CMSC414 Computer and Network Security

Midterm 1 Recap

Yizheng Chen | University of Maryland
surrealyz.github.io

Mar 7, 2024

What does it mean to be secure?

- Too difficult for attackers
- Too expensive
- Lower ROI than the next target
- …
- We raise the bar for attackers to succeed

Security Principle: Security is Economics

- Security is often a cost-benefit analysis where someone needs to make a decision regarding how much security is worth
  - The expected benefit of your defense should be proportional to the expected cost of the attack
- Focus your energy on securing the weakest links
  - A system is only as secure as the weakest link

Security Principle: Kerkhoff’s Principle

- Kerckhoffs' principle is a fundamental concept in cryptography. It states that the security of a cryptographic system shouldn't rely on the secrecy of the algorithm.
- Don’t rely on security through obscurity.

Exercise: What's wrong with this code?

```
void vulnerable() {
    size_t len;
    char *buf;

    len = read_int_from_network();
    buf = malloc(len+5);
    read(fd, buf, len);
    ...
}
```

size_t is a special unsigned integer type defined in the standard library of C and C++ languages.

Return to libc

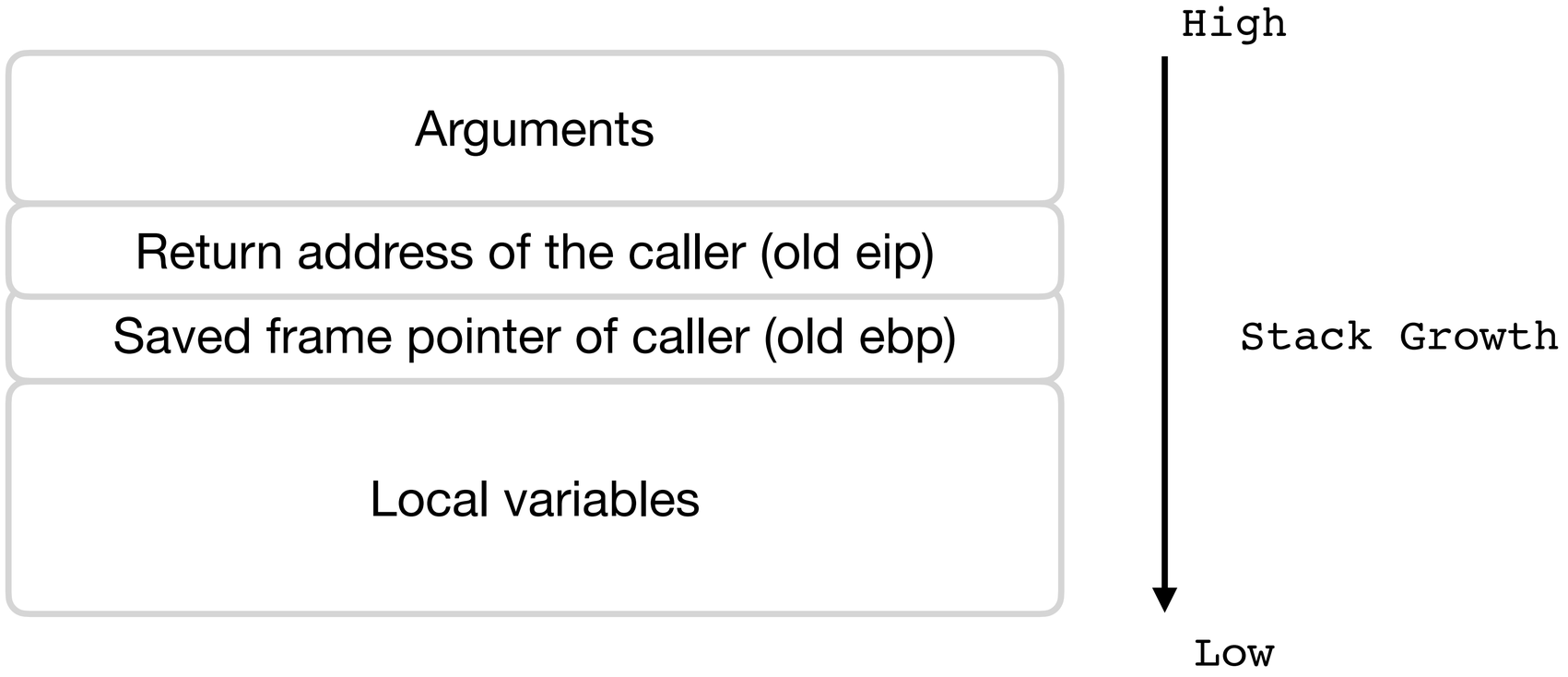
Non-Executable Pages

- Idea: Most programs don't need memory that is both written to and executed, so make portions of memory either executable or writable but not both
  - Stack, heap, and static data: Writable but not executable
  - Code: Executable but not writable
- Also known as
  - W^X (write XOR execute)
  - DEP (Data Execution Prevention, name used by Windows)
  - No-execute bit

