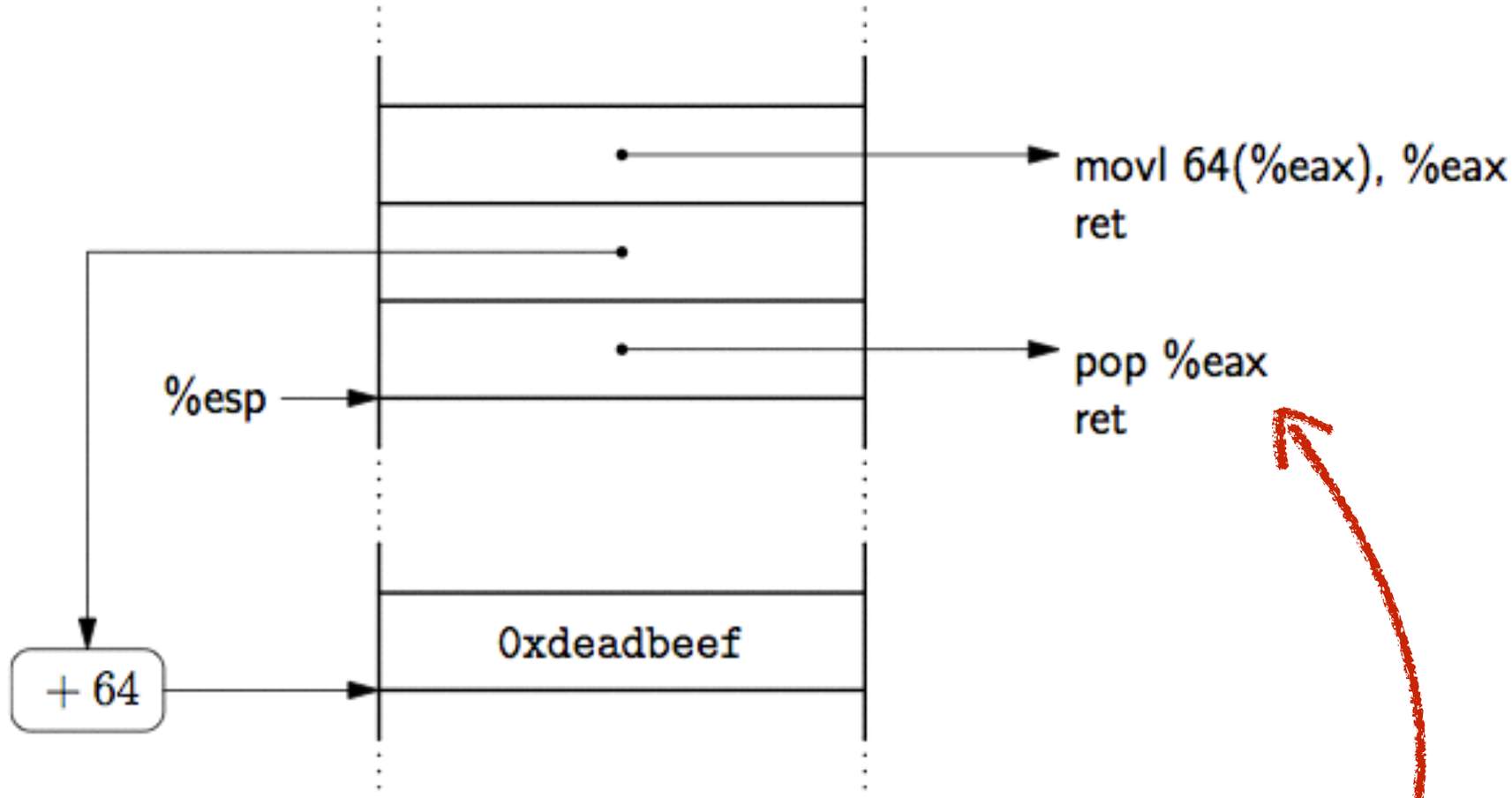


Figure 3: Load a word in memory into %eax.

rop gadgets: short sequences of instructions

[*The Geometry of Innocent Flesh on the Bone*, Schacam]

