IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

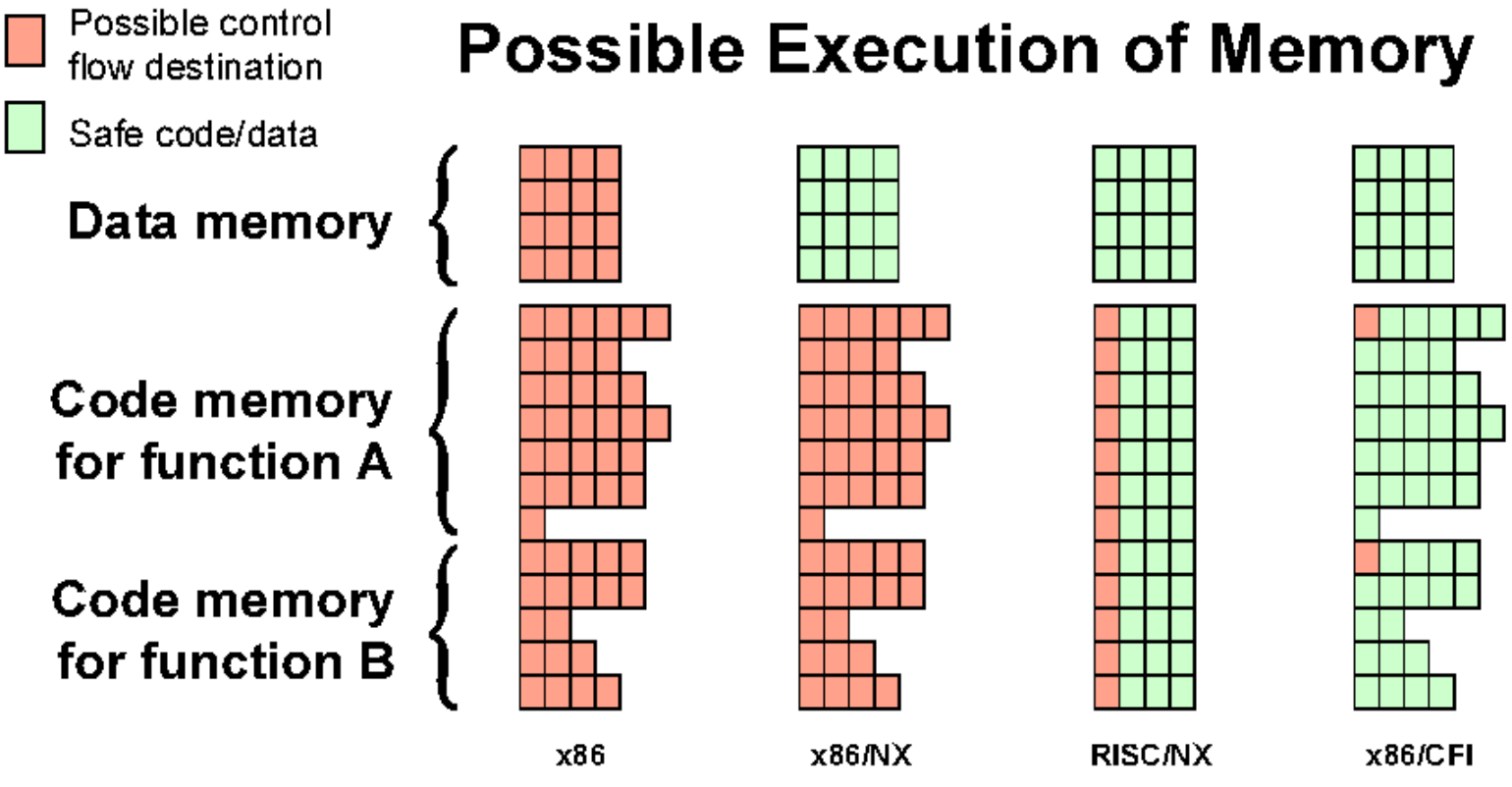
Control Flow Integrity & Secure Data Flow

IPADS, Shanghai Jiao Tong University

https://www.sjtu.edu.cn

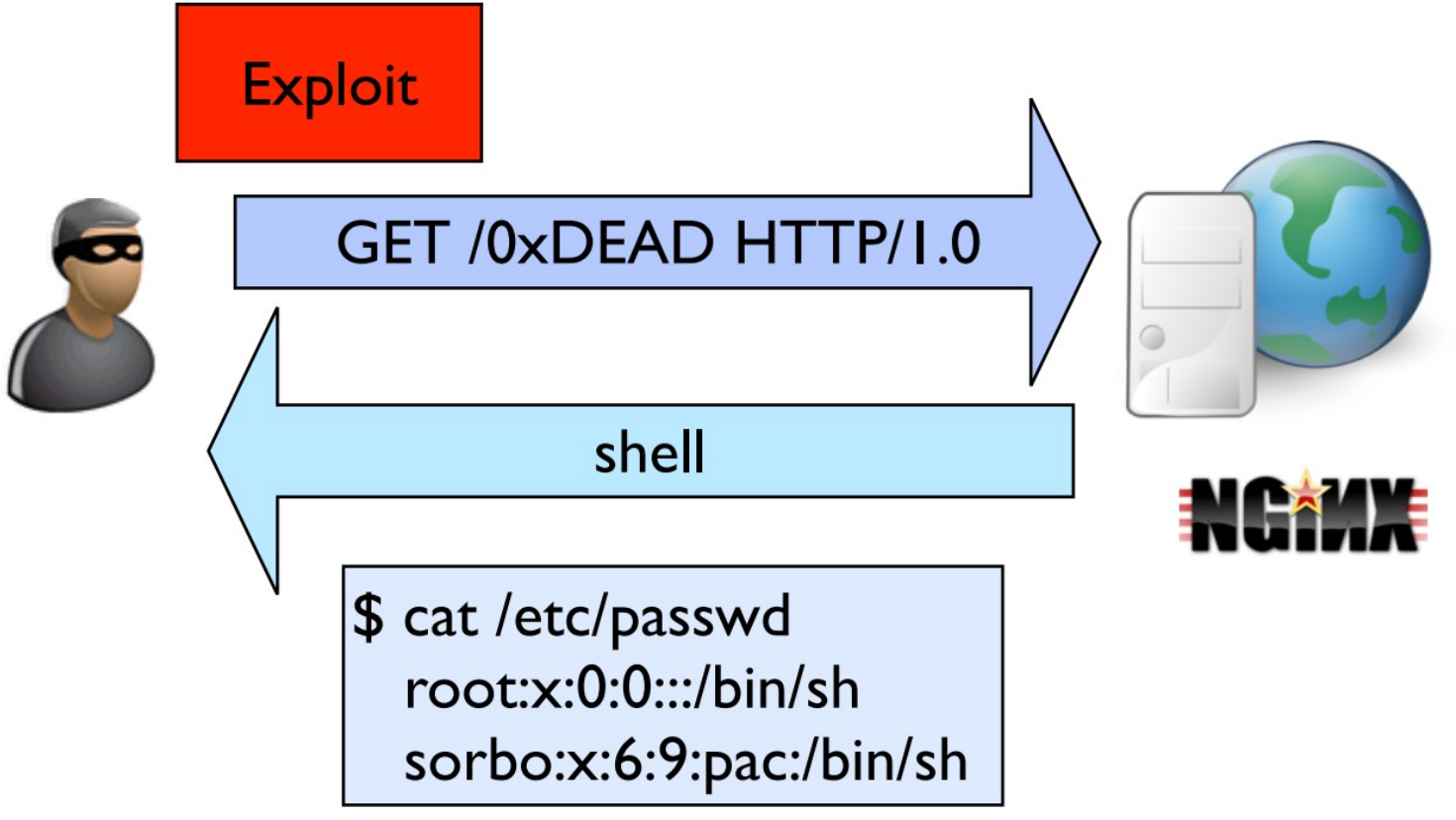
Possible Execution of Memory

[Erlingsson]

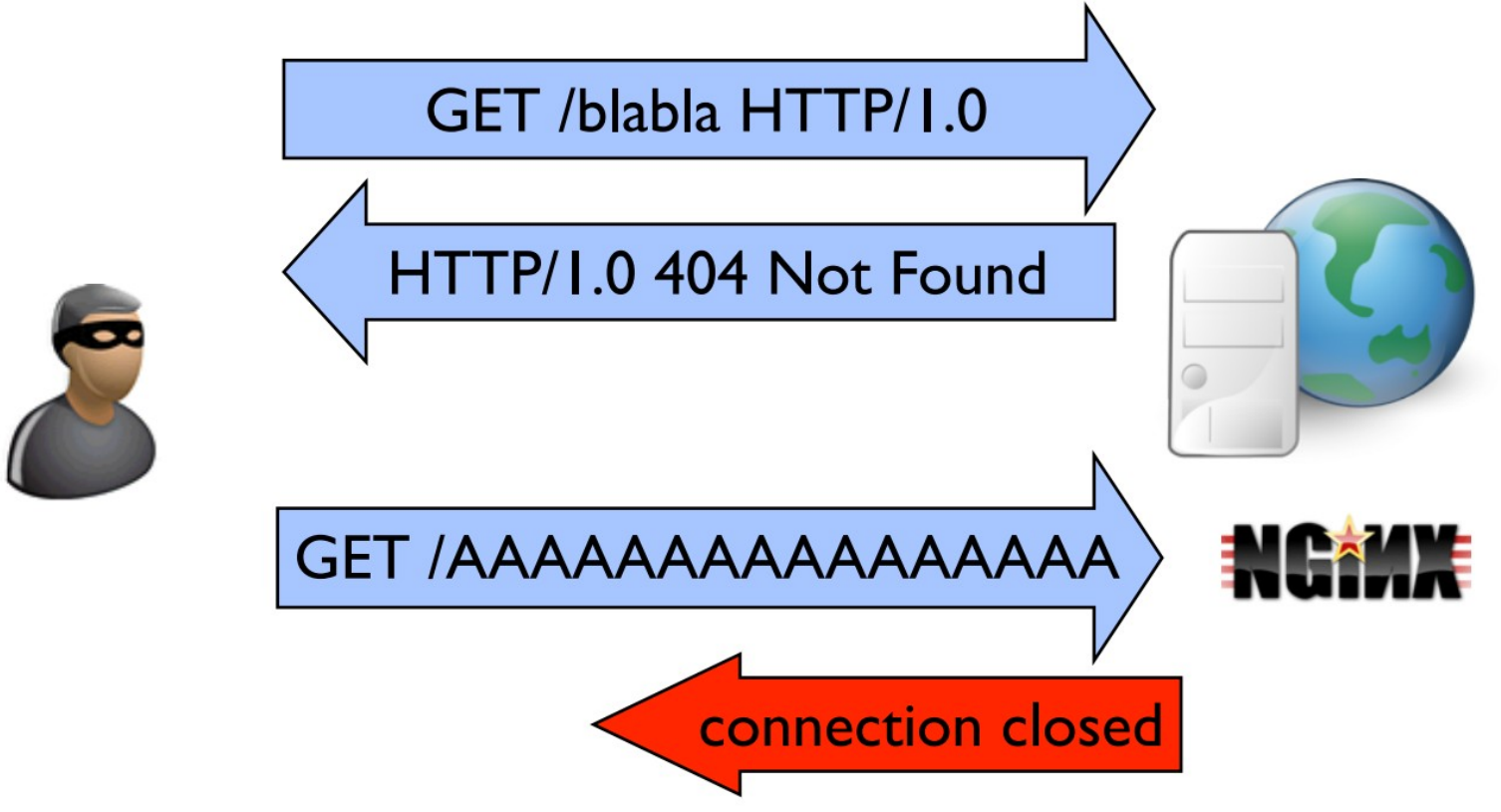


BLIND ROP

Hacking buffer overflows

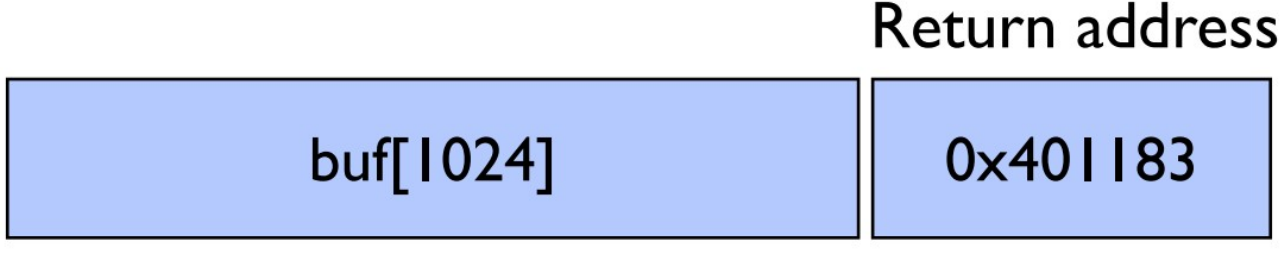


Crash or no Crash?
Enough to build exploit



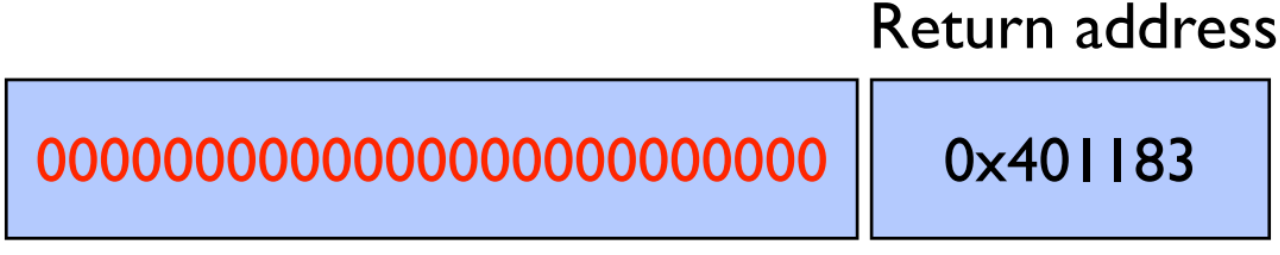
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



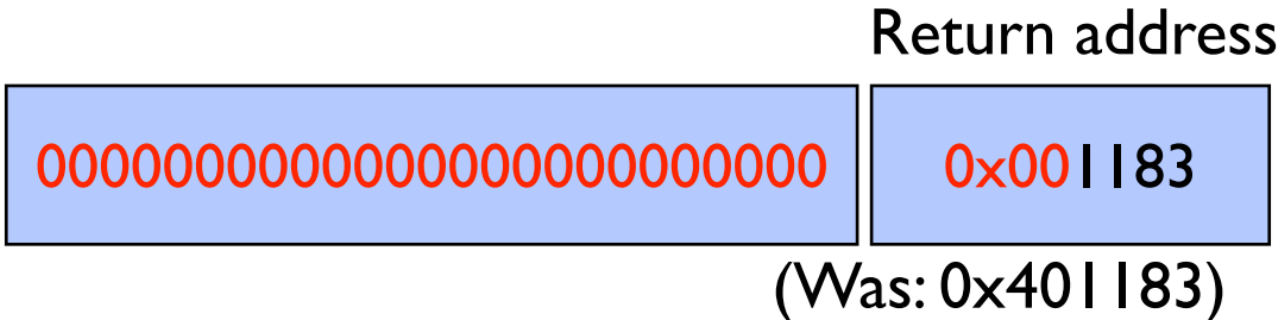
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



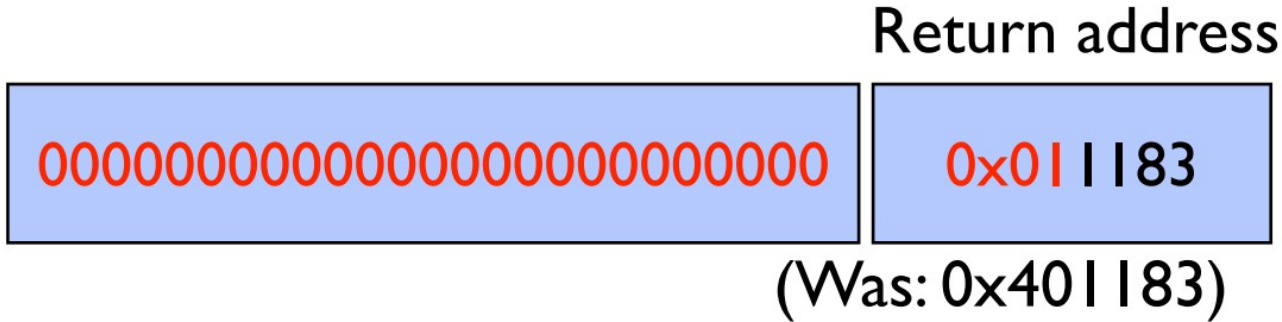
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



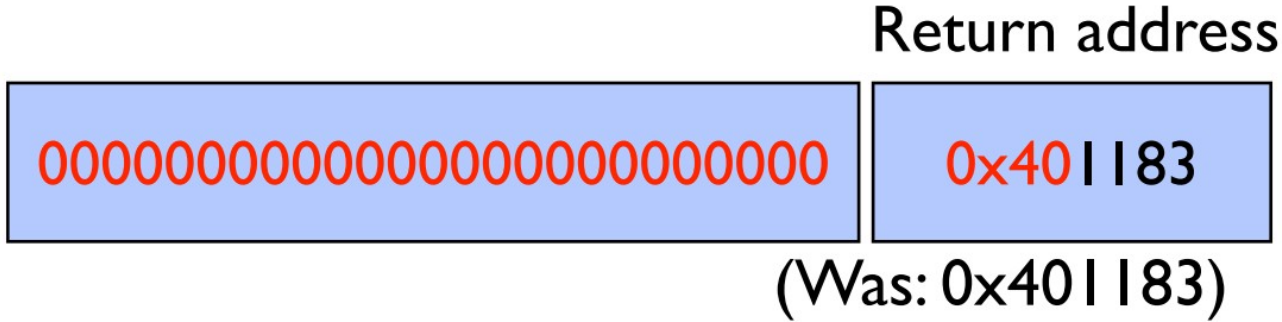
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.