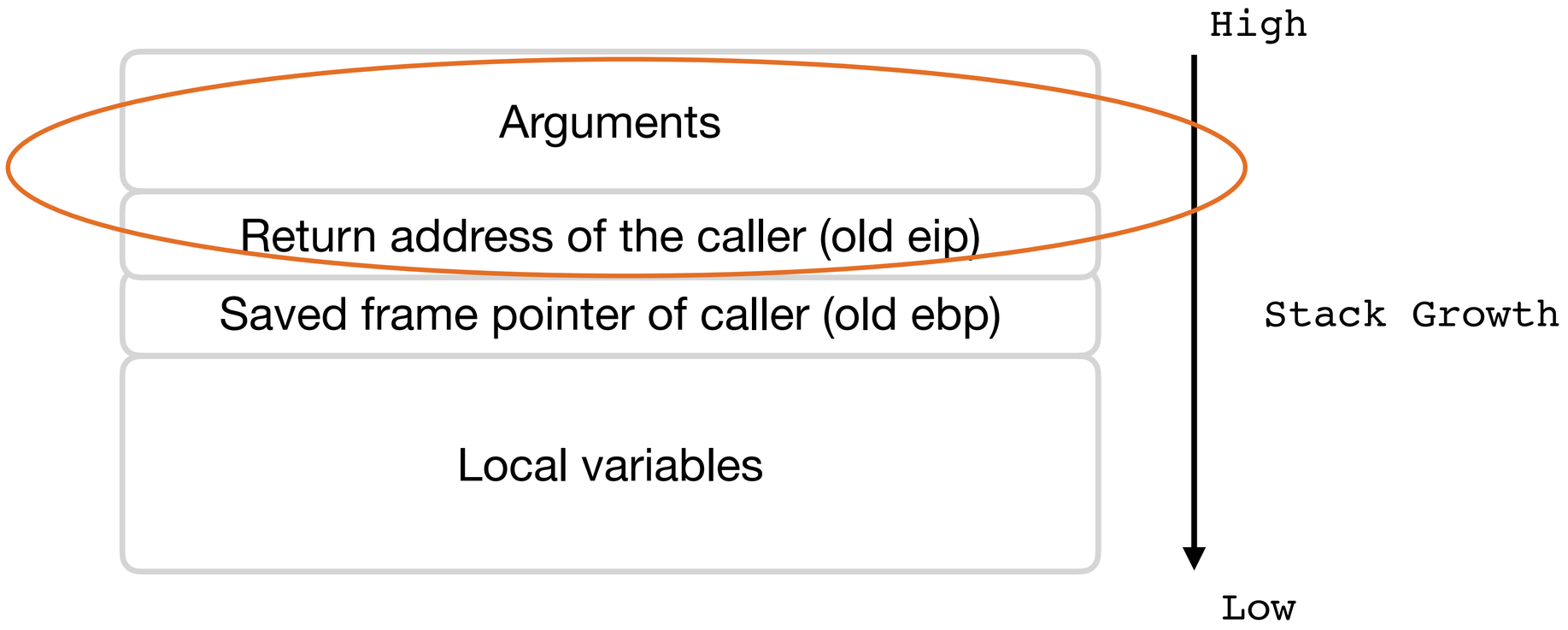
How to subvert non-executable pages?

Idea: return to existing code in memory

Stack Frame of a Function



Stack Frame of a Function



- Per the x86 calling convention, each program **expects arguments to be placed directly above the RIP** (Return Instruction Pointer, **old eip**)
- Callee saves ebp, push local vars

Return into libc: a real call

Goal: use buffer overflow to fake call system(“rm -rf /”)