[*The Geometry of Innocent Flesh on the Bone*, Schacam]

[not covered in lecture - fyi only]

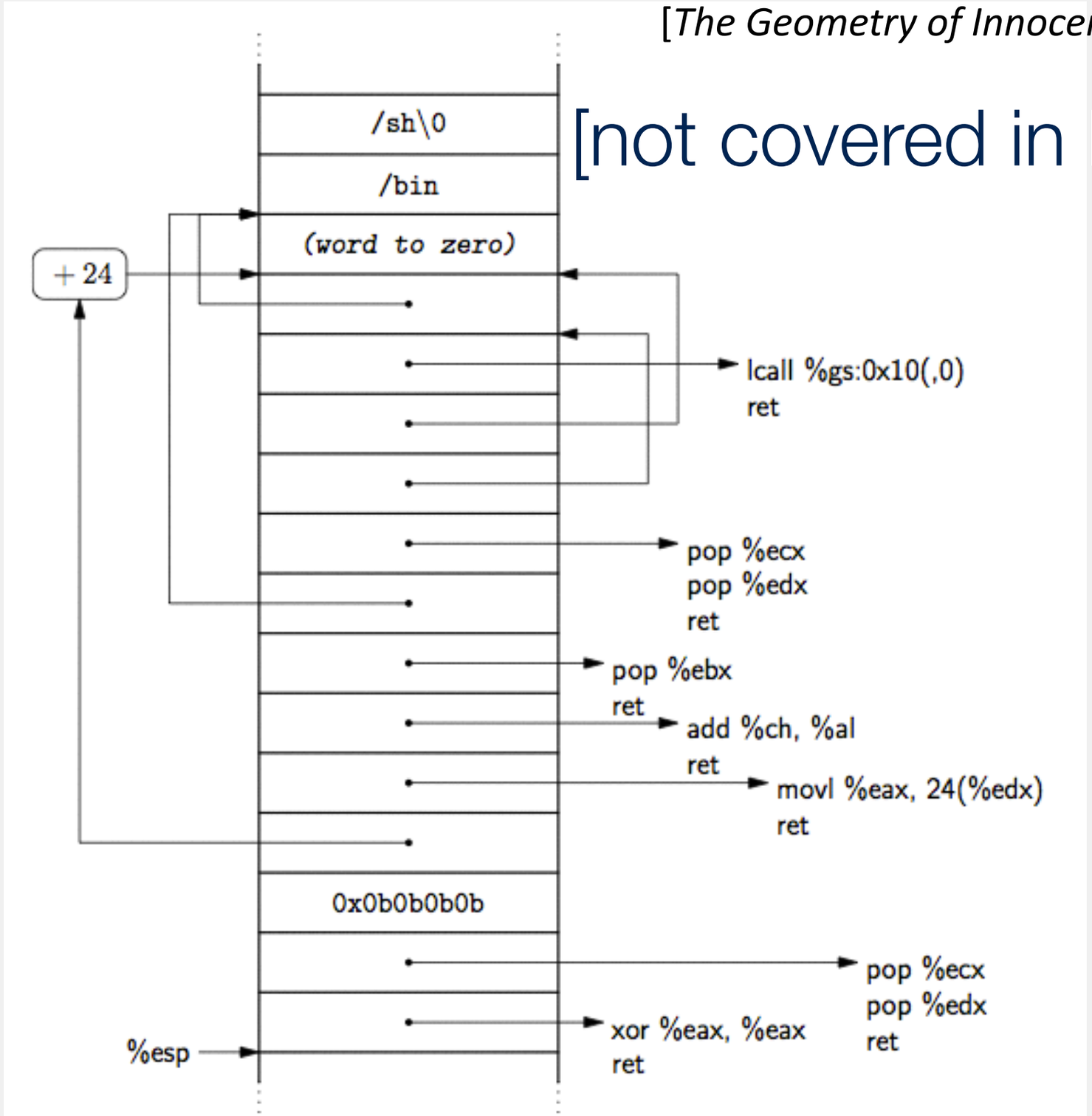


Figure 16: Shellcode.