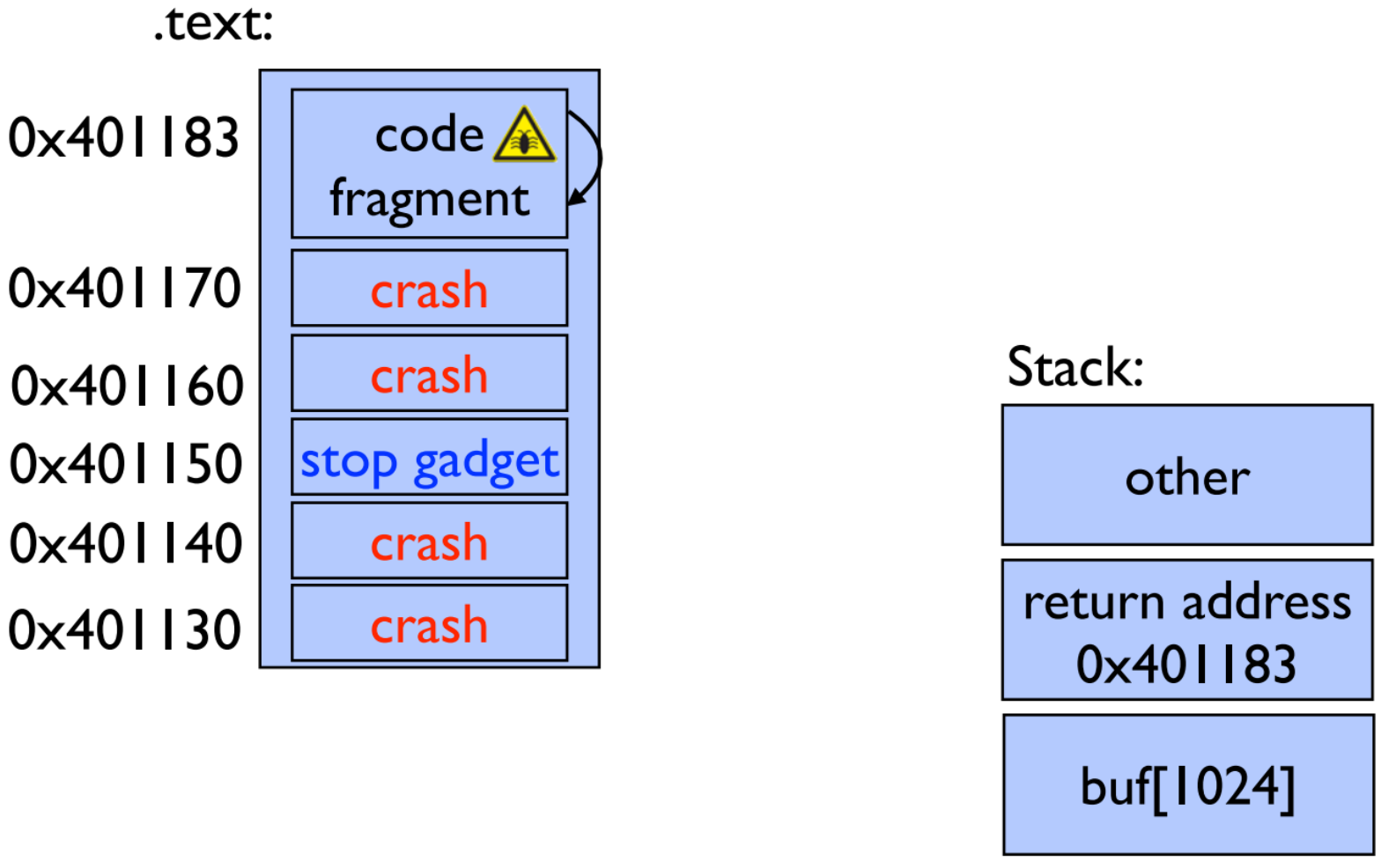


Defeating ASLR: stack reading

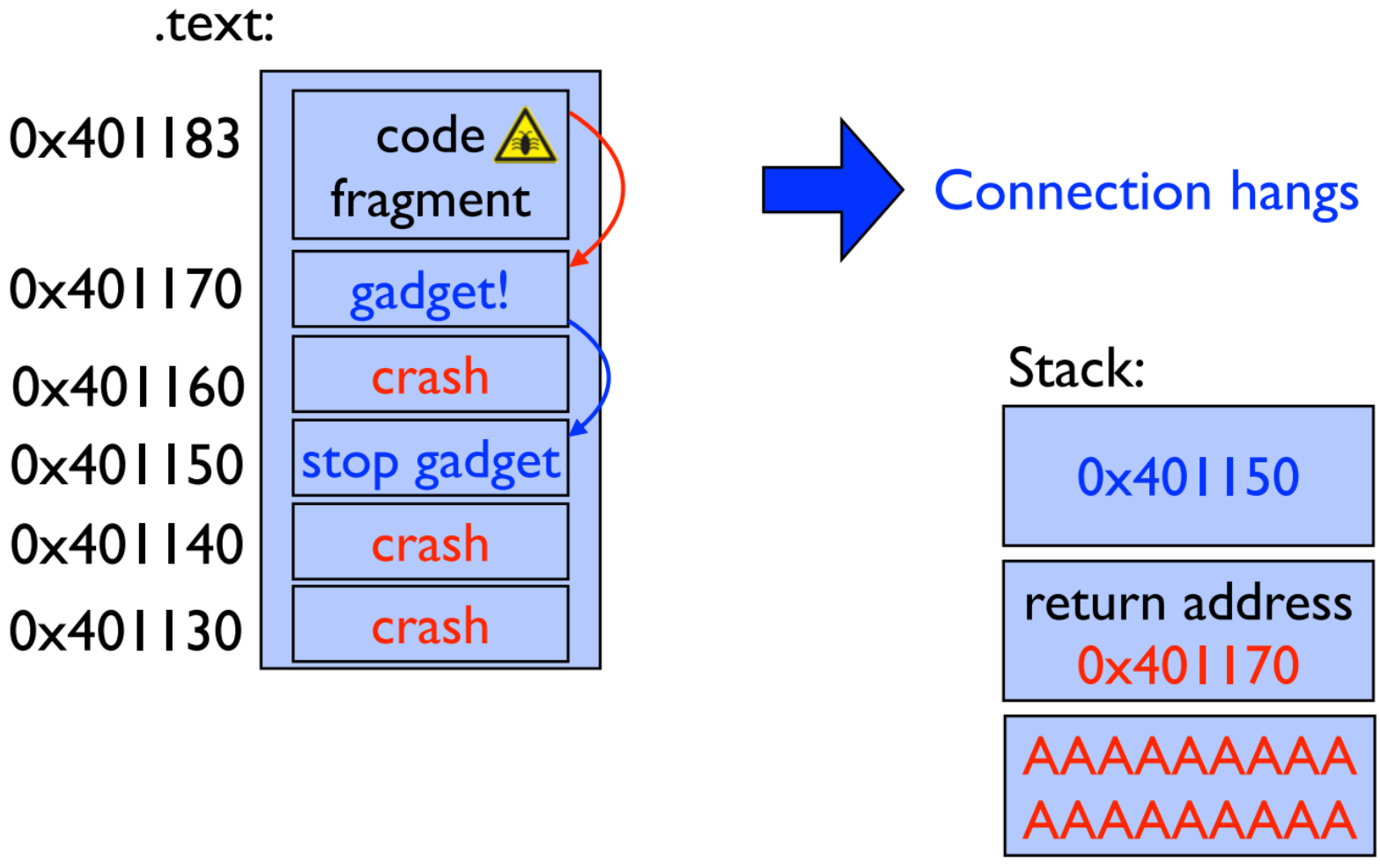
- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



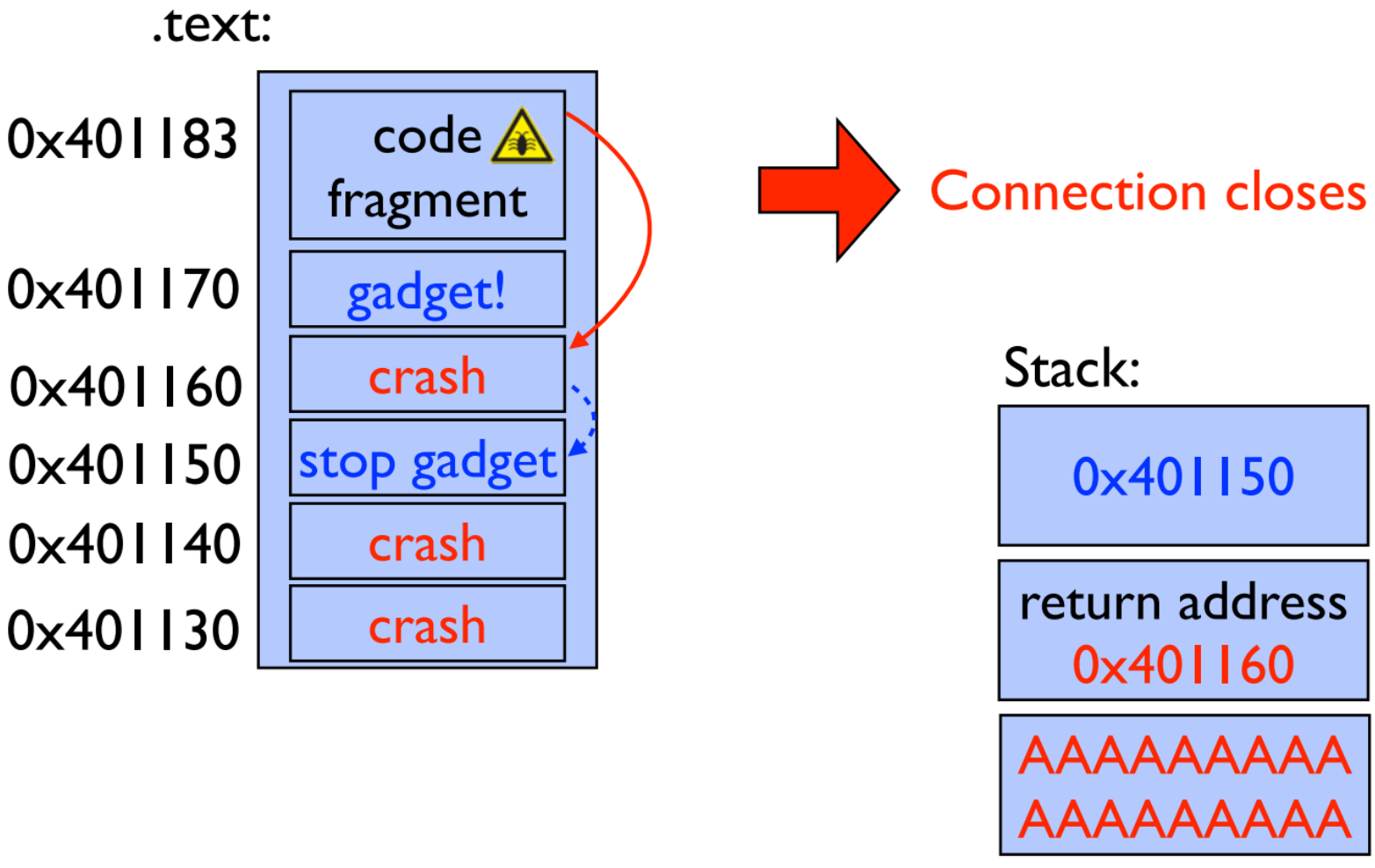
How to find gadgets?



How to find gadgets?



How to find gadgets?