NAME top

system – execute a shell command

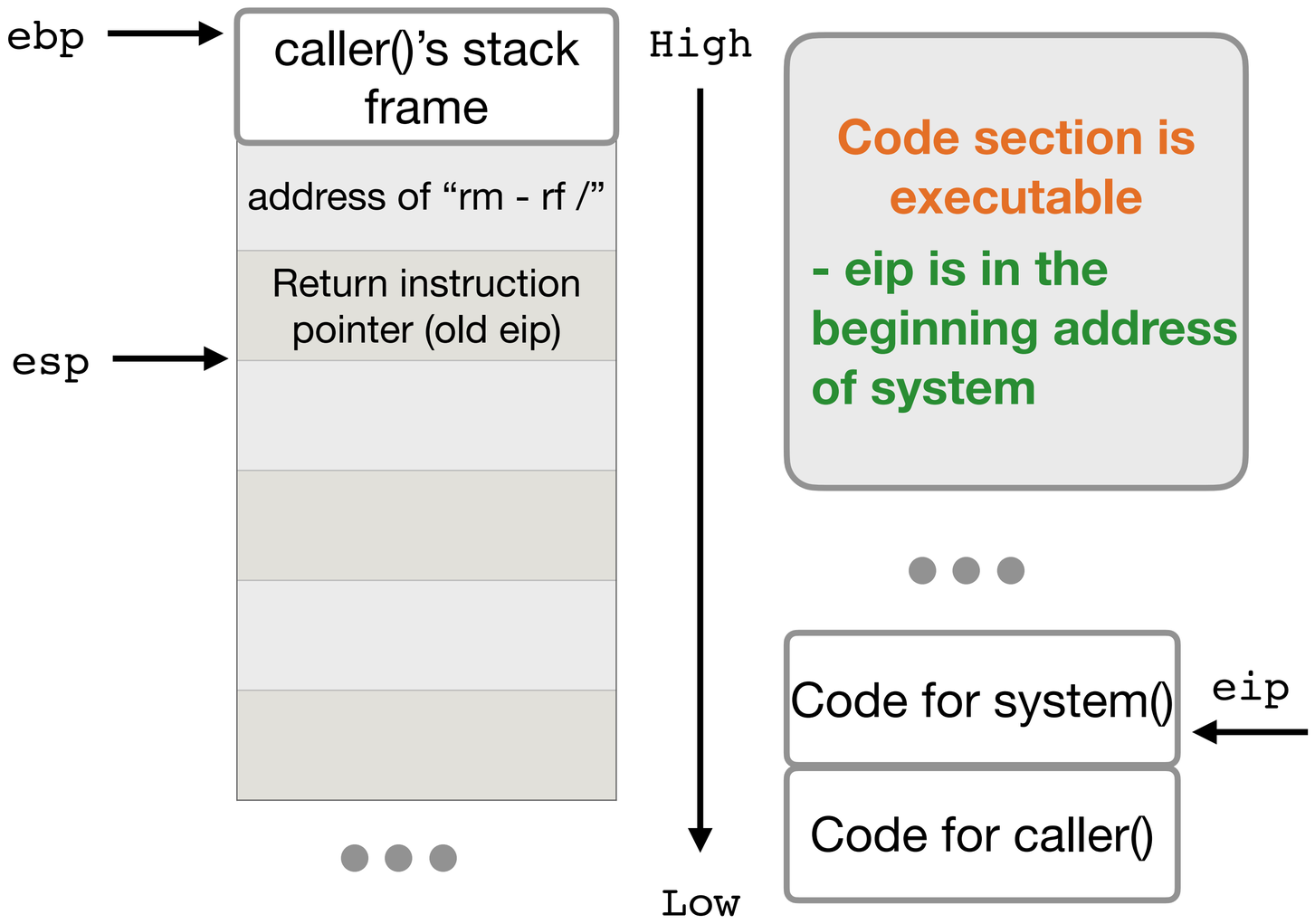
LIBRARY top

Standard C library (*libc*, *-lc*)

SYNOPSIS top

```
#include <stdlib.h>

int system(const char *command);
```



Return into libc: a real call

Goal: use buffer overflow to fake call system(“rm -rf /”)

NAME top

system – execute a shell command

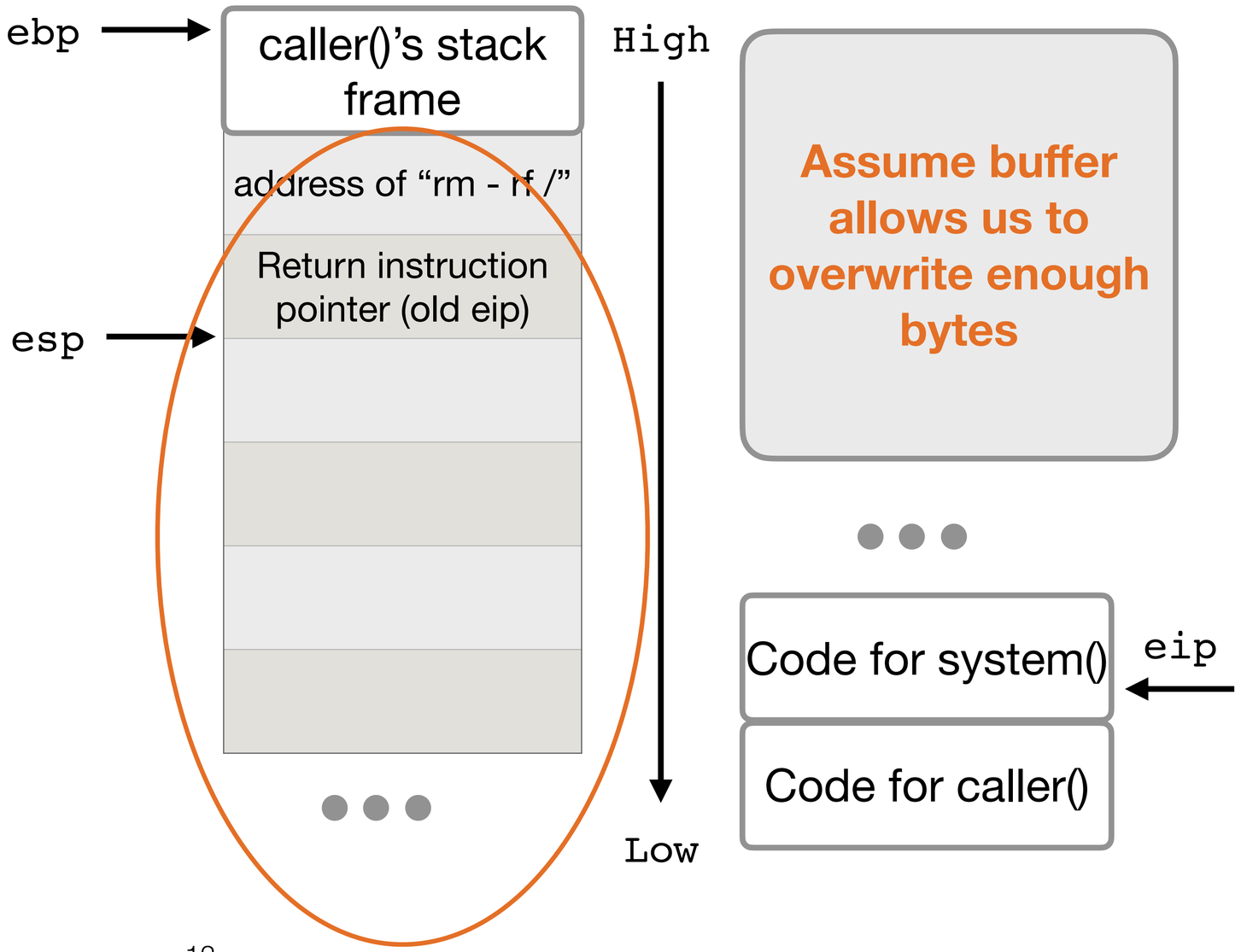
LIBRARY top

Standard C library (*libc*, *-lc*)