rop shellcode

Intended instruction sequence

[not covered in lecture - fyi only]

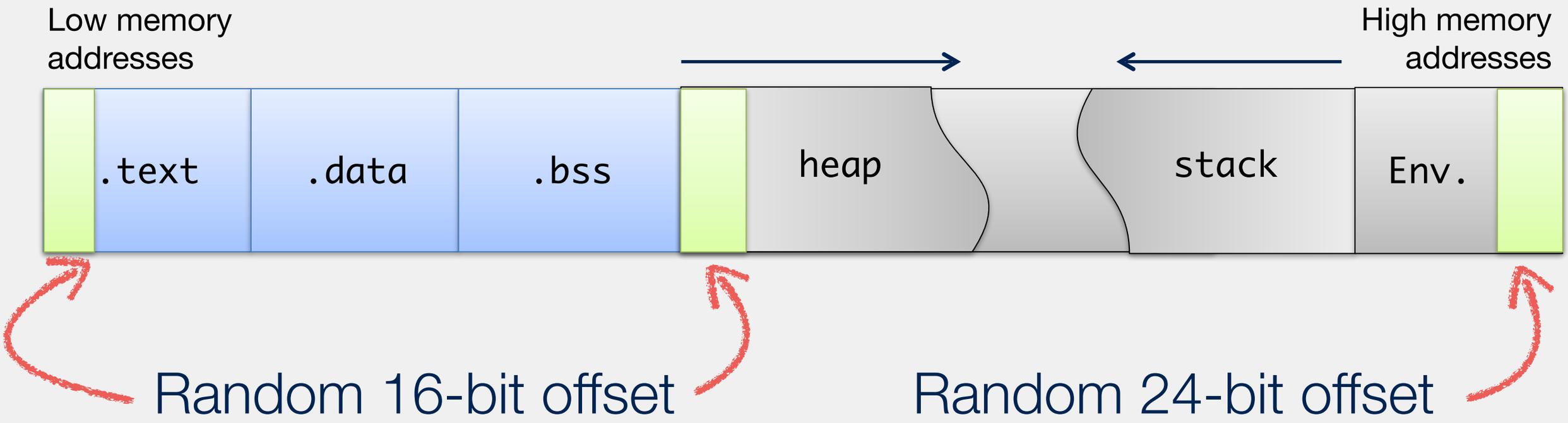
Alternate instruction sequence

Intended instruction	Alternate instruction bytes	Alternate instruction
movl $0x00000001, -44(%ebp)	c7 45 d4 01 00 00 00	
	f7	add %dh, %bh
test $0x00000007, %edi	c7 07 00 00 00	
	0f	movl $0x0F000000, (%edi)
setnzb -61(%ebp)	95	xchg %ebp, %eax
	45	inc%ebp
	c3	ret

finding rop gadgets

* Data Execution Prevention
  / Prevents many basic attacks
  / Not guaranteed to prevent all attacks
  / Makes attacks much more difficult!

dep wrap-up



* Address space layout randomization (ASLR)

* Example: PaX implementation for Linux
  / Each time process is loaded, new offsets are selected at random