Three types of gadgets

Stop gadget

```
sleep(10);
return;
```

- Never crashes

Crash gadget

```
abort();
return;
```

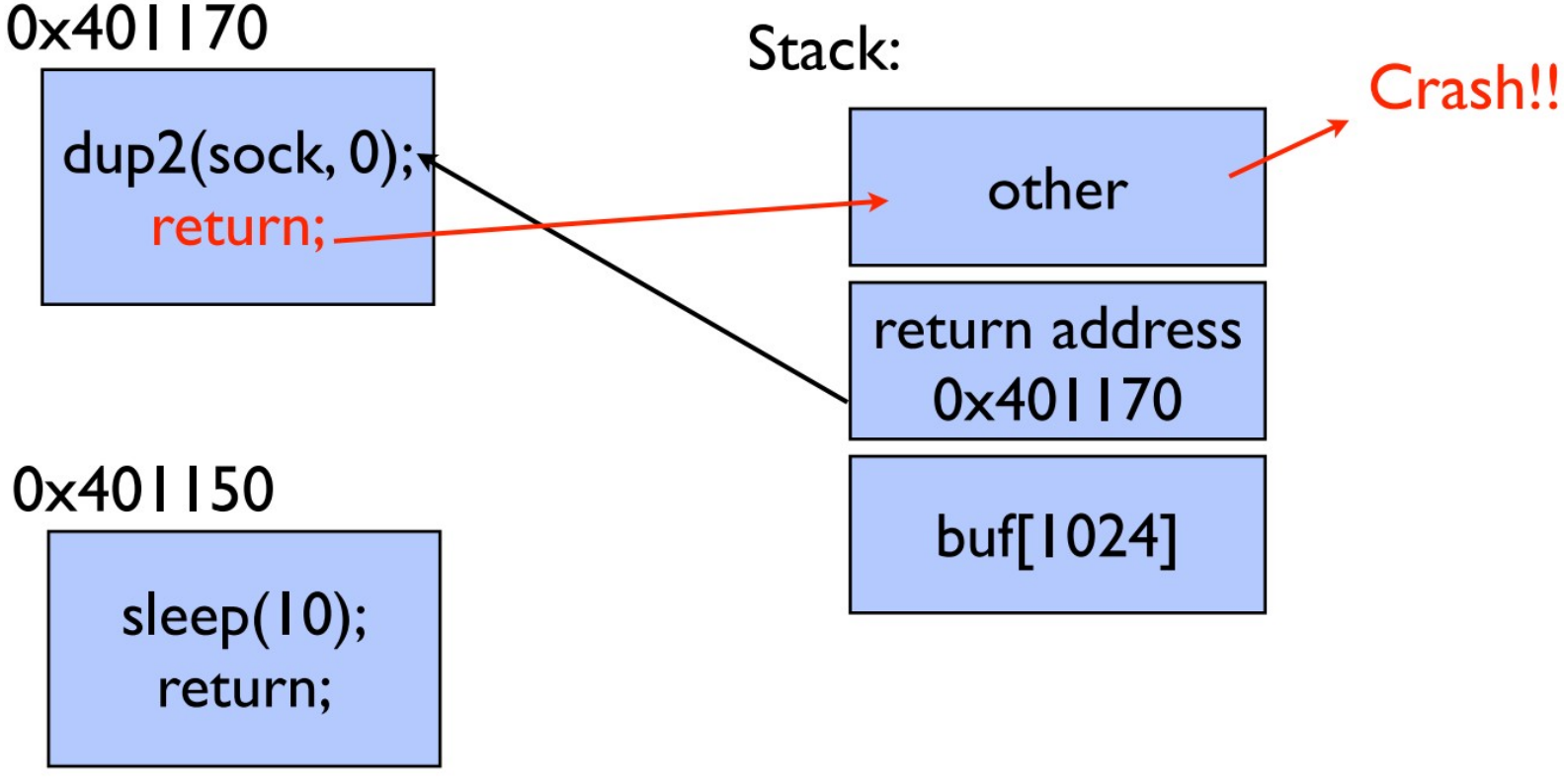
- Always crashes

Useful gadget

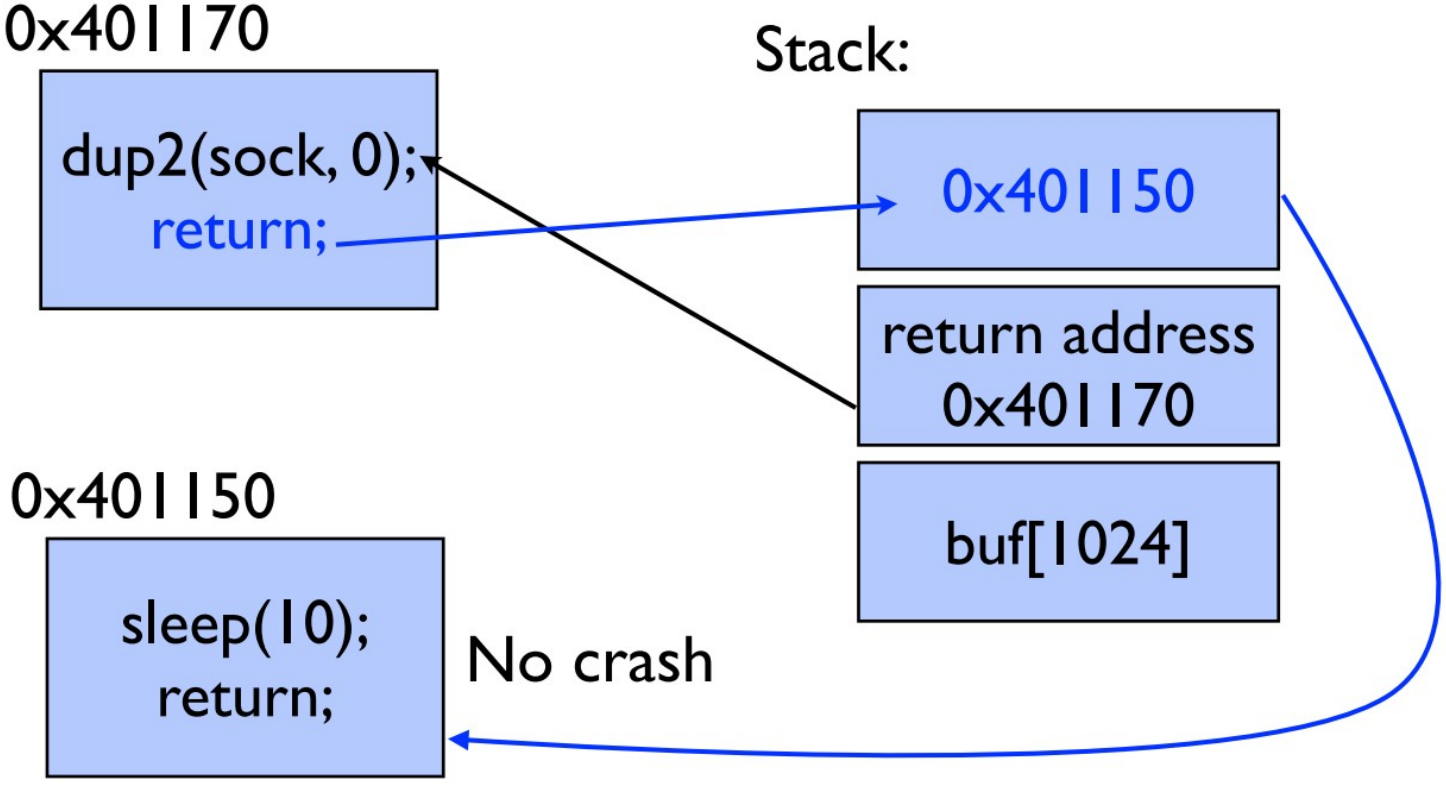
```
dup2(sock, 0);
return;
```

- Crash depends on return

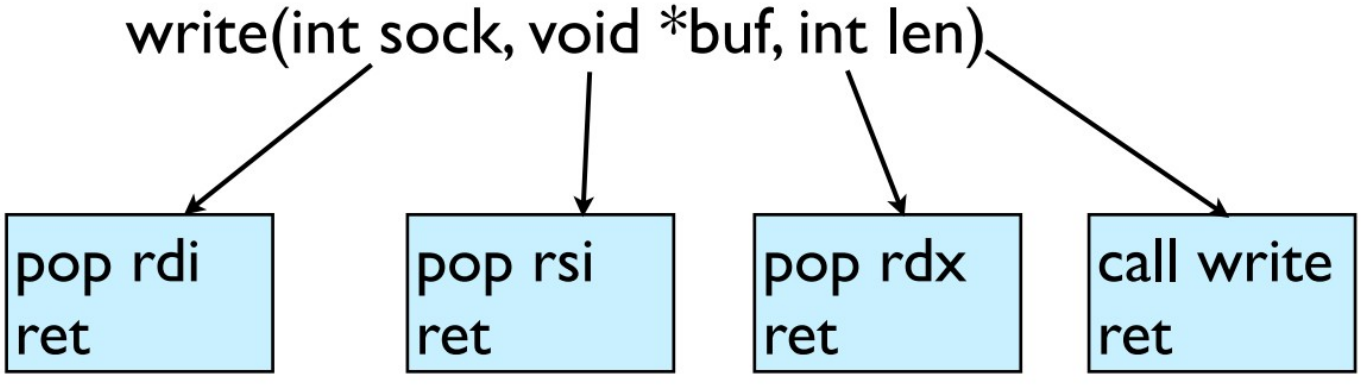
Finding useful gadgets



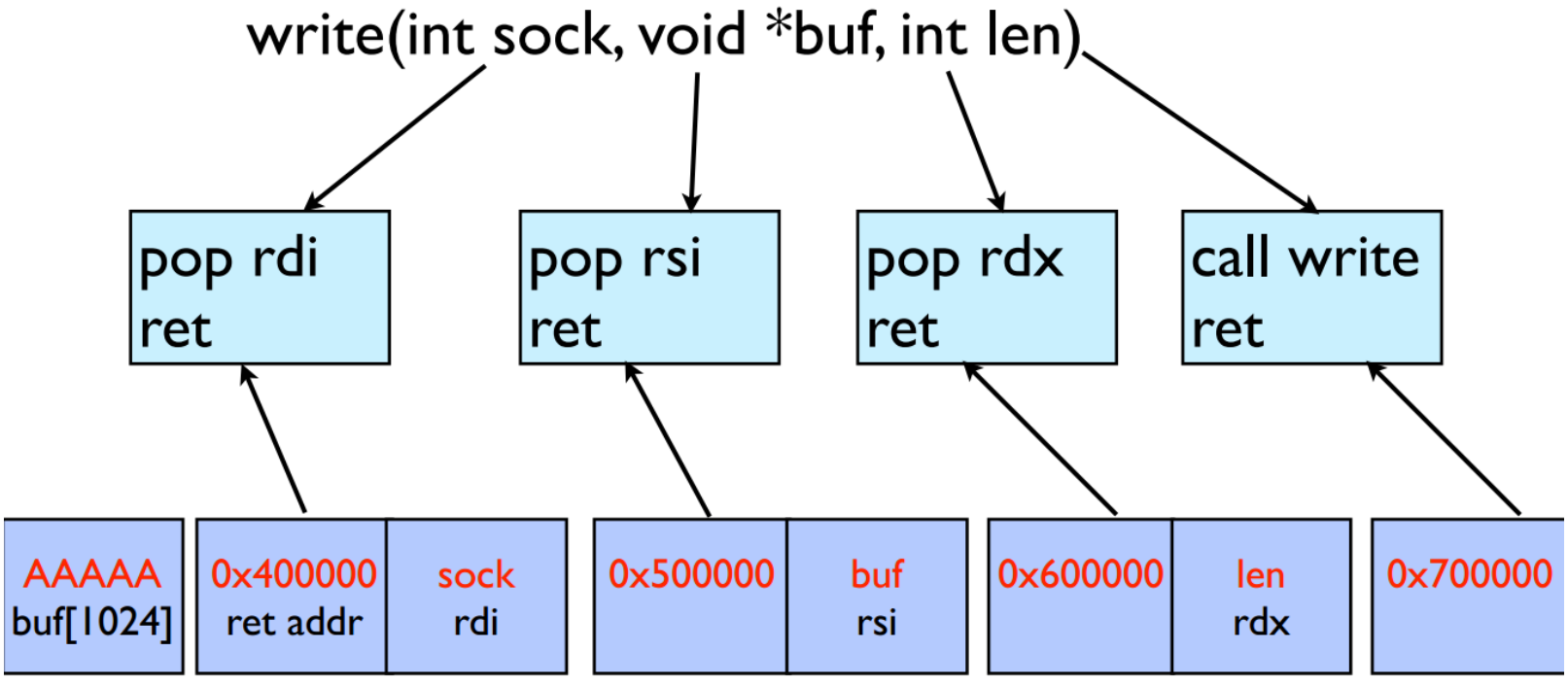
Finding useful gadgets



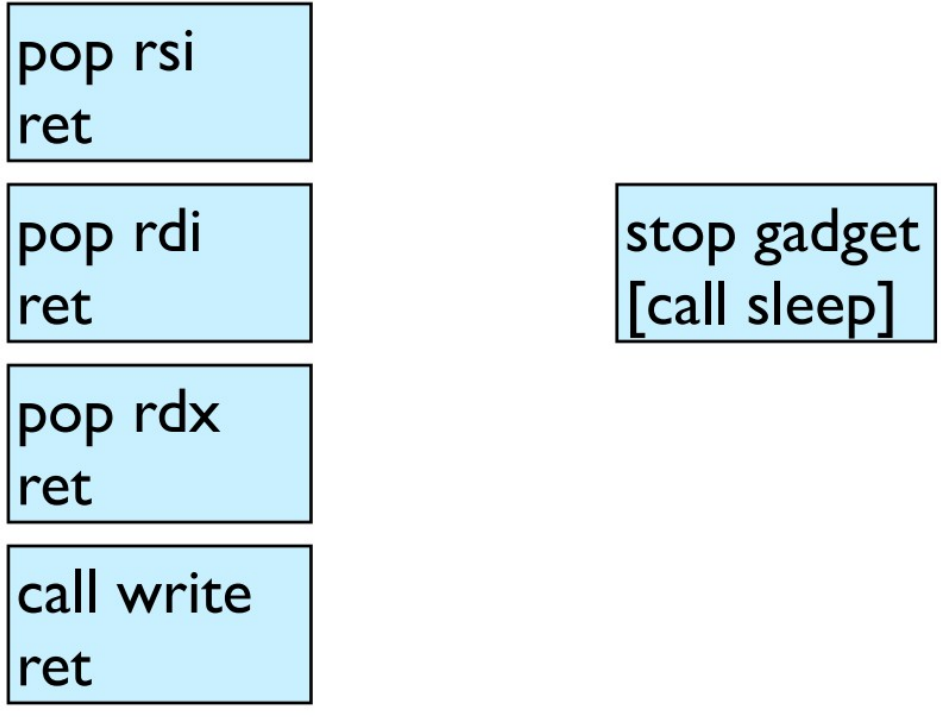
What are we looking for?



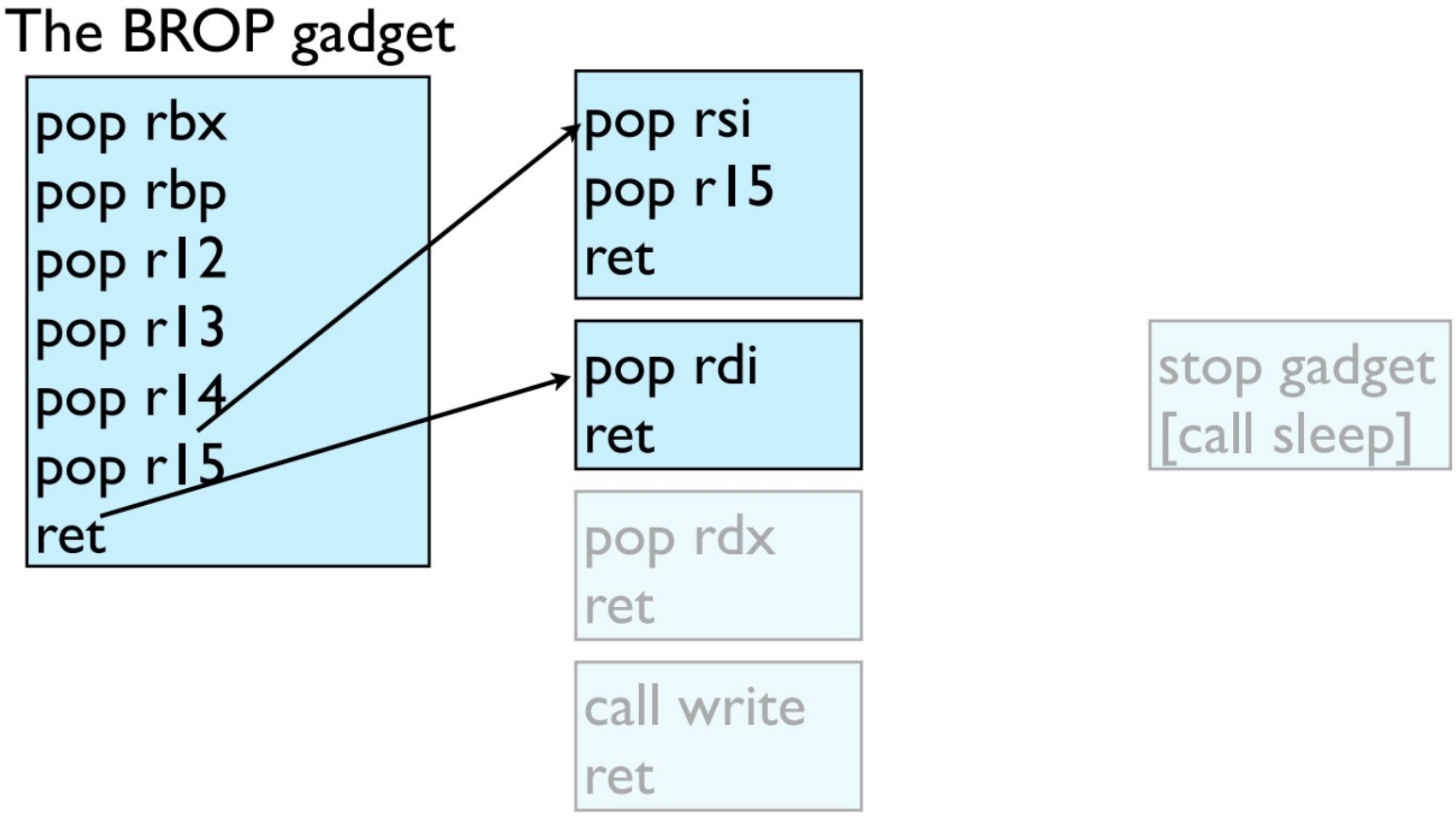
What are we looking for?