SYNOPSIS top

```
#include <stdlib.h>

int system(const char *command);
```

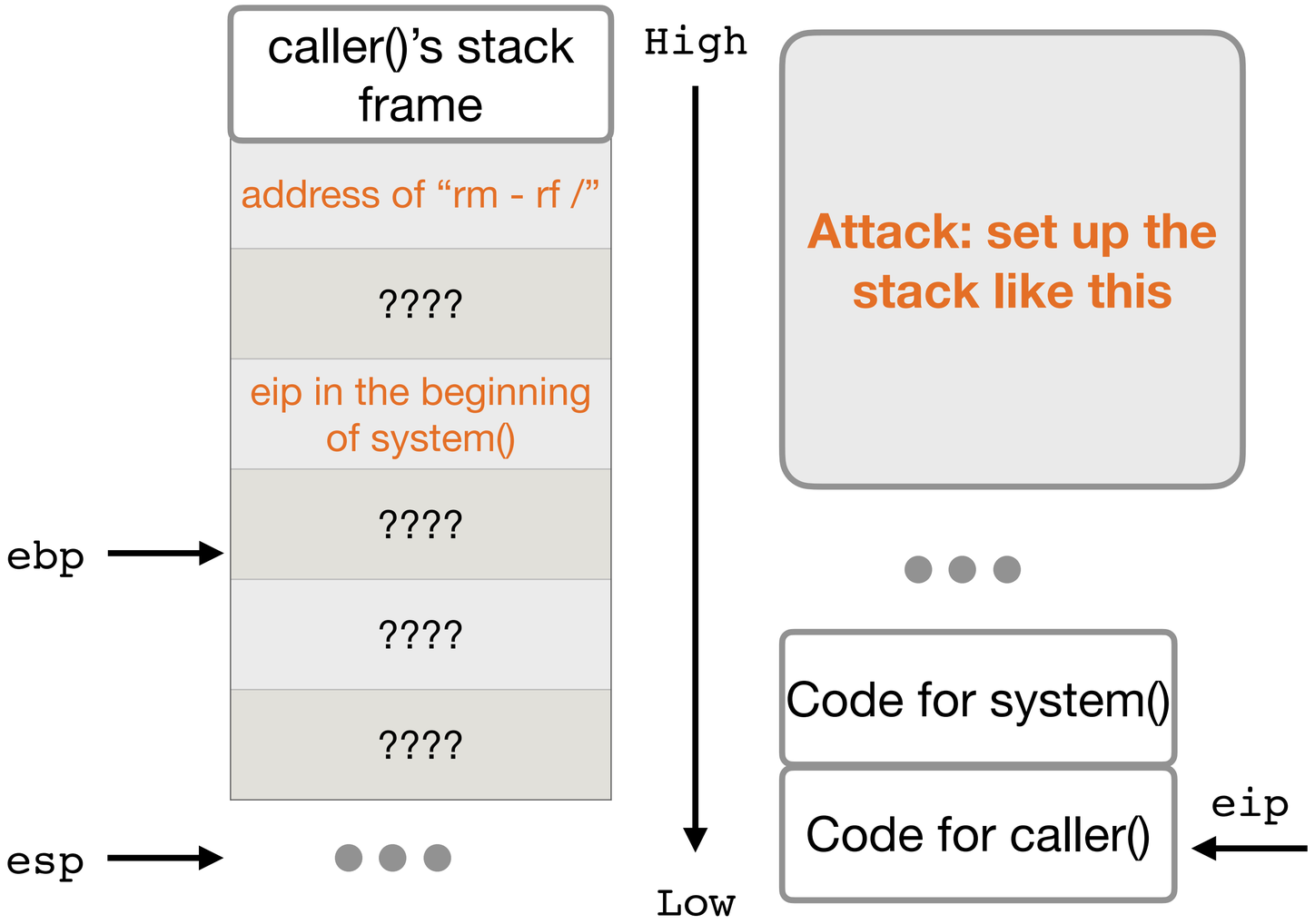


Exercise: Go through leave return

Check that we can call system("rm -rf /")