aslr

defeating aslr

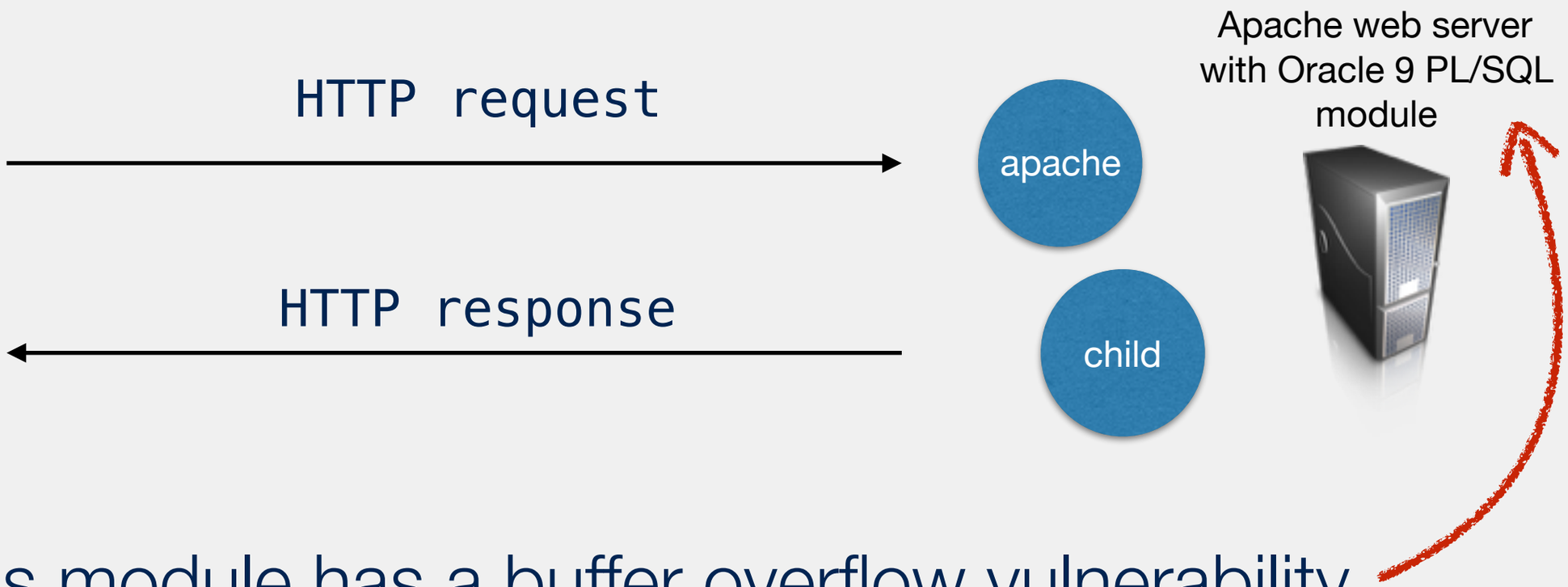
* How does an adversary defeat ASLR?

* Information disclosure that leaks offset
  / e.g. printf of arbitrary pointer

* Really large nop sled
  / How big?

* Brute force the offset

[*On the effectiveness of Address Space Layout Randomization*, Shacham et al.]



This module has a buffer overflow vulnerability,
but ASLR is enabled.

defeating aslr

[*On the effectiveness of Address Space Layout Randomization*, Shacham et al.]

HTTP request

Attacker guesses where usleep() is located in memory

top of stack (higher addresses)
⋮
0x01010101
0xDEADBEEF
guessed address of usleep()
0xDEADBEEF
64 byte buffer, now filled with A's
⋮
bottom of stack (lower addresses)

Figure 2: Stack after one probe

Apache web server with Oracle 9 PL/SQL module

Guess is wrong

Child process crashes, connection closes immediately

Guess is right

Child process sleeps for 0x01010101 usecs, then closes connection

defeating aslr

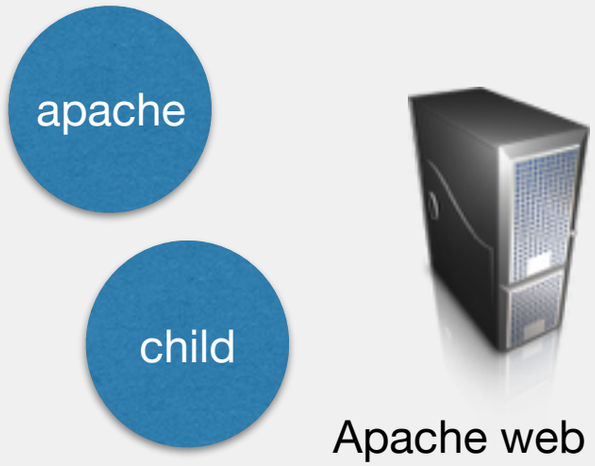
[*On the effectiveness of Address Space Layout Randomization*, Shacham et al.]