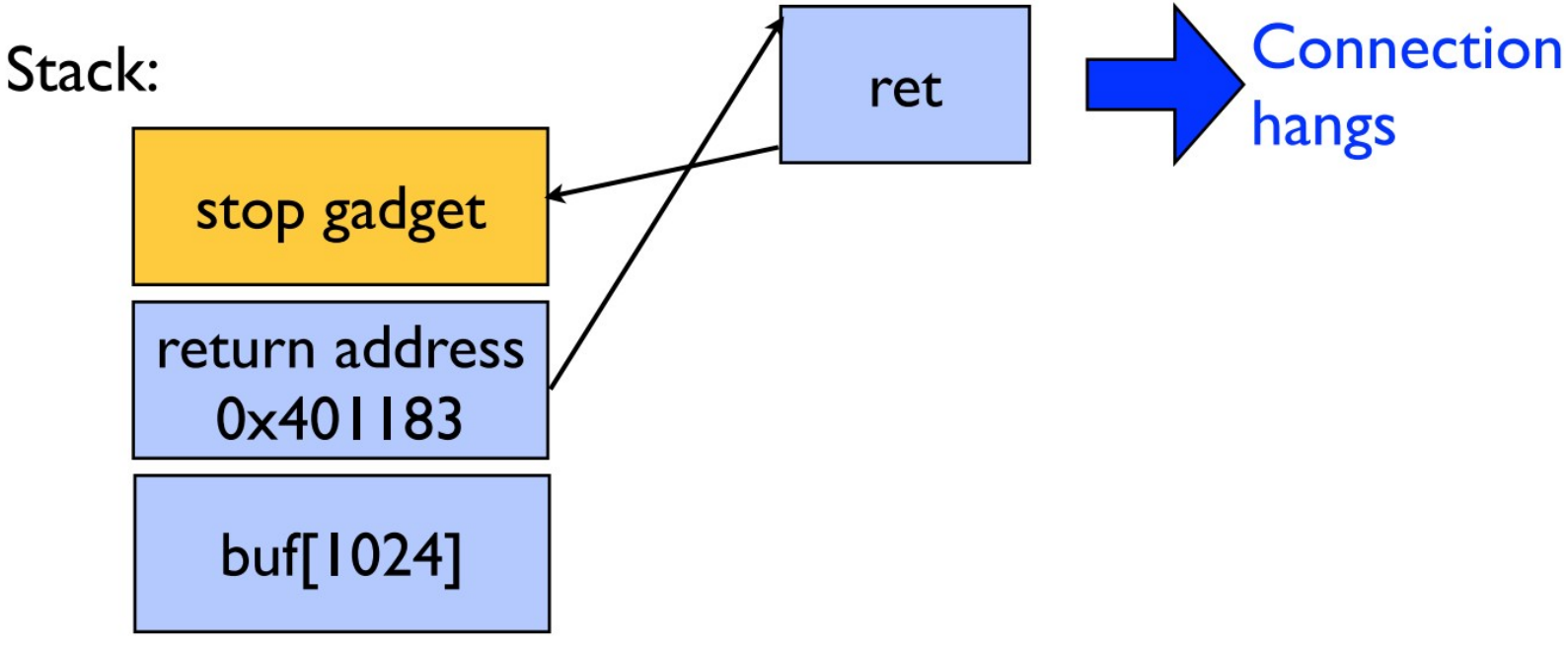
Pieces of the puzzle



Pieces of the puzzle

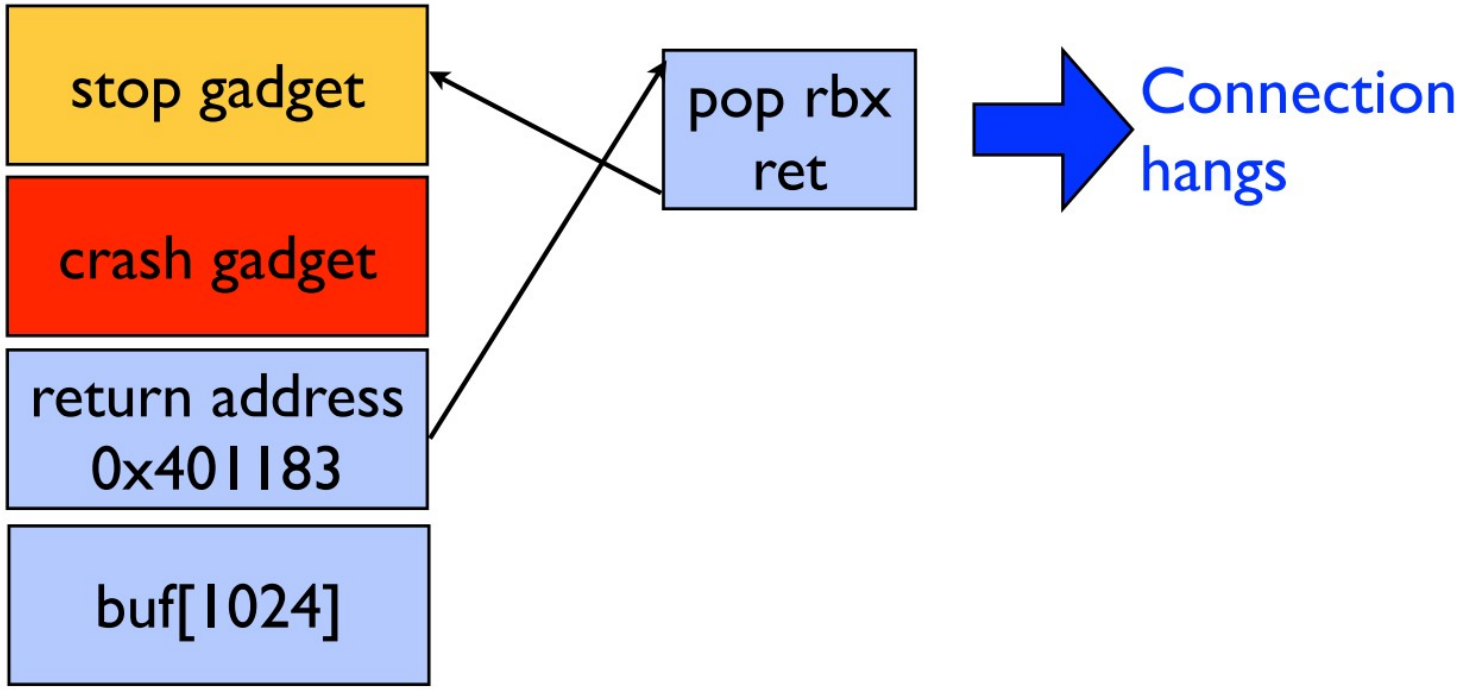


Finding the BROP gadget

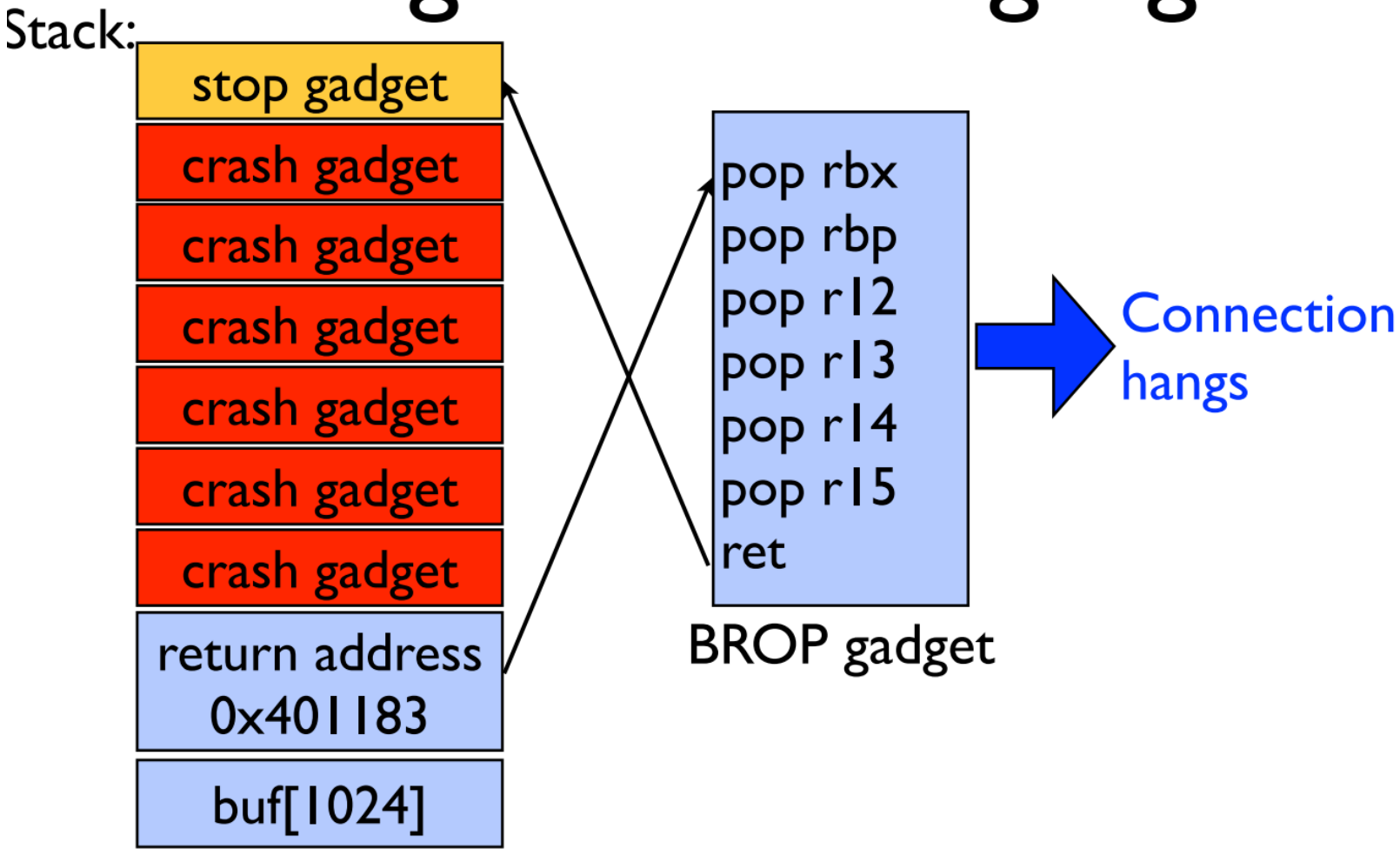


Finding the BROP gadget

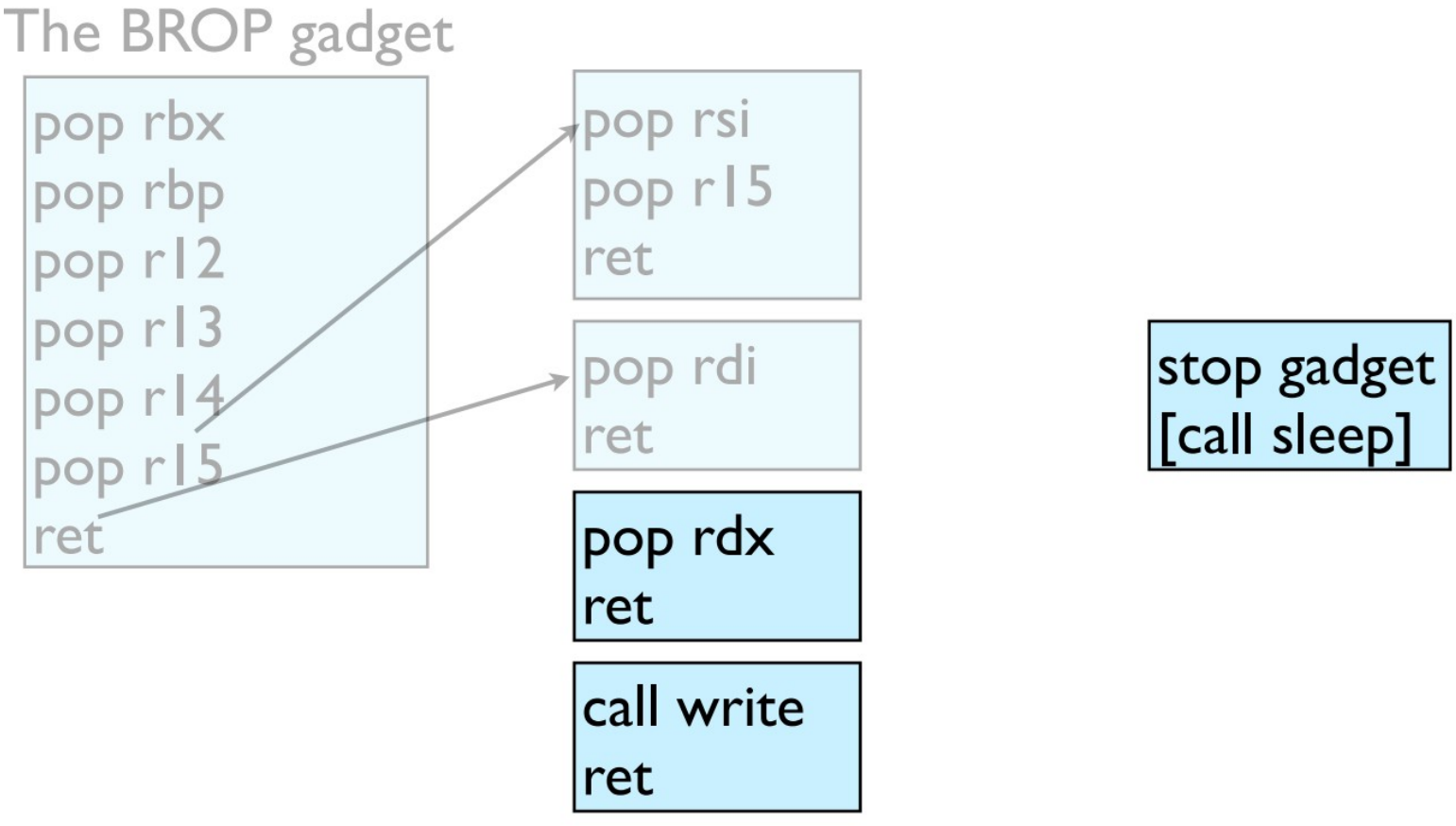
Stack:



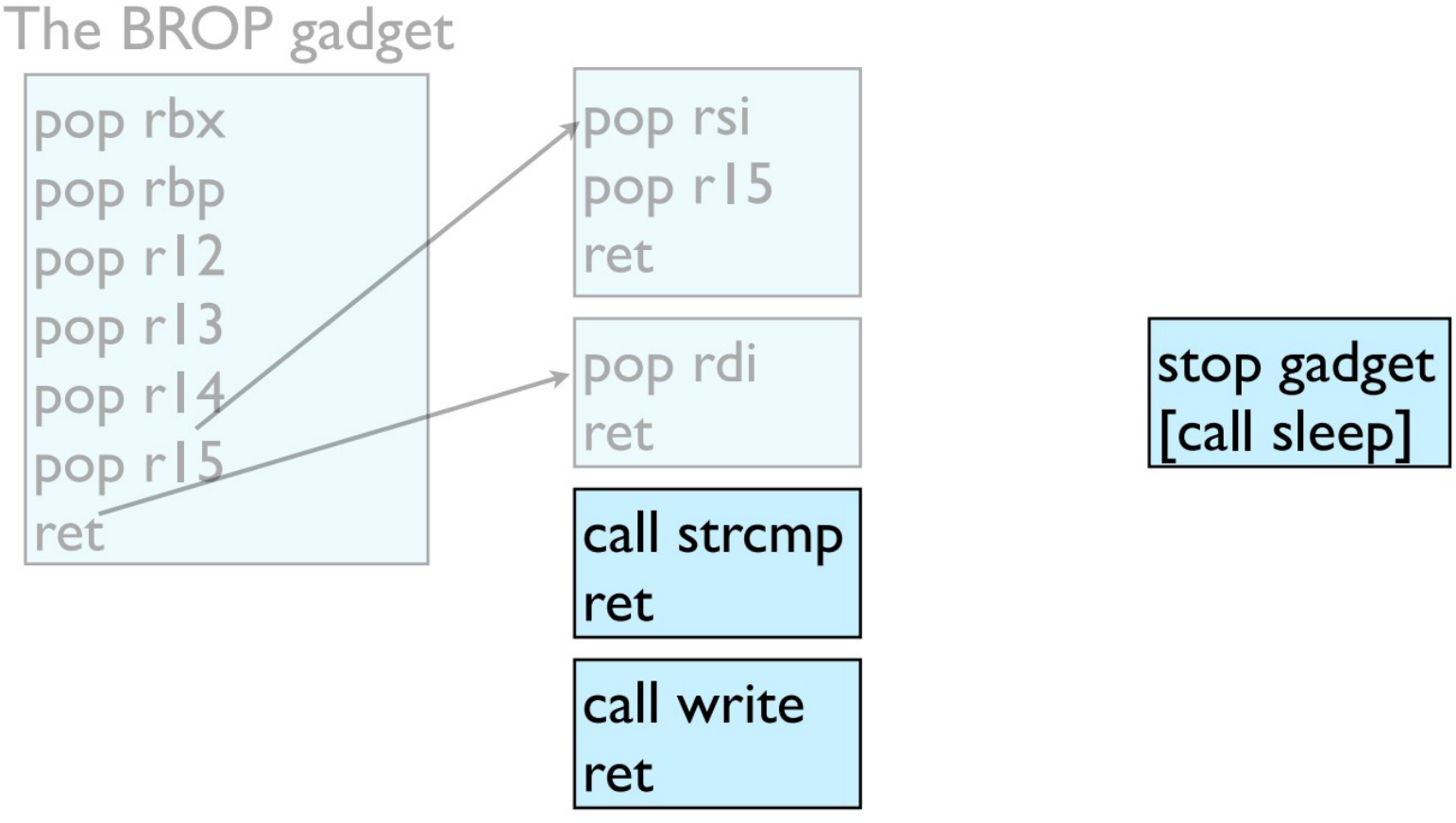
Finding the BROP gadget



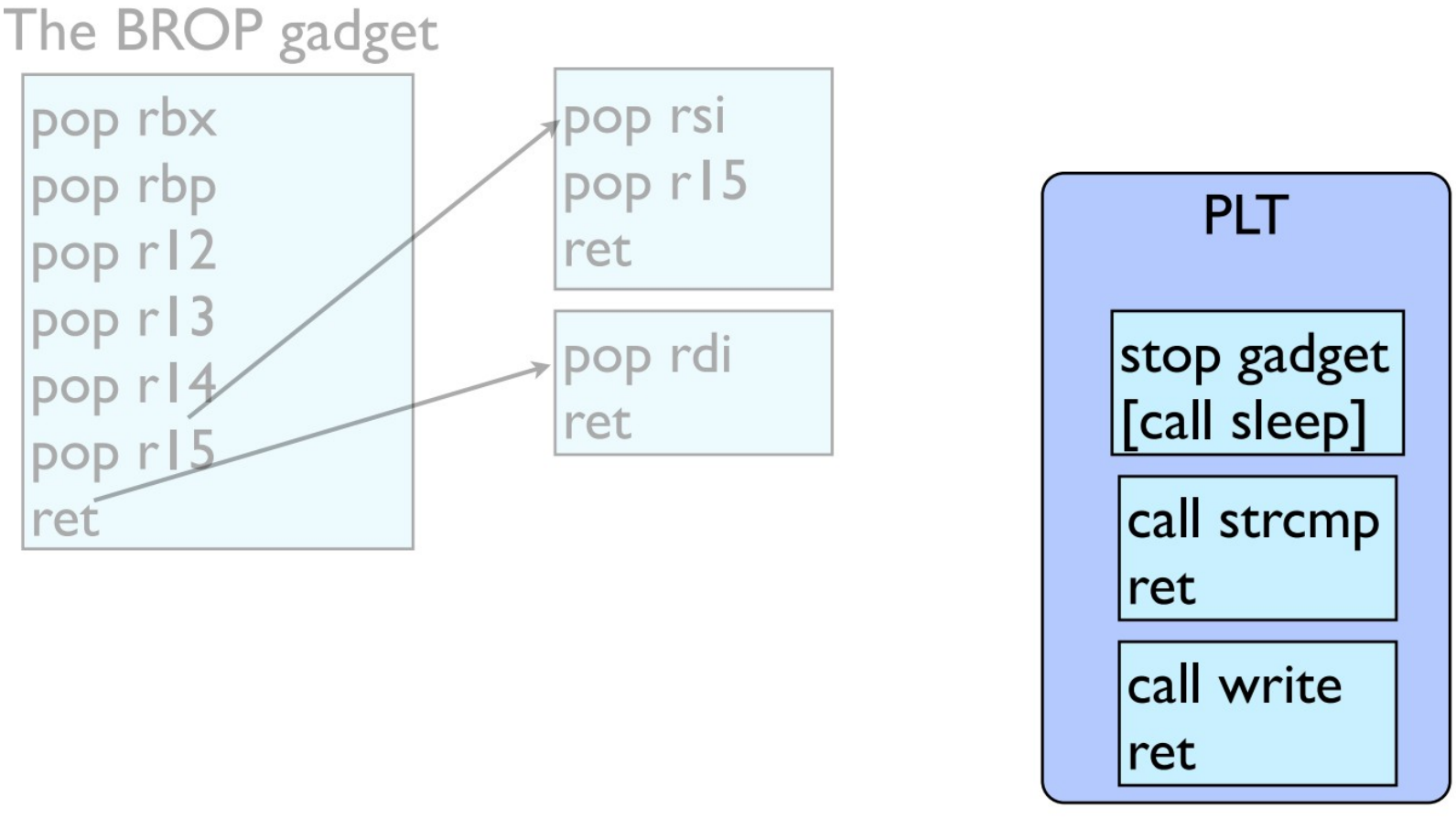
Pieces of the puzzle



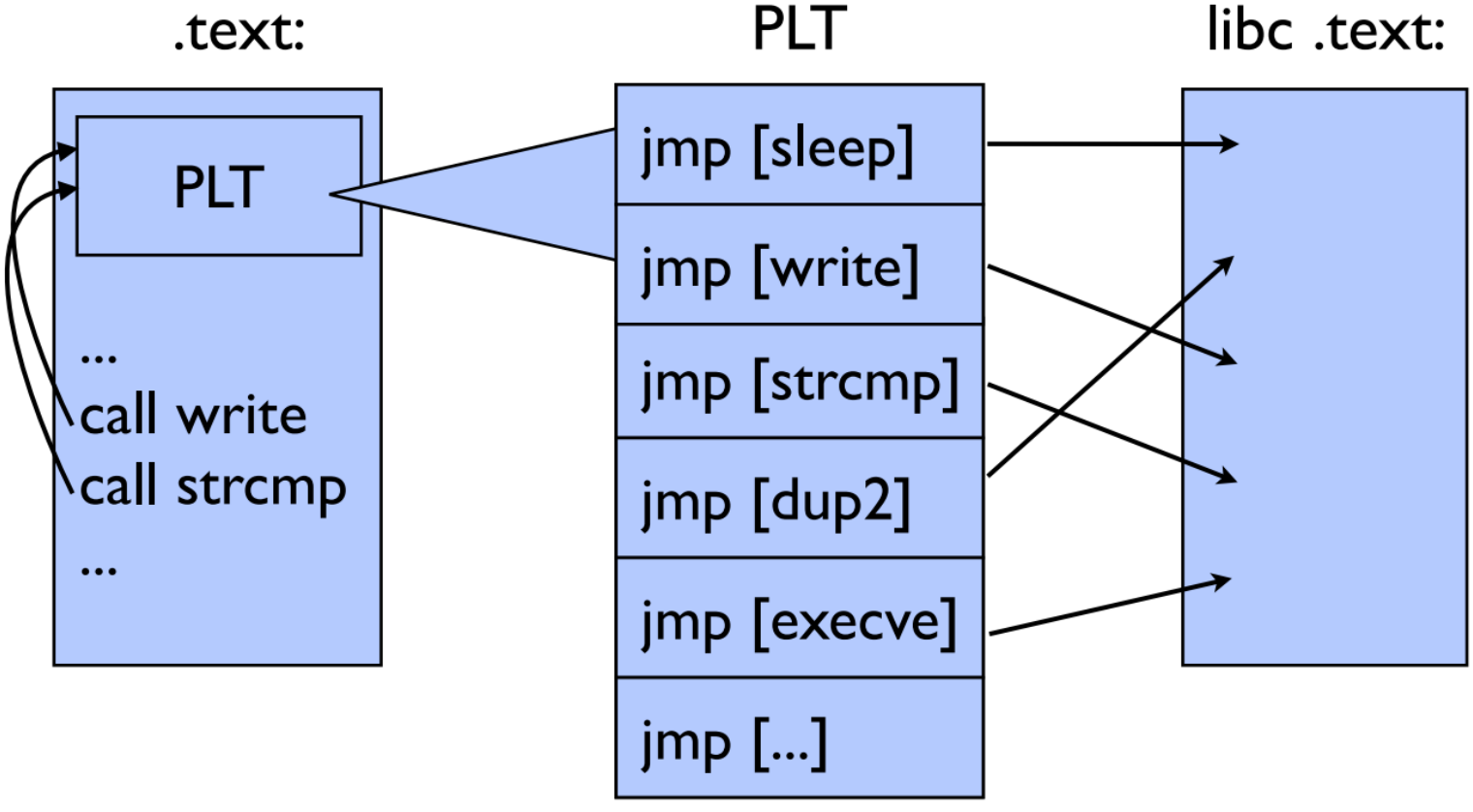
Pieces of the puzzle



Pieces of the puzzle



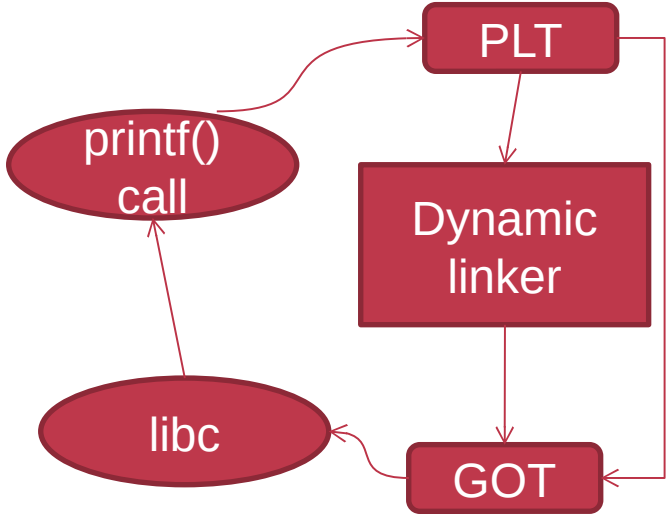
Procedure Linking Table (PLT)

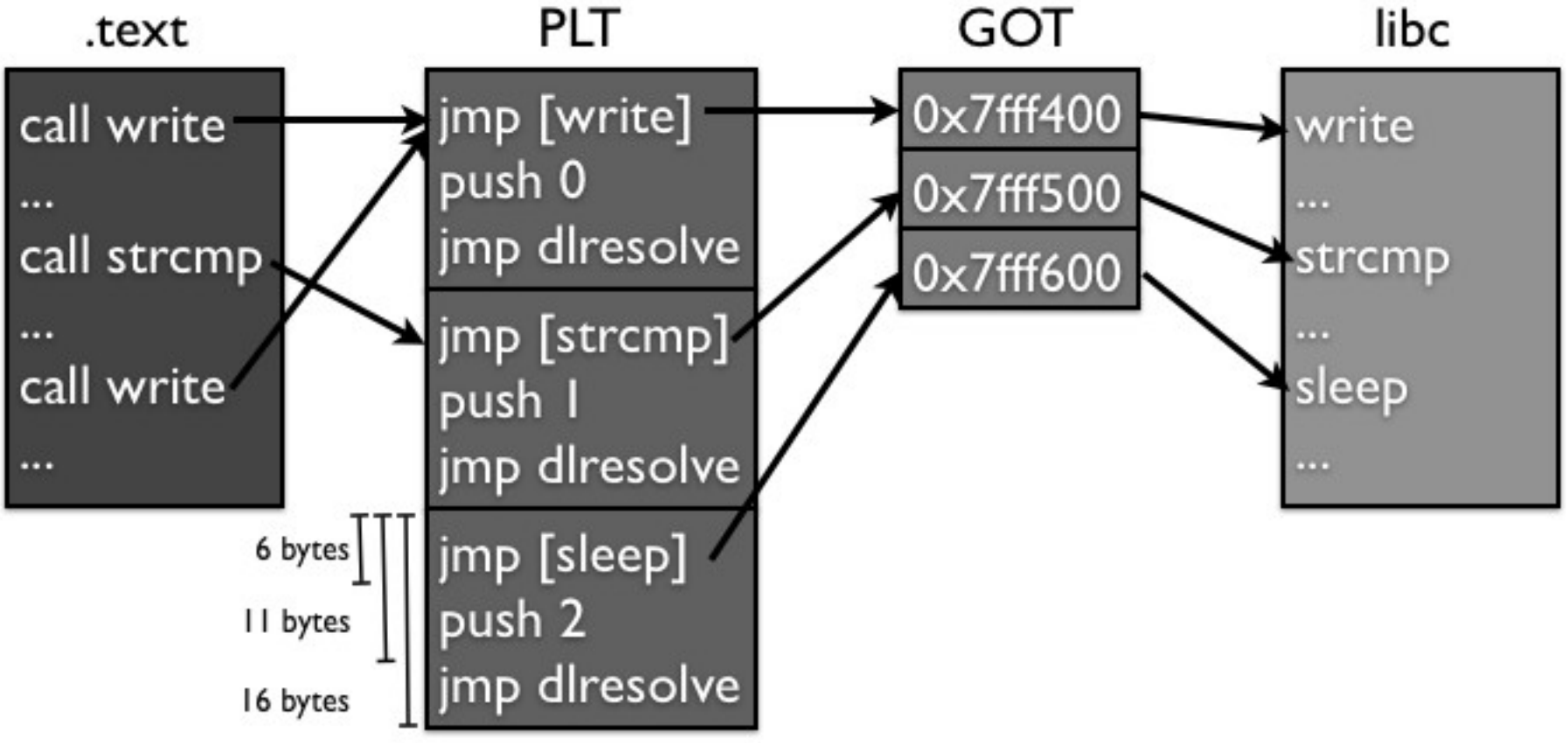


GOT & PLT

Jump to the address that this entry of the GOT contains:

- First time: call to dynamic linker and write the function address to GOT
- Not first time: the GOT already contains the address that points to printf.