after executing leave ret

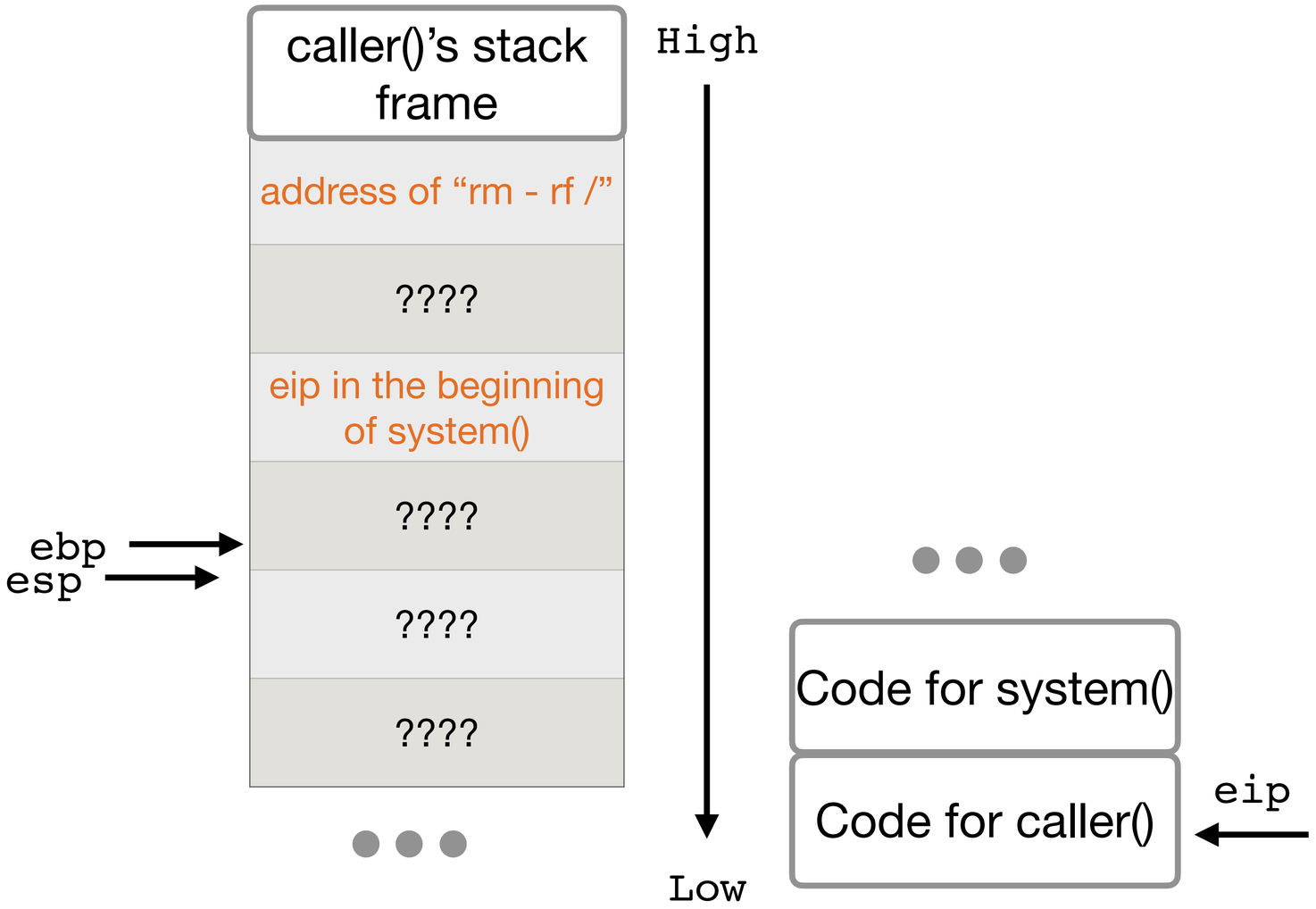
- leave
  - mov %ebp %esp
  - pop %ebp
- ret: pop %eip



Exercise: Go through leave return

restore stack pointer
(mov %ebp %esp)

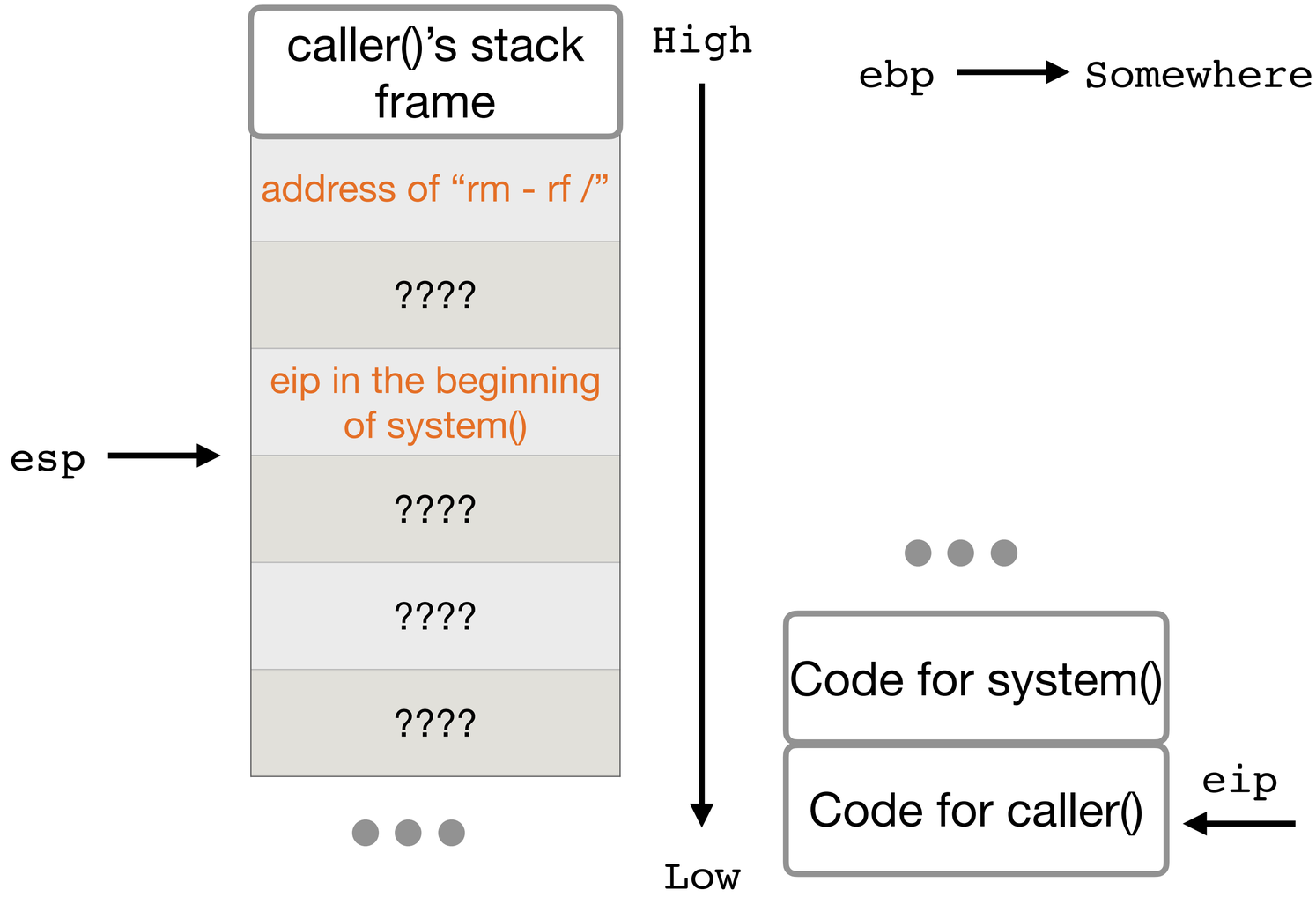
- leave
  - mov %ebp %esp
  - pop %ebp
- ret: pop %eip