HTTP request

Attacker guesses where usleep() is located in memory

top of stack (higher addresses)
⋮
0x01010101
0xDEADBEEF
guessed address of usleep()
0xDEADBEEF
64 byte buffer, now filled with A's
⋮
bottom of stack (lower addresses)

Figure 2: Stack after one probe



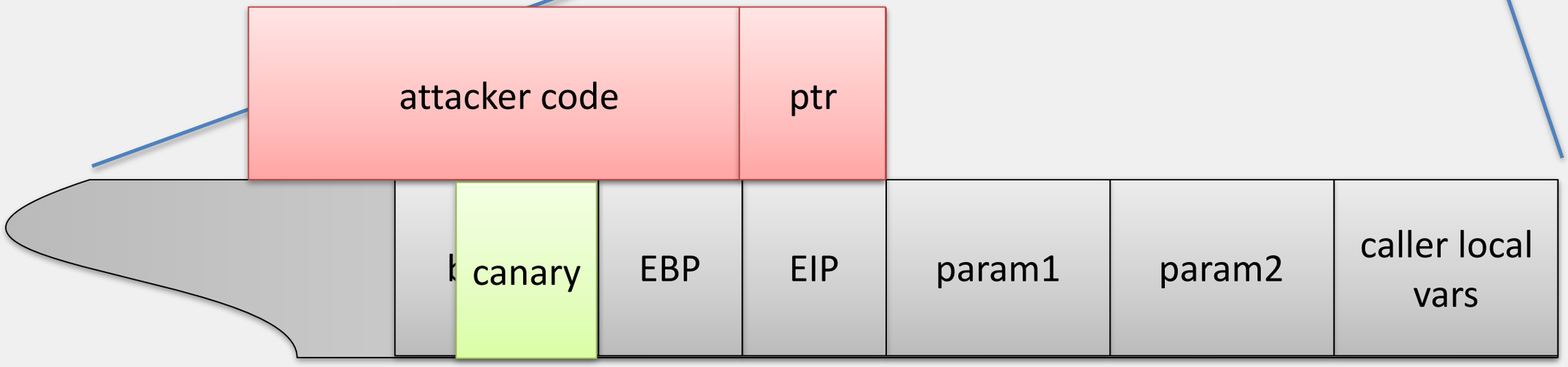
Apache web server with Oracle 9 PL/SQL module

How long does this take?

Assume:
/ address of usleep randomized with 16-bits
/ each request takes 70ms

defeating aslr

What does code do if canary value has been changed?
What are the costs?



Insert "canary value" between local variables and control data saved on stack

gcc stack protector => random value, same for entire process

Check canary before exiting function

stack protector

* DEP
* ASLR
* Stack protector

* Which of these prevent overwriting data on the stack?

* Which of these are (probably) in-place on a modern laptop, server, or smartphone?

defense recap

sandboxing

* Previous defenses make attacks harder, but are not perfect

* Even when they are effective, sometimes they get disabled

* What else can we do?

* Assume the worst and try to limit impact of compromised processes

* Confinement via sandboxing

{Threat model}

assets — attackers — vulnerabilities

attack vectors

{Security model}

subjects — trusted components

countermeasures — security goals

Why might web browsers be desirable targets?

What's involved in a typical web stack?

think-*pair*-share

web browsers

Browser
Manager

Trusted Process
Cookie, History, PW databases
User input
Window management
Location bar
Network stack
TLS
Download manager
Clipboard

google chrome browser

As few permissions as possible,
essentially: no system calls

Browser
Manager

IPC

IPC

IPC

sandbox

Rendering Engine

HTML parsing, CSS parsing, JavaScript, DOM, rendering

sandbox

Rendering Engine

HTML parsing, CSS parsing, JavaScript, DOM, rendering

sandbox

chrome extension

sandbox

Adobe Flash

Each sandbox is a separate process

chrome sandbox

recap

* Data Execution Prevention
  / mark memory pages (esp the stack) as "do not execute"
  / return-into-lib, return-oriented programming

* Address Space Layout Randomization
  / and some methods to defeat it

* Stack protector
  / insert and check "canary" values on the stack

* Sandboxing

Exit slips: 1 thing you learned; 1 thing you didn't understand