.text
PLT
GOT
libc
call write
...
call strcmp
...
call write
...
jmp [write]
push 0
jmp dlresolve
jmp [strcmp]
push 1
jmp dlresolve
jmp [sleep]
push 2
jmp dlresolve
0x7fff400
0x7fff500
0x7fff600
write
...
strcmp
...
sleep
...
6 bytes
11 bytes
16 bytes

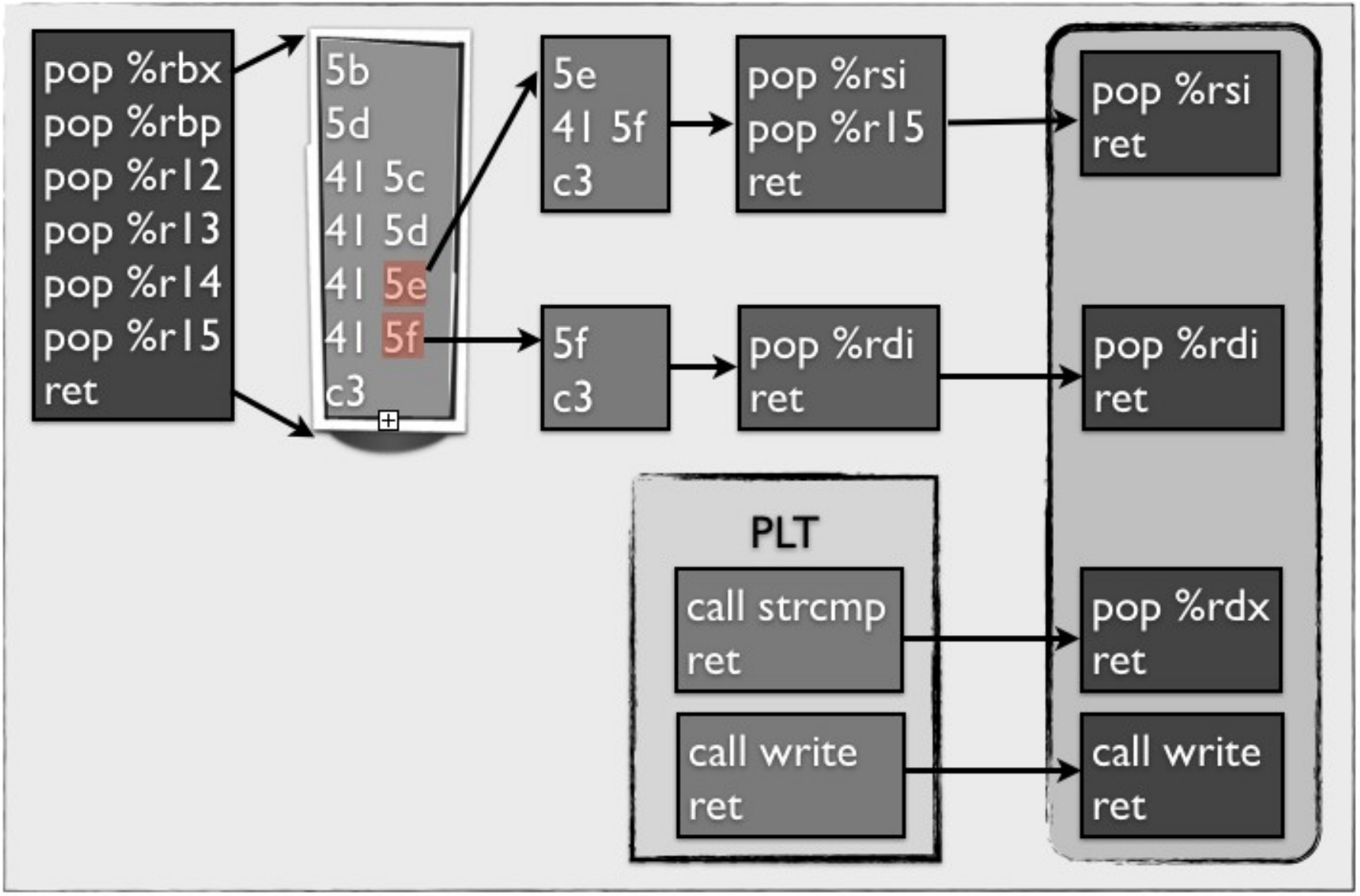
Fingerprinting strcmp

arg1	arg2	result
readable	0x0	crash
0x0	readable	crash
readable	readable	nocrash

Can now control three arguments:
strcmp sets RDX to length of string

Finding write

- Try sending data to socket by calling candidate PLT function.
- check if data received on socket.
- chain writes with different FD numbers to find socket. Use multiple connections.


pop %rbx
pop %rbp
pop %r12
pop %r13
pop %r14
pop %r15
ret
5b
5d
41 5c
41 5d
41 5e
41 5f
c3
5e
41 5f
c3
pop %rsi
pop %r15
ret
5f
c3
pop %rdi
ret
pop %rsi
ret
pop %rdi
ret
PLT
call strcmp
ret
call write
ret
pop %rdx
ret
call write
ret

Launching a shell

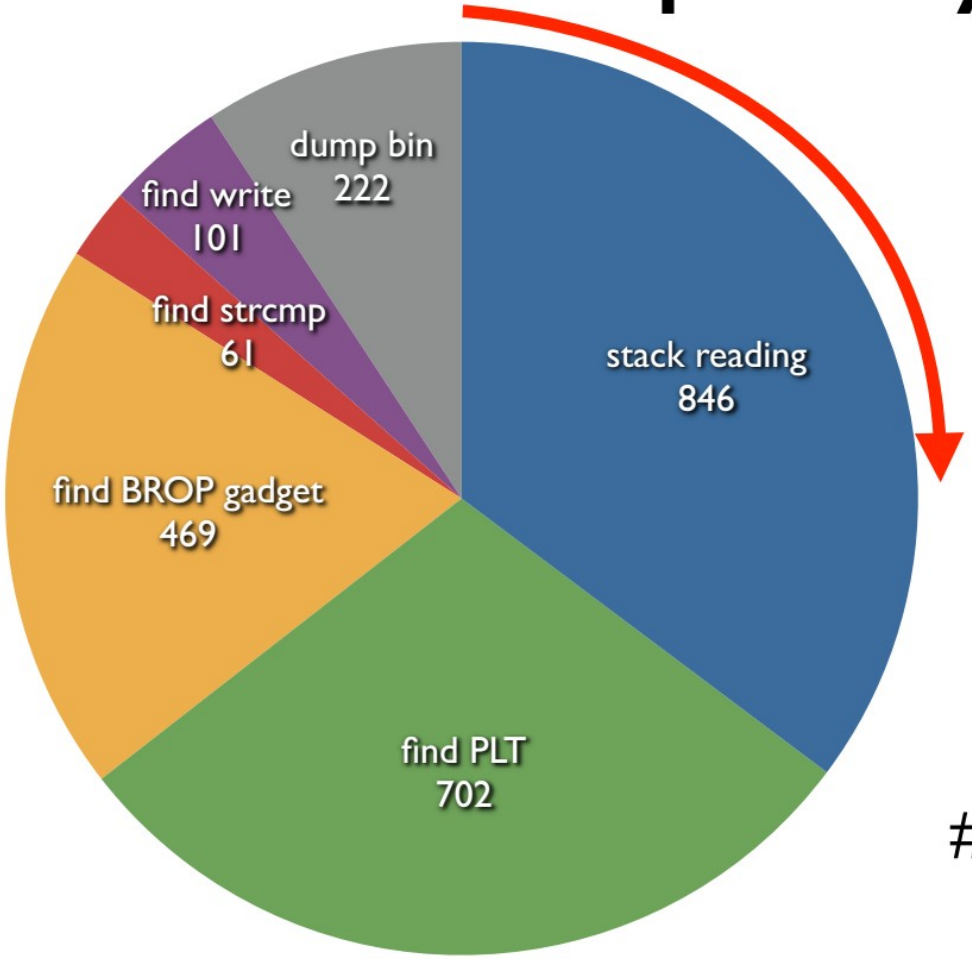
1. dump binary from memory to network. Not blind anymore!
2. dump symbol table to find PLT calls.
3. redirect stdin/out to socket:
   - dup2(sock, 0); dup2(sock, 1);
4. read() “/bin/sh” from socket to memory
5. execve(“/bin/sh”, 0, 0)

Braille

- Fully automated: from first crash to shell.
- 2,000 lines of Ruby.
- Needs function that will trigger overflow:
  - nginx: 68 lines.
  - MySQL: 121 lines.
  - toy proprietary service: 35 lines.

```
try_exp(data) → true  crash
                false no crash
```

Attack complexity



# of requests
for nginx

Try It!

http://www.scs.stanford.edu/brop/

More information

- S&P'14 paper: http://www.scs.stanford.edu/brop/bittau-brop.pdf
- Blog: http://drops.wooyun.org/tips/3071

DATA FLOW PROTECTION

Two Usages of Data Flow Tracking

Usage-1: protect critical data

- E.g., track data to ensure none is leaked

Usage-2: defend against suspicious input data

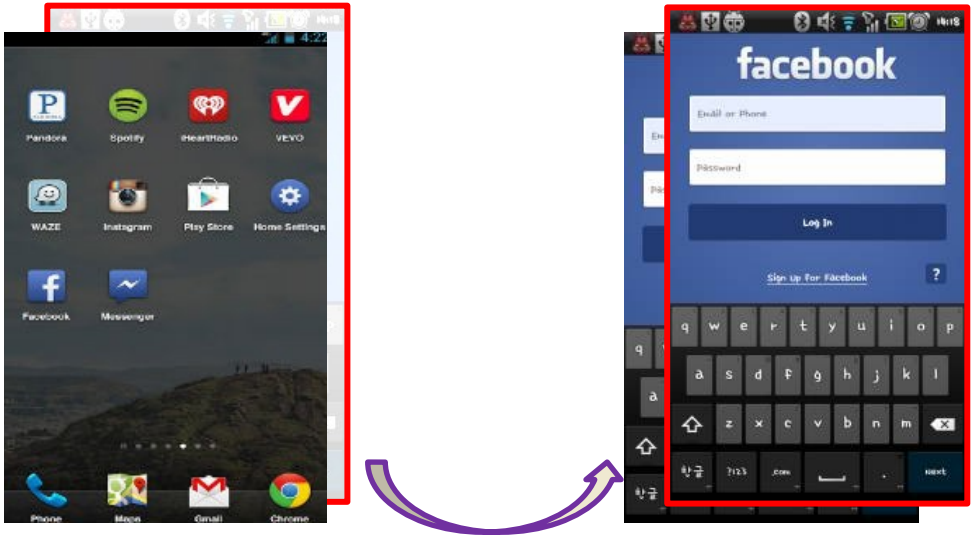
- E.g., one input data cannot flow to return address!

Ways an Attacker Can Steal Your Secrets

KeyLogger: install a KeyLogger/TouchLogger on the phone

Phishing: install a malware and lure the user to enter password

Malicious app in background, waiting for the Facebook app to launch



Once Facebook app starts, the malicious app will get active and make itself cover the real UI

Bianchi, Antonio, et al. "What the App is That? Deception and Countermeasures in the Android User Interface." *S&P* 2015

Ways an Attacker Can Steal Your Secrets

KeyLogger: install a KeyLogger/TouchLogger on the phone

Phishing: install a malware and lure the user to enter password

MemScan: use rootkit and scan the memory for any plaintext

```
01010010101010101000101
01010101010101010101010
10101001010101010101010
01010101010101010101001
01001101010101010101010
10101000100100000010 1
010010010010
```

The Samsung memory mapping bug: http://forum.xda-developers.com/showthread.php?t=2048511

The "Data Residue" Problem

App	Extracted Cleartext Data
Email	password, email contents, subjects, from/to, contacts
OI Notepad (doc)	document and metadata
KeePass (password mgr)	app password, all stored passwords & descriptions
Pageonce (finance)	password, transactions, bank account information
Facebook (social)	wall posts and messages

(a) Examples of data extracted from RAM / DB.

App	Data	When App Uses Data
Email	password	user/automatic refresh
	contents	user opens the email
OI Notepad	note title	on the note list screen
	note body	user edits the note
KeePass	master password	app launches
	entry name	on the entry list screen
	entry password	user opens the entry

(c) Example usage of hoarded data by apps.

App	Description	Extracted Cleartext Data			Cleartext Data Hoarding					
					RAM			SQLite DB		
		Pass-word	Cont-ents	Meta-data	Pass-word	Cont-ents	Meta-data	Pass-word	Cont-ents	Meta-data
Email	email (default)	Y	Y	Y	Y		Y	Y	Y	Y
GMail	email		Y	Y					Y	Y
Y! Mail	email	Y	Y	Y					Y	Y
GDocs	documents			Y			Y		Y	Y
OI Notepad	documents									
DropBox	documents			Y			Y			
KeePass	password mgr	Y	Y	Y	Y	Y	Y			
	password mgr	Y	Y	Y	Y		Y			
Amazon	e-commerce									
Mint	finance		Y	Y		Y	Y		Y	Y
Google+	social		Y	Y					Y	Y
Facebook	social		Y	Y					Y	Y
LinkedIn	social		Y	Y			Y		Y	Y

(b) Exposure of cleartext sensitive data across all 14 apps.

Android's default Email app:
has plaintext of password in RAM 10 minutes after login

Tang, Yang, et al. "CleanOS: Limiting Mobile Data Exposure with Idle Eviction." *OSDI*. 2012.

Ways an Attacker Can Steal Your Secrets

KeyLogger: install a KeyLogger/TouchLogger on the phone

Phishing: install a malware and lure the user to enter password

MemScan: use rootkit and scan the memory for any plaintext

ScreenCapture: get information directly from the screen

Jang, Yeongjin, et al. "A11y Attacks: Exploiting Accessibility in Operating Systems." *CCS*, 2014.

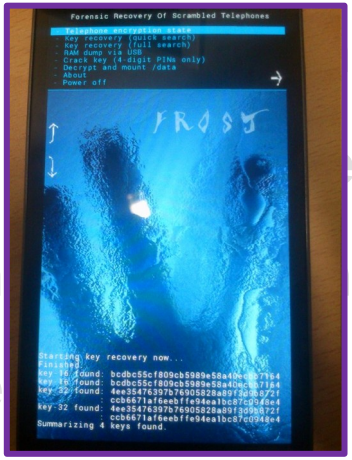
Ways an Attacker Can Steal Your Secrets

KeyLogger: the phone

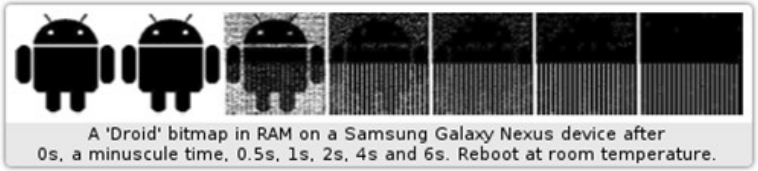
Phishing: enter password

MemScan: y plaintext

ScreenCaptu screen



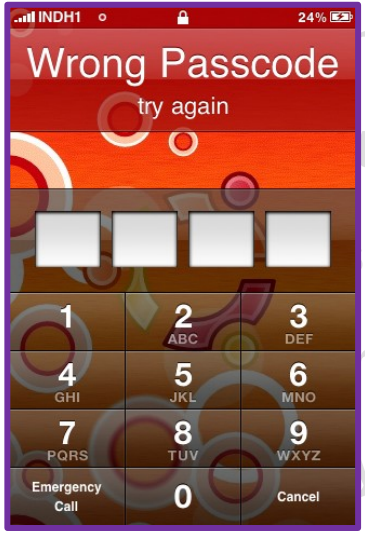
Cold-boot: physically attack that can scan memory to get any plaintext

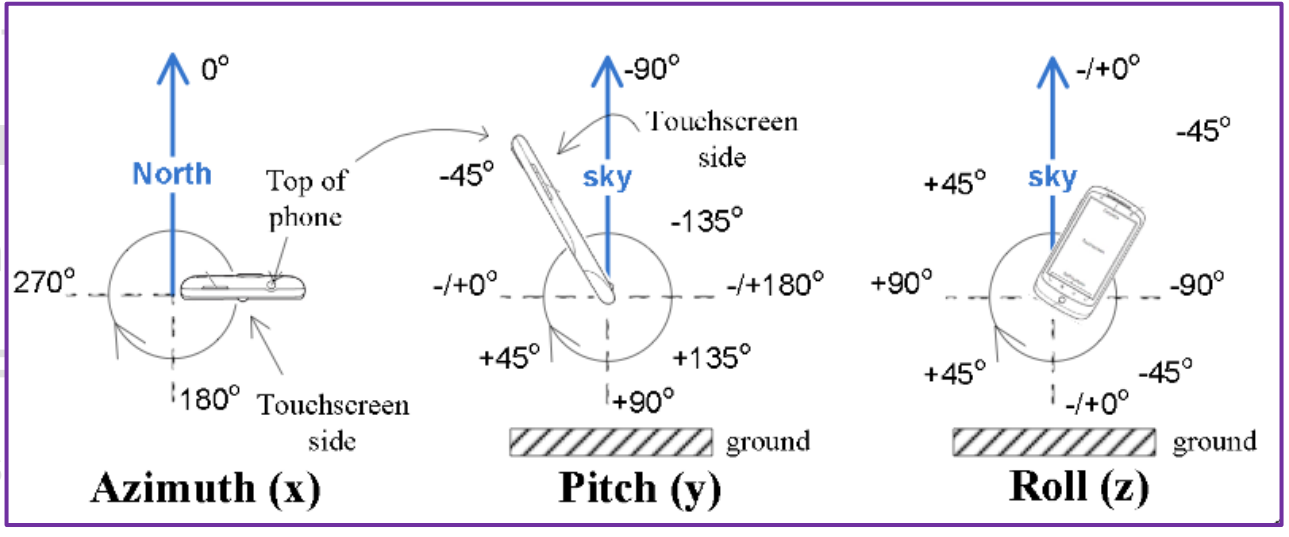


A 'Droid' bitmap in RAM on a Samsung Galaxy Nexus device after 0s, a minuscule time, 0.5s, 1s, 2s, 4s and 6s. Reboot at room temperature.

Müller, Tilo, and Michael Spreitzenbarth. "Frost." *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2013.

Ways an Attacker Can Steal Your Secrets





Side-Channel: infer PIN code through motion-detector

Cai, Liang, and Hao Chen. "TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone motion." *HotSec*. 2011.

Data Protection

TAINT TRACKING

Data Lifetime

The lifetime of sensitive data should be minimized

- The longer data remains in memory the greater its chances of being leaked to disk
- Swapping, hibernation, VM suspending, core dump, etc.