Exercise: Go through leave return

restore the base pointer
(pop %ebp)

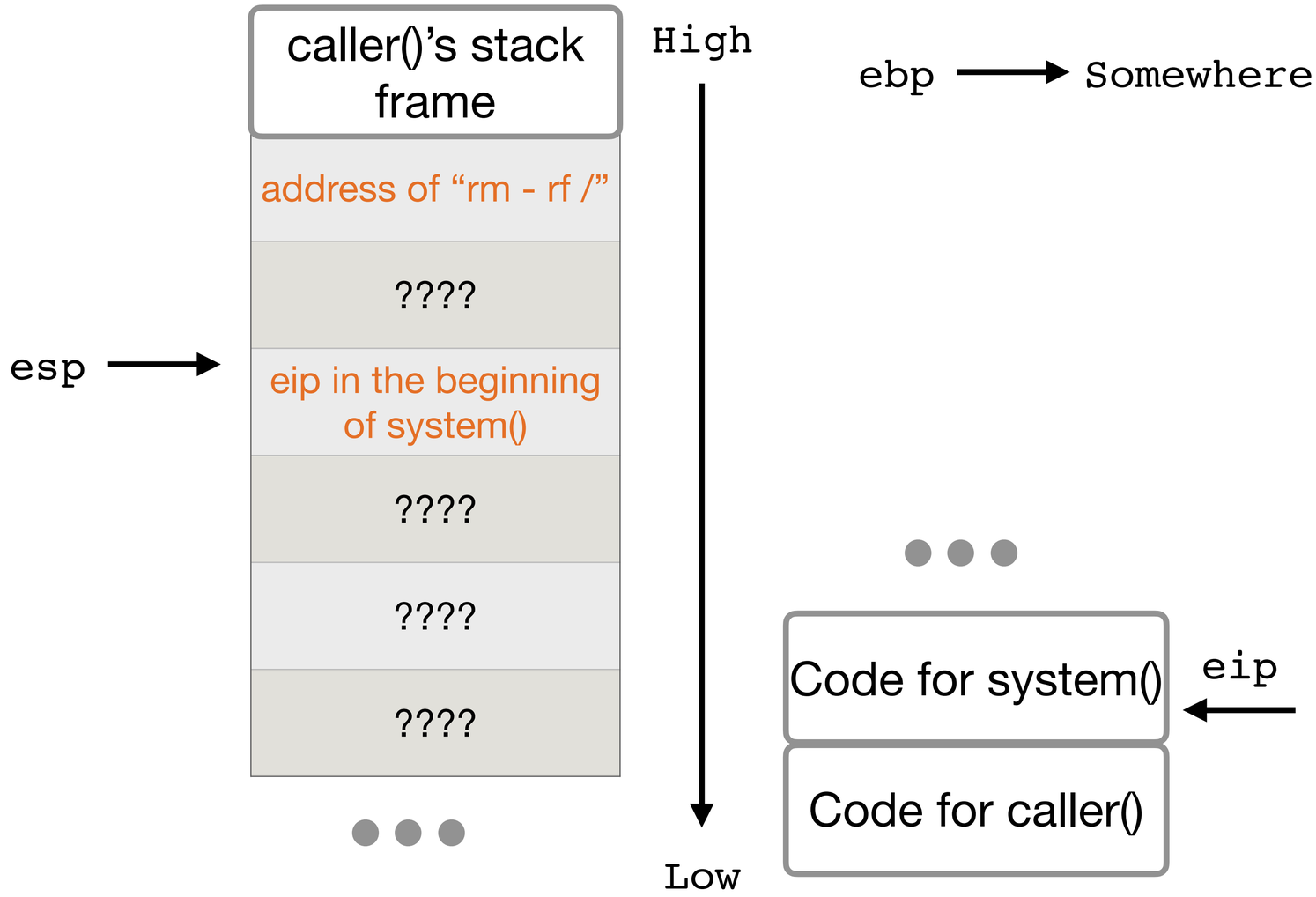
- leave
  - mov %ebp %esp
  - pop %ebp
- ret: pop %eip