That's what we called "data exposure"

Tainting: Data Flow Tracking

Apply access policy along the data flow

- E.g., a password cannot be sent to Internet
- E.g., credit card number cannot be sent to other apps
- E.g., untrusted data cannot be executed

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Example sourced from CMU ECE

- Source

Will show the basic approach of dynamic taint analysis

Two concrete executions will be presented

Goal: evaluate whether control can be hijacked by [malicious] user input

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	6	true

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	6	true
two	2	false

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	6	true
two	2	false
j	8	true

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	6	true
two	2	false
j	8	true
l	8	true

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	7	true

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	7	true
two	2	false

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

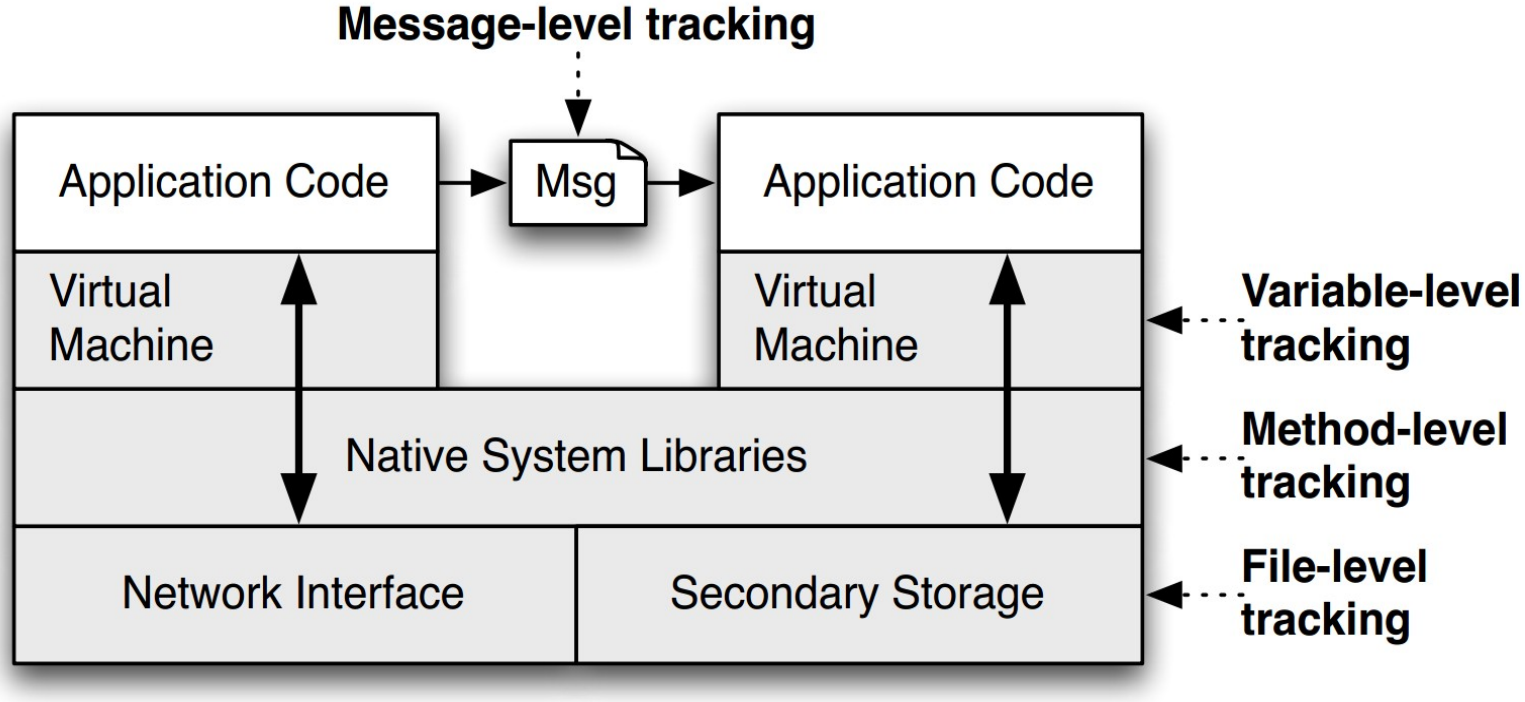
Variable	Value	Taint Status
i	7	true
two	2	false
k	4	false

Dynamic Taint Analysis

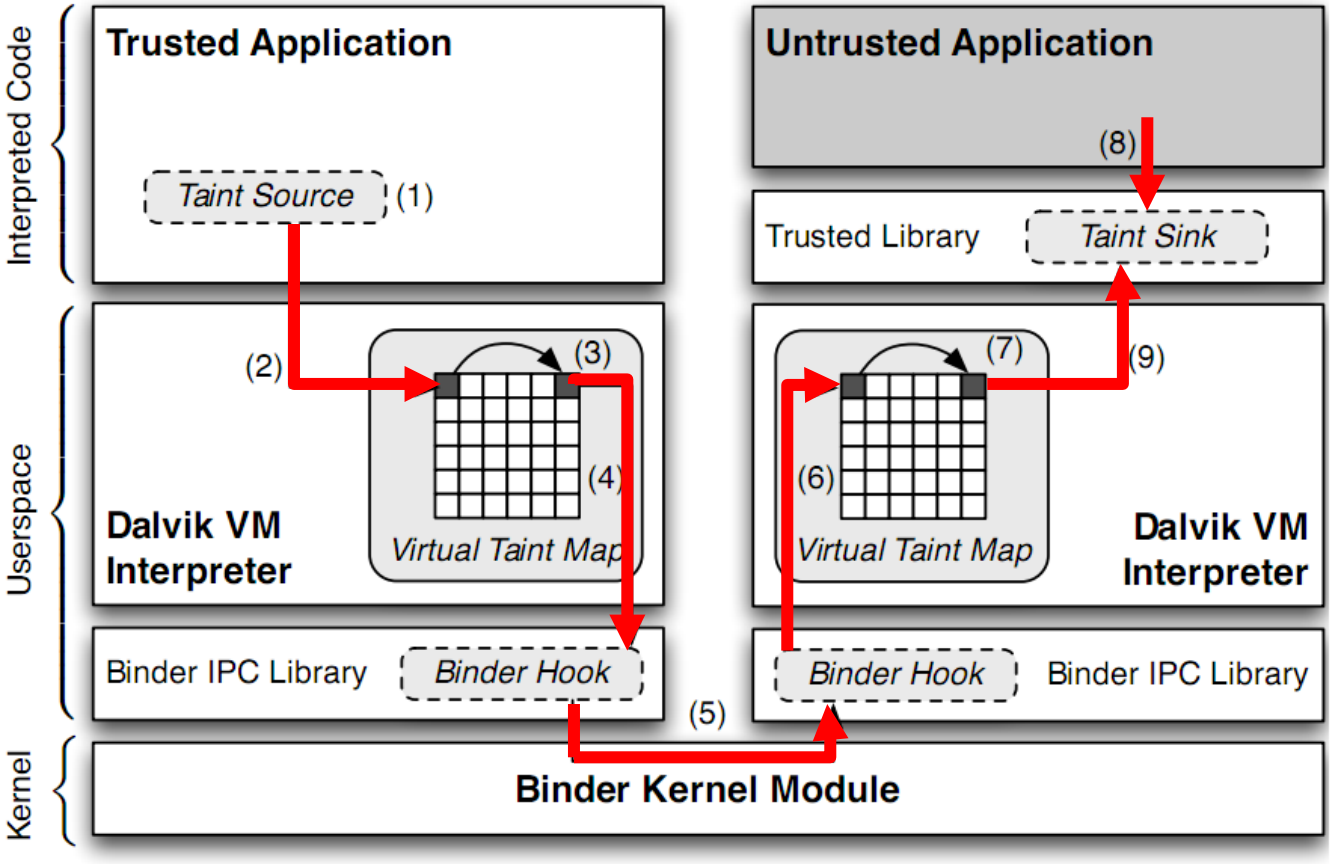
```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	7	true
two	2	false
k	4	false
l	4	false

TaintDroid

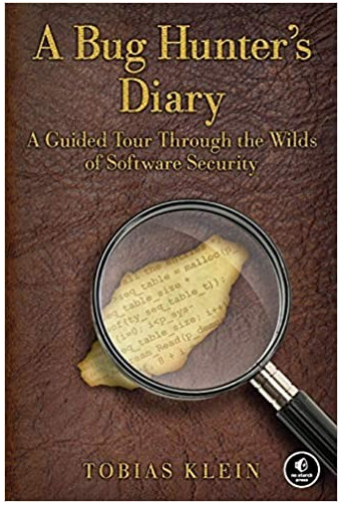

Message-level tracking
Application Code
Msg
Application Code
Virtual Machine
Virtual Machine
Native System Libraries
Network Interface
Secondary Storage
Variable-level tracking
Method-level tracking
File-level tracking

TaintDroid Data Flow



Taint Propagation

Op Format	Op Semantics	Taint Propagation	Description
const-op v_A C	$v_A \leftarrow C$	$\tau(v_A) \leftarrow \emptyset$	Clear v_A taint
move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint
unary-op v_A v_B	$v_A \leftarrow \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
binary-op v_A v_B v_C	$v_A \leftarrow v_B \otimes v_C$	$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_C)$	Set v_A taint to v_B taint $\cup$ v_C taint
binary-op v_A v_B	$v_A \leftarrow v_A \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_A) \cup \tau(v_B)$	Update v_A taint with v_B taint
binary-op v_A v_B C	$v_A \leftarrow v_B \otimes C$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
aput-op v_A v_B v_C	$v_B[v_C] \leftarrow v_A$	$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$	Update array v_B taint with v_A taint
aget-op v_A v_B v_C	$v_A \leftarrow v_B[v_C]$	$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$	Set v_A taint to array and index taint
sput-op v_A f_B	$f_B \leftarrow v_A$	$\tau(f_B) \leftarrow \tau(v_A)$	Set field f_B taint to v_A taint
sget-op v_A f_B	$v_A \leftarrow f_B$	$\tau(v_A) \leftarrow \tau(f_B)$	Set v_A taint to field f_B taint
iput-op v_A v_B f_C	$v_B(f_C) \leftarrow v_A$	$\tau(v_B(f_C)) \leftarrow \tau(v_A)$	Set field f_C taint to v_A taint
iget-op v_A v_B f_C	$v_A \leftarrow v_B(f_C)$	$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$	Set v_A taint to field f_C and object reference taint



DEFENDING MALICIOUS INPUT

Two Usages of Data Flow Tracking

Usage-1: protect critical data

- E.g., track data to ensure none is leaked

Usage-2: defend against suspicious input data

- E.g., one input data cannot flow to return address!

How does a Hacker Search a Bug?

Step-0: Chose a library (open-source, of course)

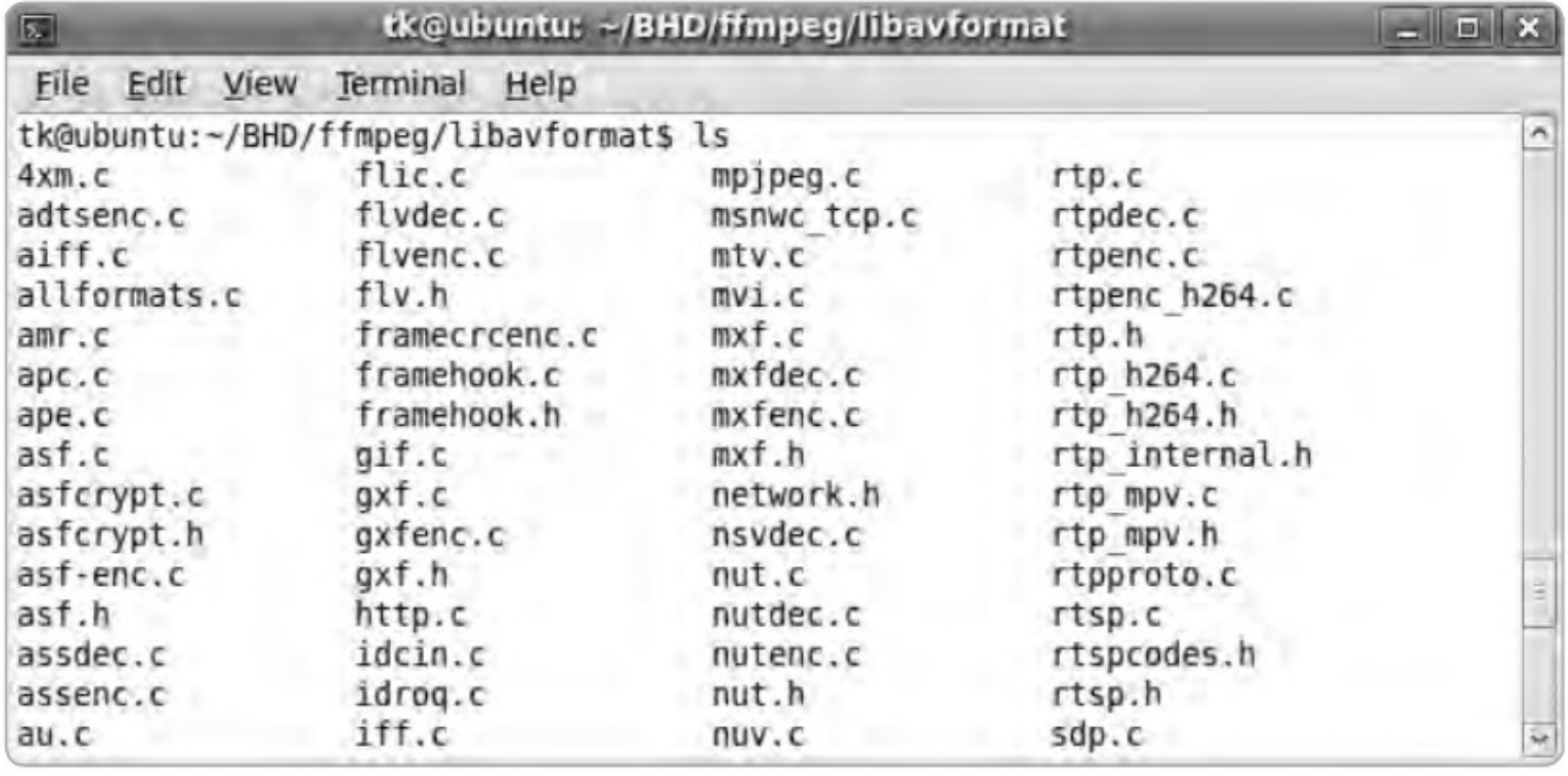
- Let's try ffmpeg, which is used in Chrome, VLC, Mplayer...
- There are also rumors that YouTube uses it for conversion

Step-1: List the demuxers of ffmpeg

Step-2: Identify the input data

Step-3: Trace the input data

Step-1: List the Demuxers of FFmpeg



Step-2: Identify the Input Data

***demuxername_read_header*()**

- Most demuxers declear a function called it
- Takes a parameter of type AVFormatContext
- This function initializes a pointer:

```
[..]
ByteIOContext *pb = s->pb;
[..]
```

- Many different get_something() are used to extract pb data
  - E.g., get_le32(), get_buffer()
- Conclusion: pb is a pointer to the input data of media files

Step-3: Trace the Input Data

Start to read the first demuxer: libavformat/4xm.c

```
[..]
93 static int fourxm_read_header(AVFormatContext *s,
94                                AVFormatParameters *ap)
95 {
96   ByteIOContext *pb = s->pb;
..
101  unsigned char *header;
..
103  int current_track = -1;
..
106  fourxm->track_count = 0;
107  fourxm->tracks = NULL;
..
```

```
 ..
120     /* allocate space for the header and load the whole thing */
121     header = av_malloc(header_size);
122     if (!header)
123         return AVERROR(ENOMEM);
124     if (get_buffer(pb, header, header_size) != header_size)
125         return AVERROR(EIO);
 ..
160     } else if (fourcc_tag == strk_TAG) {
161         /* check that there is enough data */
162         if (size != strk_SIZE) {
163             av_free(header);
164             return AVERROR_INVALIDDATA;
165         }
166         current_track = AV_RL32(&header[i + 8]);
```

int unsigned int

Step-3: Trace the Input Data

Supplying a value >= 0x80000000 for current_track causes a change in sign that results in current_track being interpreted as negative

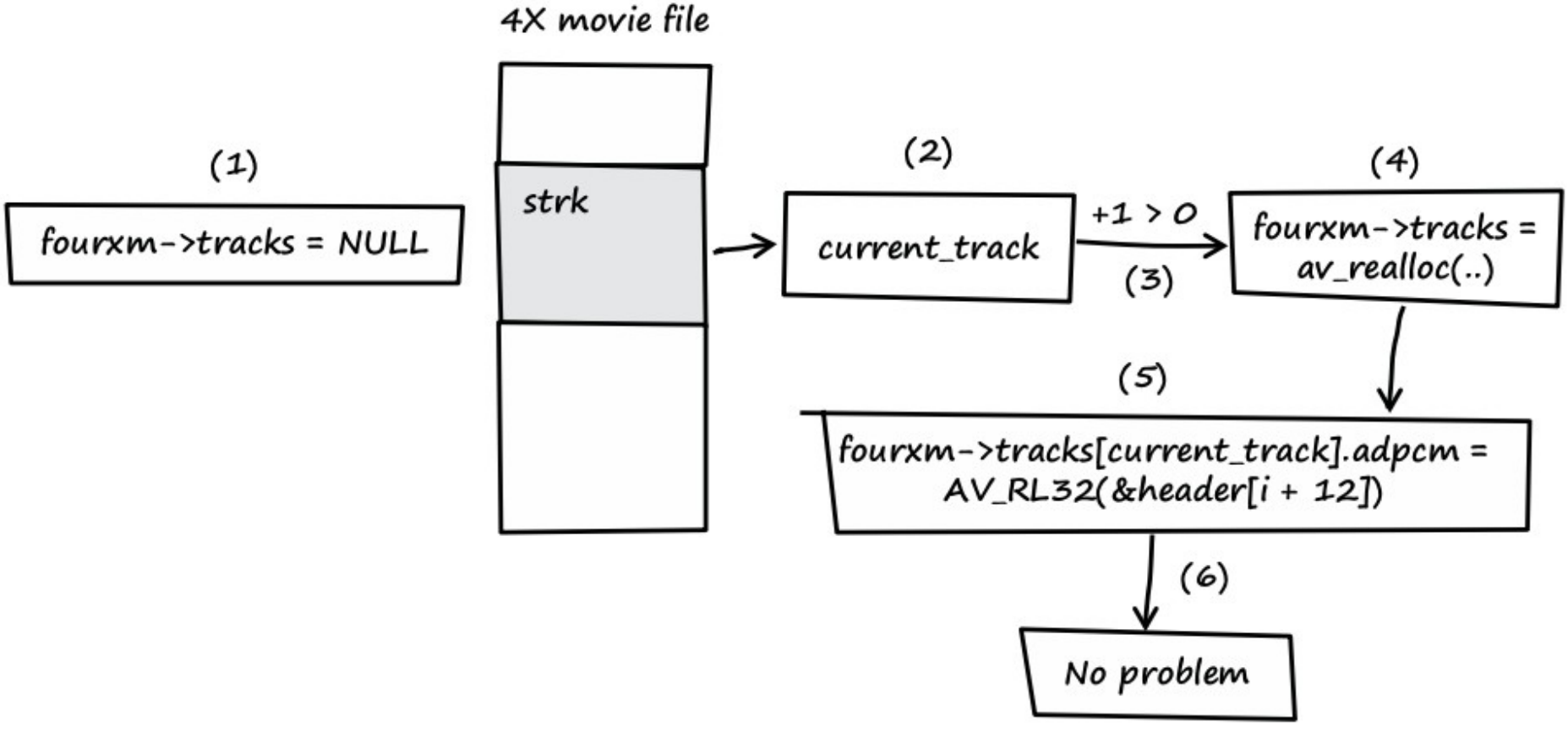
The if statement in line 167 will always return FALSE

Thus, the buffer allocation in line 171 will never be reached

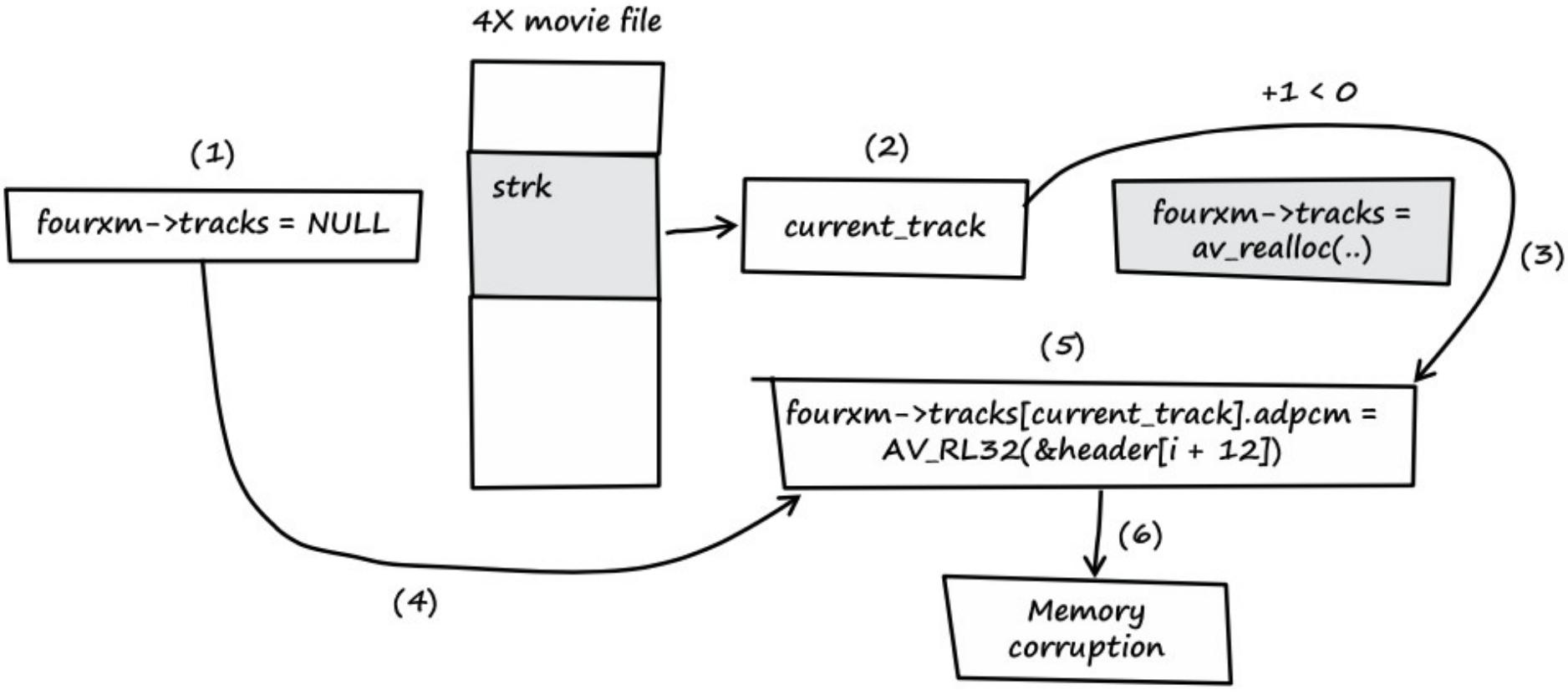
Step-3: Trace the Input Data