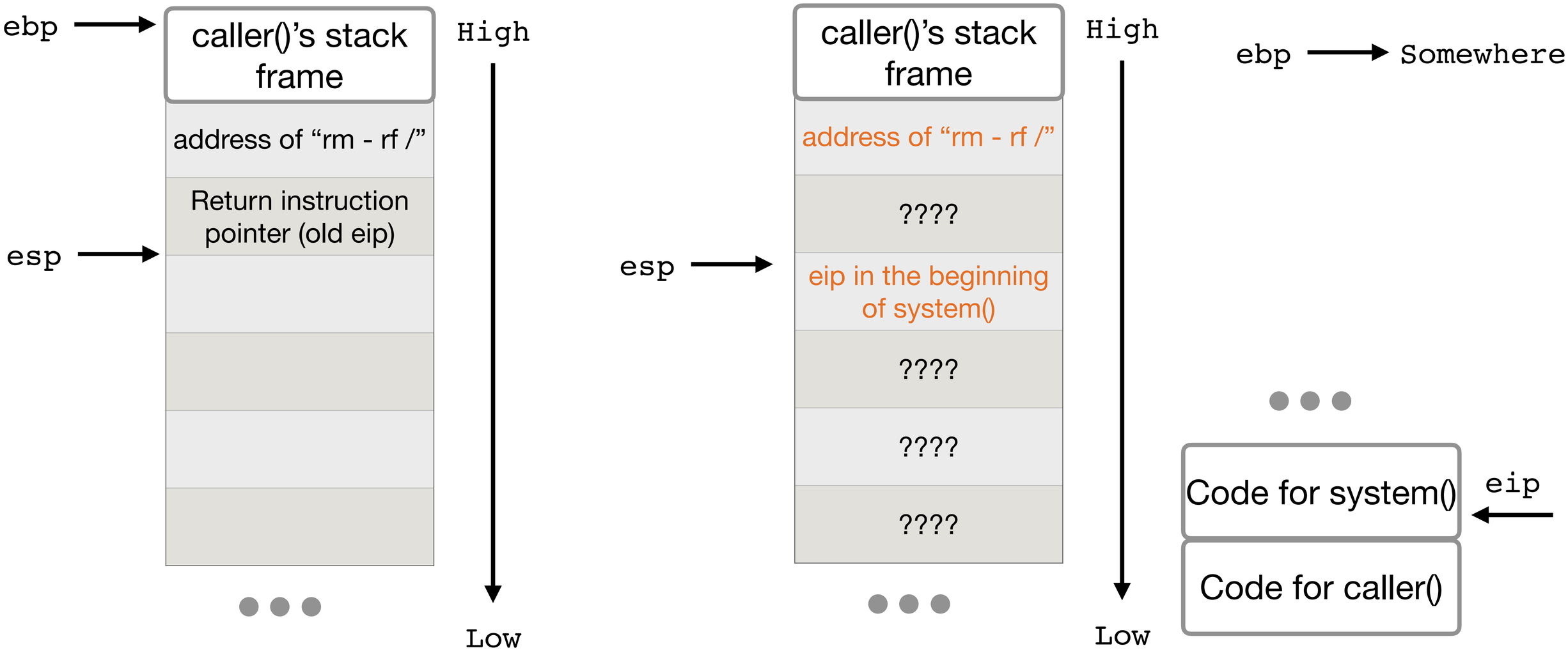
Exercise: Go through leave return

restore the base pointer
(pop %ebp)