```
167         if (current_track + 1 > fourxm->track_count) {
168             fourxm->track_count = current_track + 1;
169             if((unsigned)fourxm->track_count >= UINT_MAX / sizeof(AudioTrack))
170               return -1;
171             fourxm->tracks = av_realloc(fourxm->tracks,
172                 fourxm->track_count * sizeof(AudioTrack));
173             if (!fourxm->tracks) {
174               av_free(header);
175               return AVERROR(ENOMEM);
176             }
177         }
178         fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
179         fourxm->tracks[current_track].channels = AV_RL32(&header[i + 36]);
180         fourxm->tracks[current_track].sample_rate = AV_RL32(&header[i + 40]);
181         fourxm->tracks[current_track].bits = AV_RL32(&header[i + 44]);
[..]
```

Step-3: Trace the Input Data



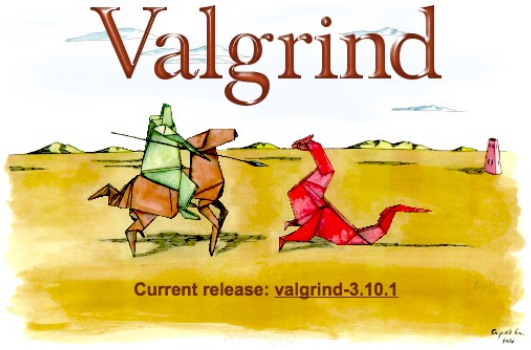
Step-3: Trace the Input Data



Question

How can we use taint tracking to find such bugs?

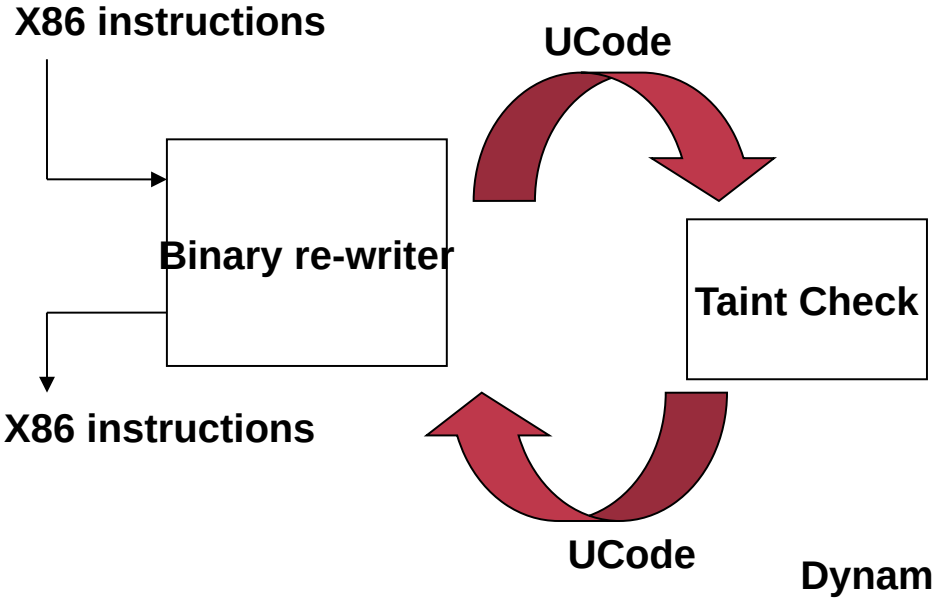
TaintCheck: Basic Ideas



1. Program execution normally derived from trusted sources, not attacker input
2. Mark all input data to the computer as "tainted" (e.g., network, stdin, etc.)
3. Monitor program execution and track how tainted data propagates (follow bytes, arithmetic operations, jump addresses, etc.)
4. Detect when tainted data is used in dangerous ways

Add Taint Checking code

- TaintCheck first runs the code through an emulation environment (Valgrind) and adds instructions to monitor tainted memory



Dynamic taint analysis

TaintCheck Detection Modules

TaintSeed: Mark untrusted data as tainted

TaintTracker: Track each instruction, determine if result is tainted

TaintAssert: Check is tainted data is used dangerously

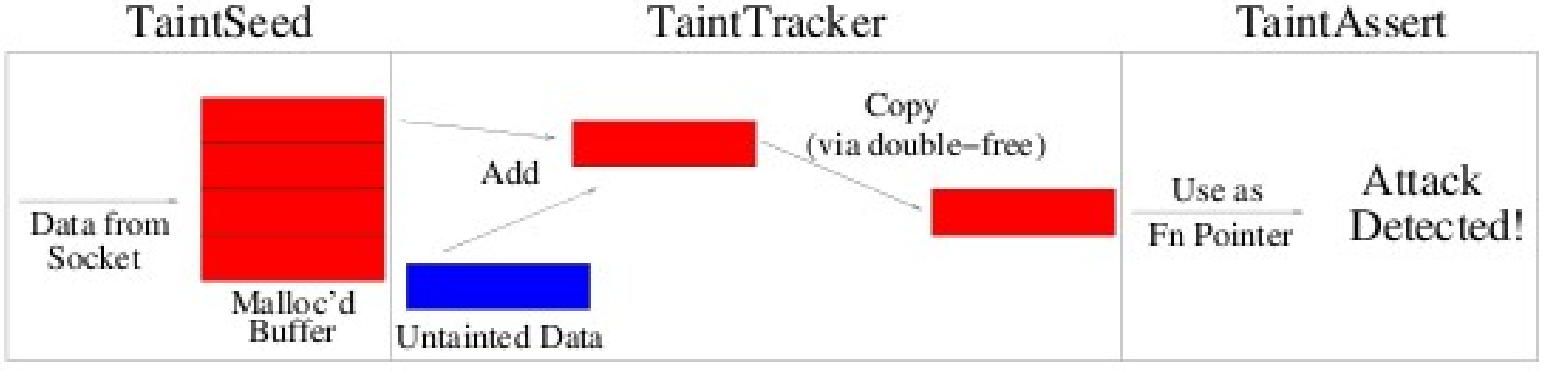


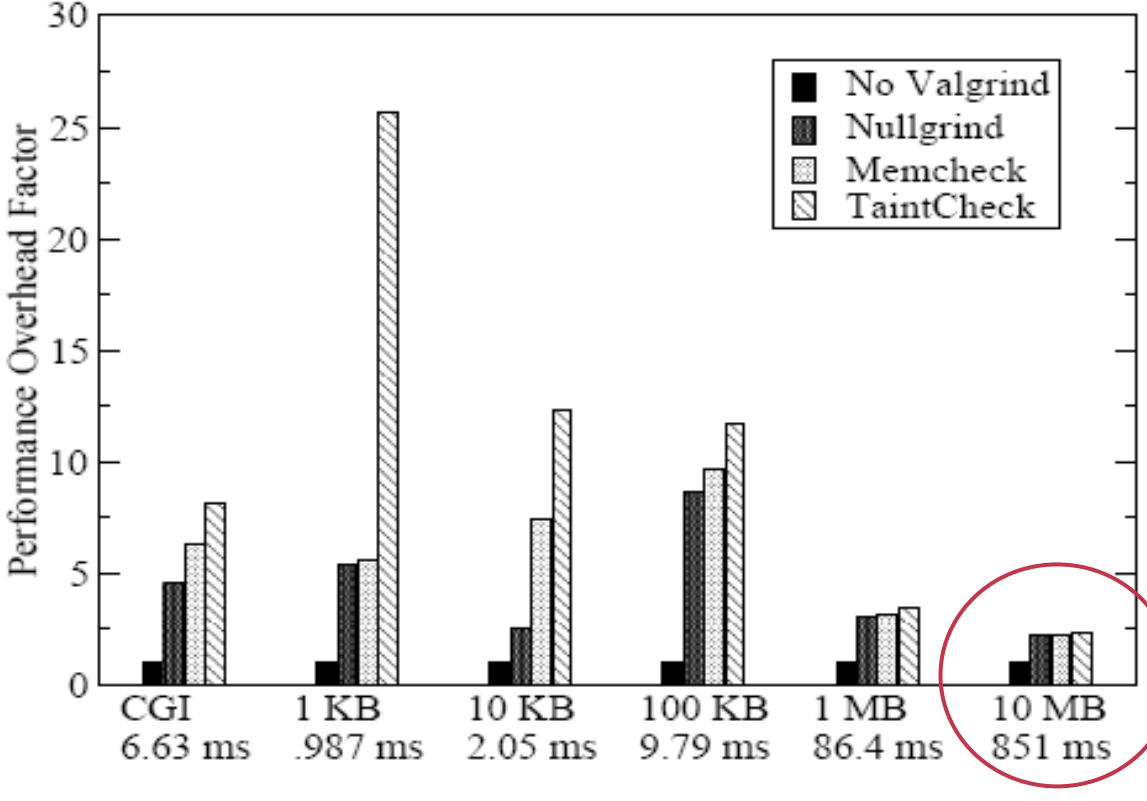
Figure 1. TaintCheck detection of an attack. (Exploit Analyzer not shown).

Performance Evaluation – CPU Bound Process

- Hardware: 2.00 GHz Pentium 4, 512 MB RAM, RedHat 8.0
- Application: bzip2(15mb)
  - Normal runtime 8.2s
  - Valgrind nullgrind skin runtime: 25.6s (3.1x)
  - Memcheck runtime: 109s (13.3x)
  - TaintCheck runtime: 305s **(37.2x)**

Performance Overhead of Apache

- A more representative case, network and I/O



Better performance

SECURE CHANNEL

Encryption Properties

encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

```
encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world
```

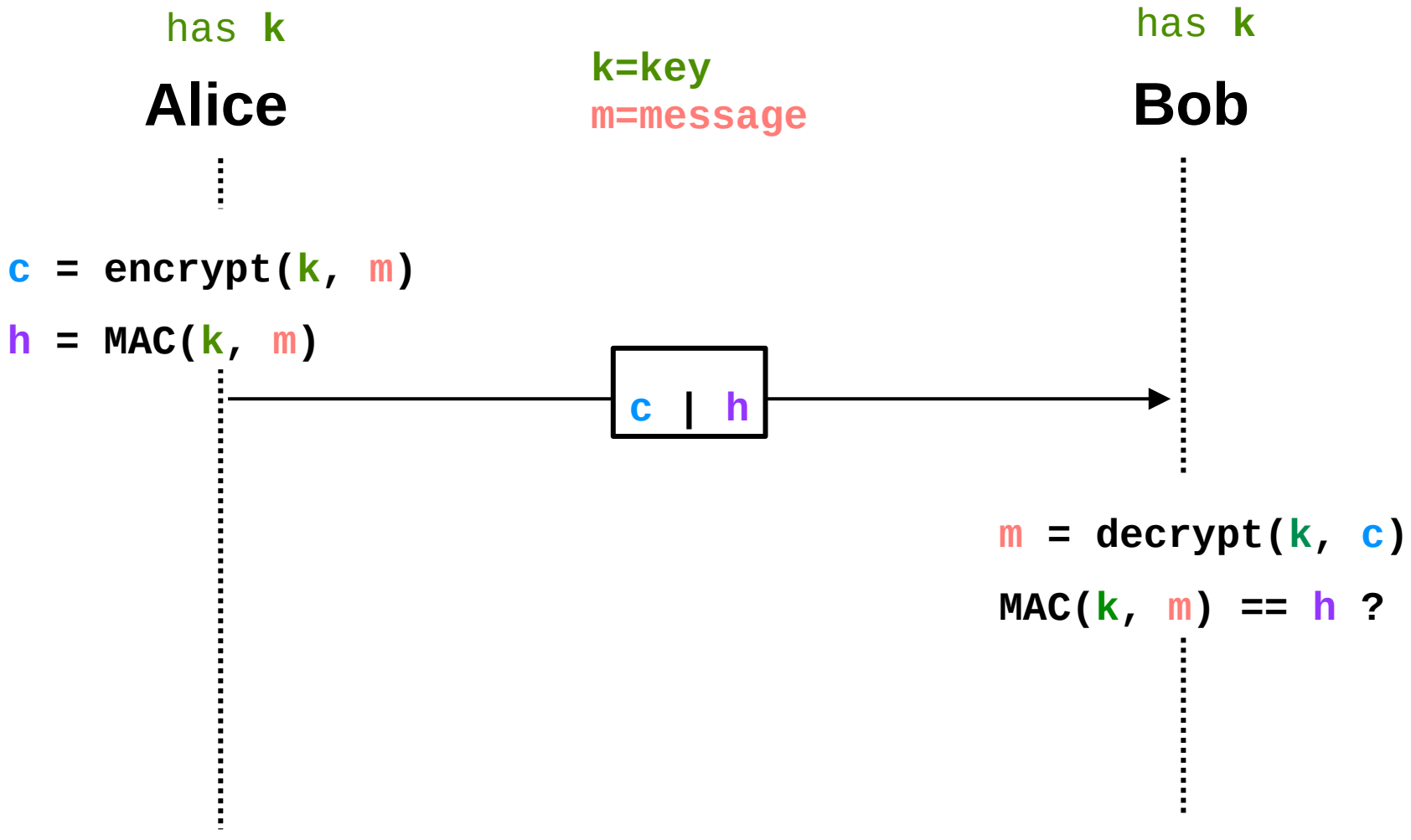
property: given the **ciphertext**, it is (virtually) impossible to obtain the **message** without knowing the **key**

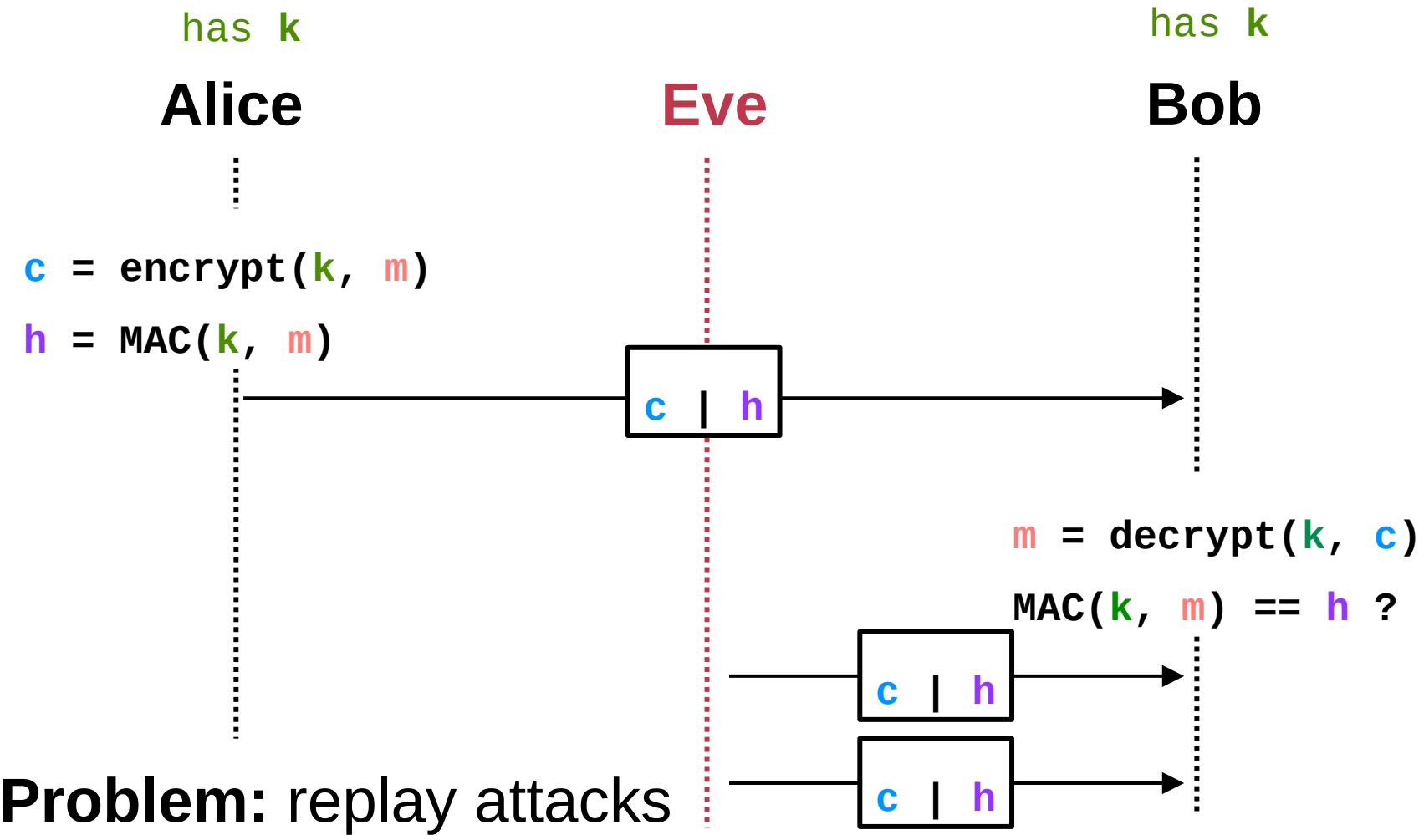
MAC(key, message) → token

```
MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c
```

property: given the **message**, it is (virtually) impossible to obtain the **token** without knowing the **key**

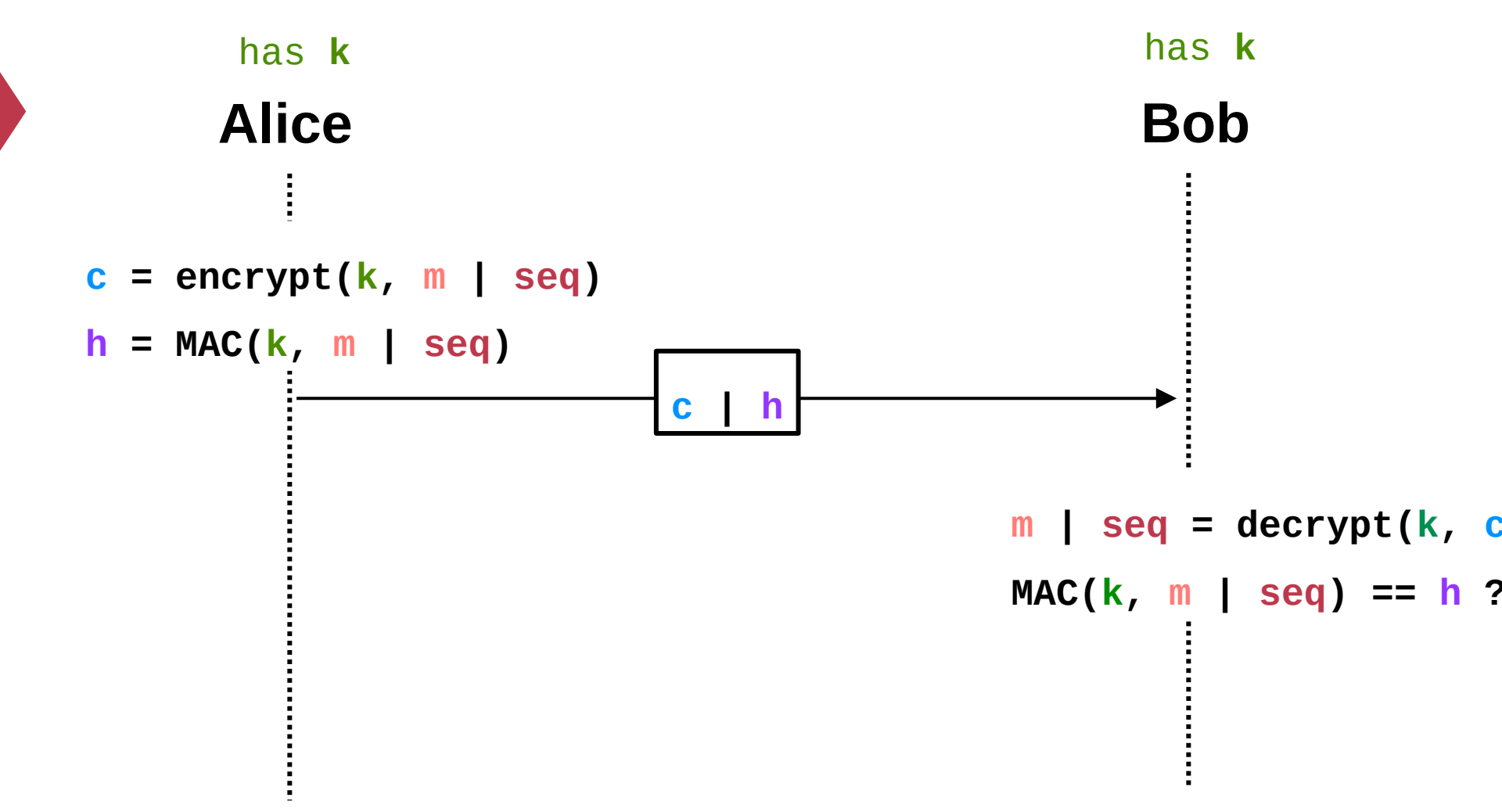
(it is also impossible to go in the reverse direction)

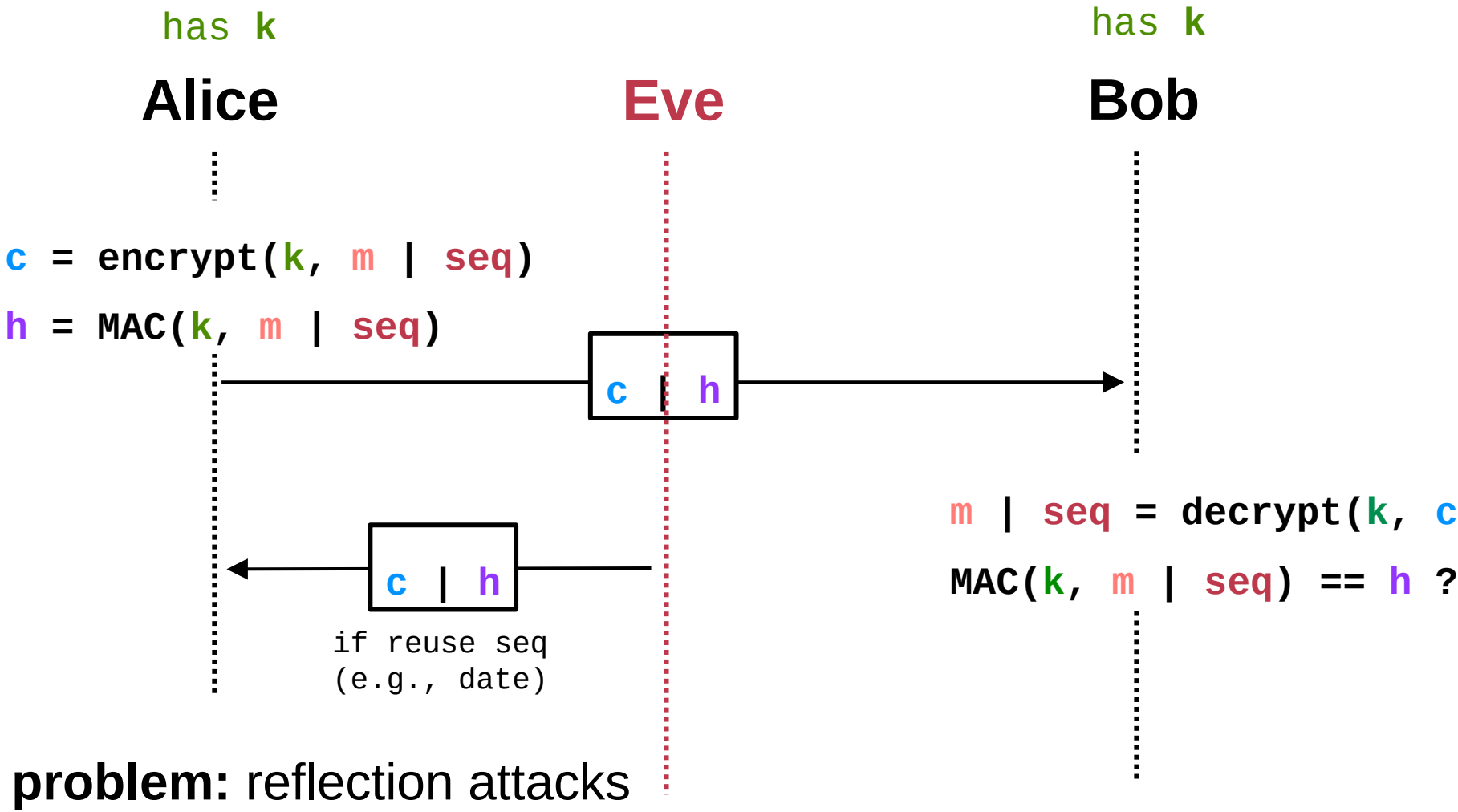

has k
Alice
k=key
m=message
has k
Bob
c = encrypt(k, m)
h = MAC(k, m)
c | h
m = decrypt(k, c)
MAC(k, m) == h ?



Problem: replay attacks

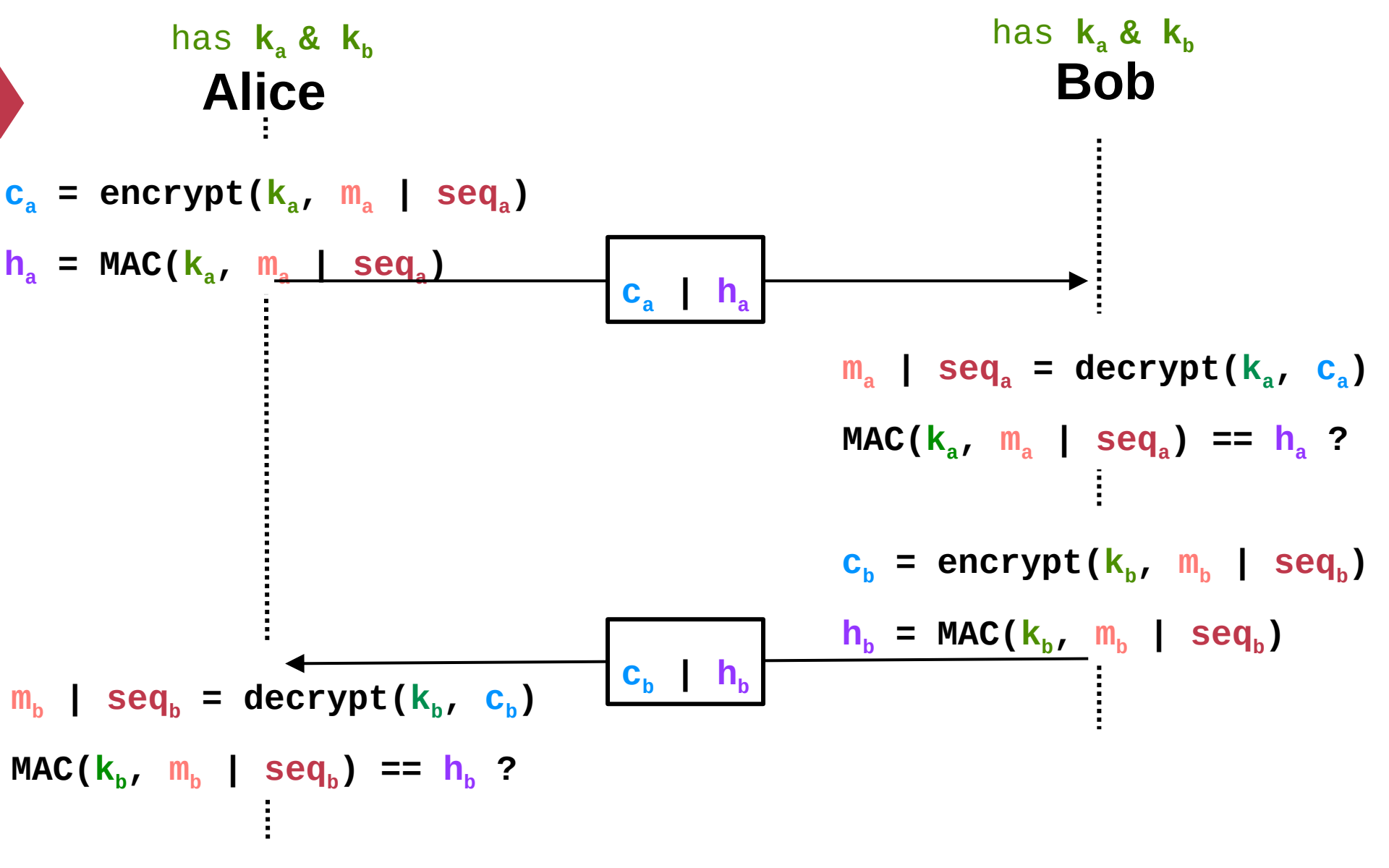
(adversary could intercept a message, re-send it at a later time)


has k
Alice
has k
Bob
c = encrypt(k, m | seq)
h = MAC(k, m | seq)
c | h
m | seq = decrypt(k, c
MAC(k, m | seq) == h ?



problem: reflection attacks

(adversary could intercept a message, re-send it later in the opposite direction)

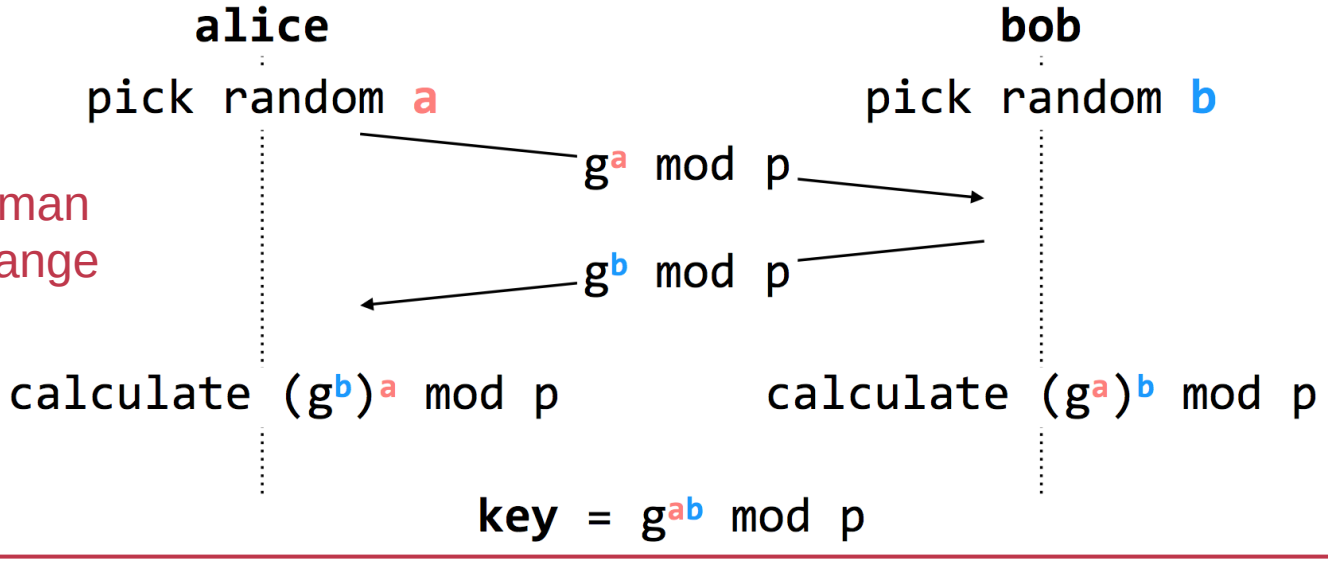

has k_a & k_b
Alice
has k_a & k_b
Bob
c_a = encrypt(k_a, m_a | seq_a)
h_a = MAC(k_a, m_a | seq_a)
c_a | h_a
m_a | seq_a = decrypt(k_a, c_a)
MAC(k_a, m_a | seq_a) == h_a ?
c_b = encrypt(k_b, m_b | seq_b)
h_b = MAC(k_b, m_b | seq_b)
c_b | h_b
m_b | seq_b = decrypt(k_b, c_b)
MAC(k_b, m_b | seq_b) == h_b ?

problem: how do the parties know the keys?

known: p (prime), g

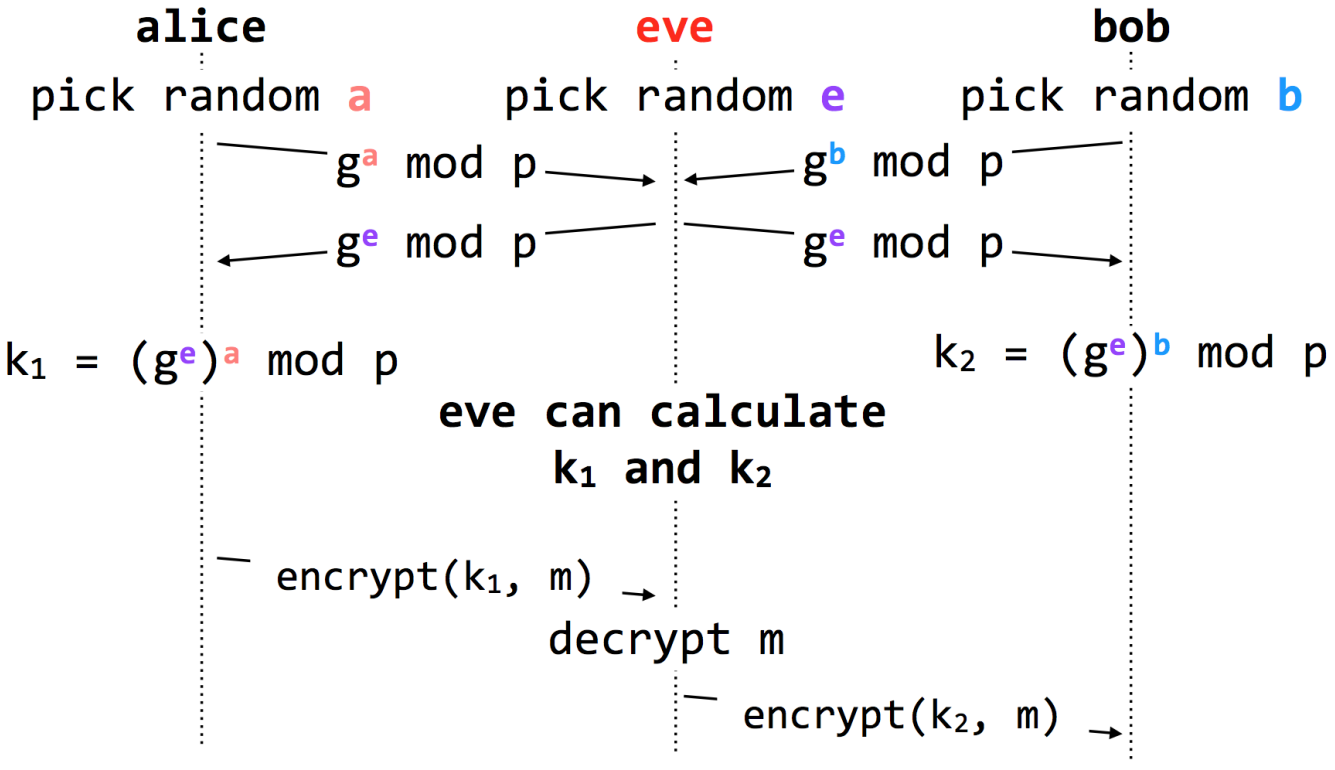
property: given $g^r \bmod p$, it is (virtually) impossible to determine **r** *even if* you know **g** and **p**

Diffie-Hellman Key Exchange



$(g^b \bmod p)^a \bmod p \quad = \quad g^{ab} \bmod p \quad = \quad (g^a \bmod p)^b \bmod p$

Diffie-Hellman Key Exchange & Man-in-the-Middle



problem: alice and bob don't know they're not communicating directly

RSA Algorithm

cryptographic signatures

allow users to verify identities using public-key cryptography

users generate key pairs

the two keys in the pair are related mathematically

`{public_key, secret_key}`

```
sign(secret_key, message) → sig
verify(public_key, message, sig) → yes/no
```

Public Key Distribution

1. Alice remembers the key she used last time

- Easy to implement
- Effective against subsequent man-in-the-middle attacks
- Doesn't protect against MITM attacks the first time around
- Doesn't allow parties to change keys

2. Consult some authority that knows everyone's public key

- Does not scale (client asks for a PK for every new name)
- Alice needs server's public key beforehand

Public Key Distribution

3. Authority, but pre-compute responses

- Authority creates signed messages: {Bob, PK_bob}_{SK_as}
- Anyone can verify the authority signed this message, given PK_as
- When Alice wants to talk to Bob, she needs a signed message from the authority, but it doesn't matter where this message comes from as long as the signature checks out
  - I.e., Alice could retrieve the message from a different server
- This signed message is a **certificate**
- More scalable

Review: Two Types of Encryption

Symmetric key encryption

- Single key (symmetric) is shared between parties, kept secret from everyone else
- Ciphertext = (M)^K

Asymmetric key encryption

- Keys come in pairs, public and private
- Secret: (M)^K-public
- Authentic: (M)^K-private

Questions on Certificate Authorities

Who should run the certificate authority?

How does the browser get this list of CAs?

- Generally they come with the browser

How does the CA build its table of names <-> public keys?

- Have to agree on how to name principals, and
- Need a mechanism to check that a key corresponds to a name

What if a CA makes a mistake?

- Need a way to revoke certificates: Expiration date? Not useful for immediate problems
- Publish certificate revocation list? Works in theory, not as well in practice
- Query online server to check certificate freshness? Not a bad idea