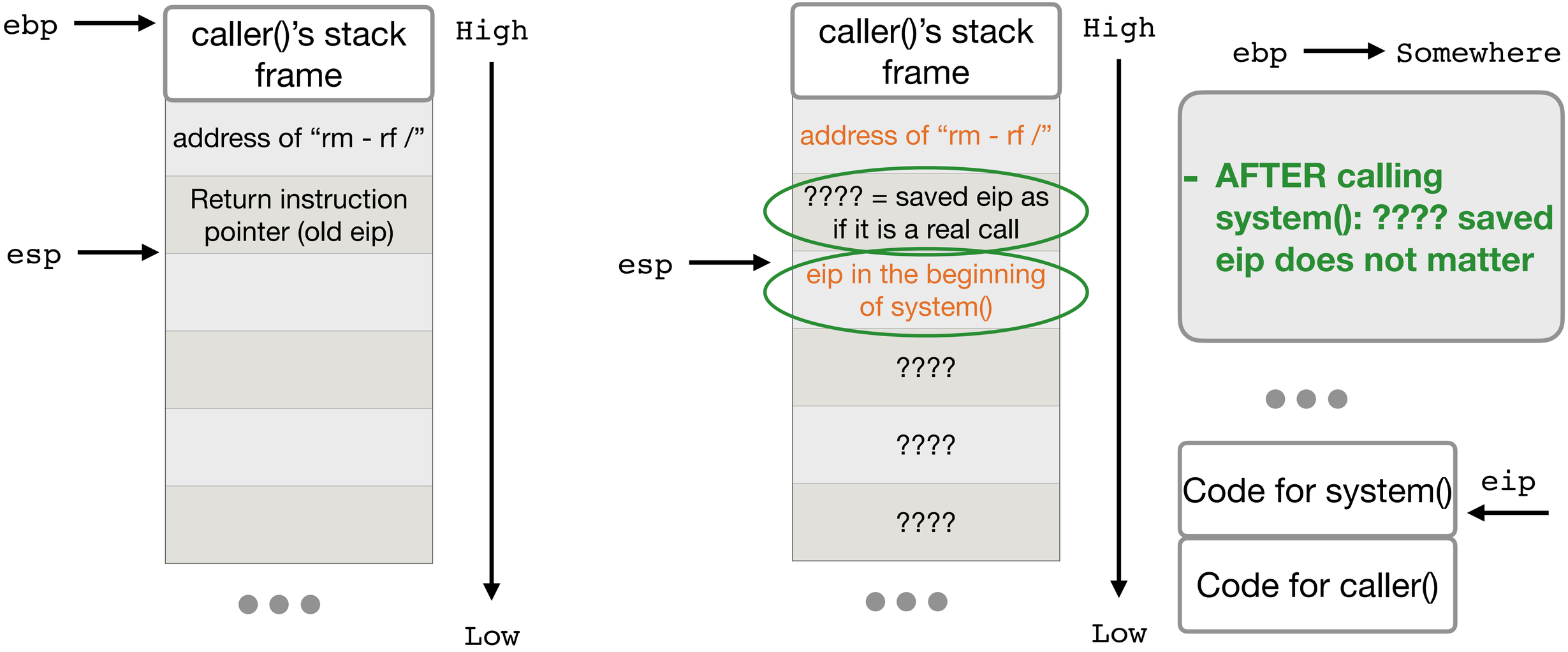
- leave
  - mov %ebp %esp
  - pop %ebp
- ret: pop %eip



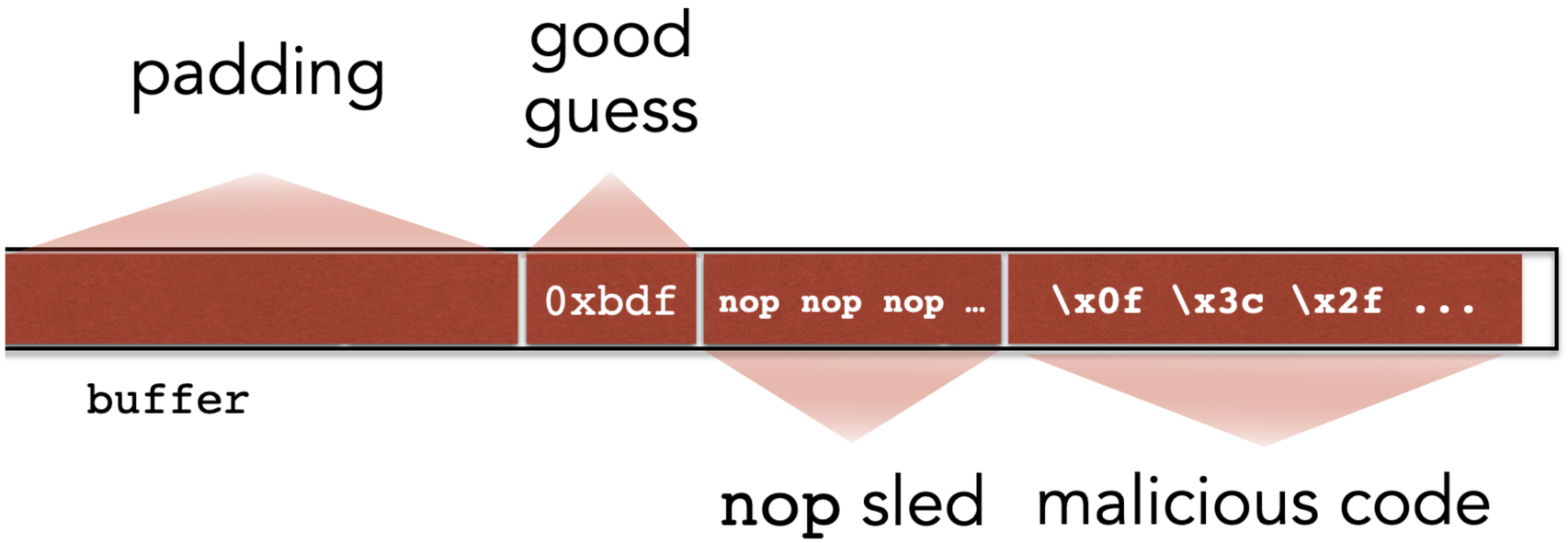
Exercise: Go through leave return



Exercise: Go through leave return



NOP Slide / NOP Sled



- Putting the shell code in the end of the payload buffer can maximize the number of NOPs
- Good guess of somewhere in NOP: jumping anywhere inside the NOP will make the attack successful.
- This improves our chances of guessing by a factor of # of NOPs.

Exceptions to the Same-Origin Policy

- Exception: JavaScript runs with the origin of the page that loads it

How to exploit this?

- Attacker goal: access information on the legitimate website
- Idea: the attacker adds malicious JS to a legitimate website
- JS will run with the origin of the legitimate website

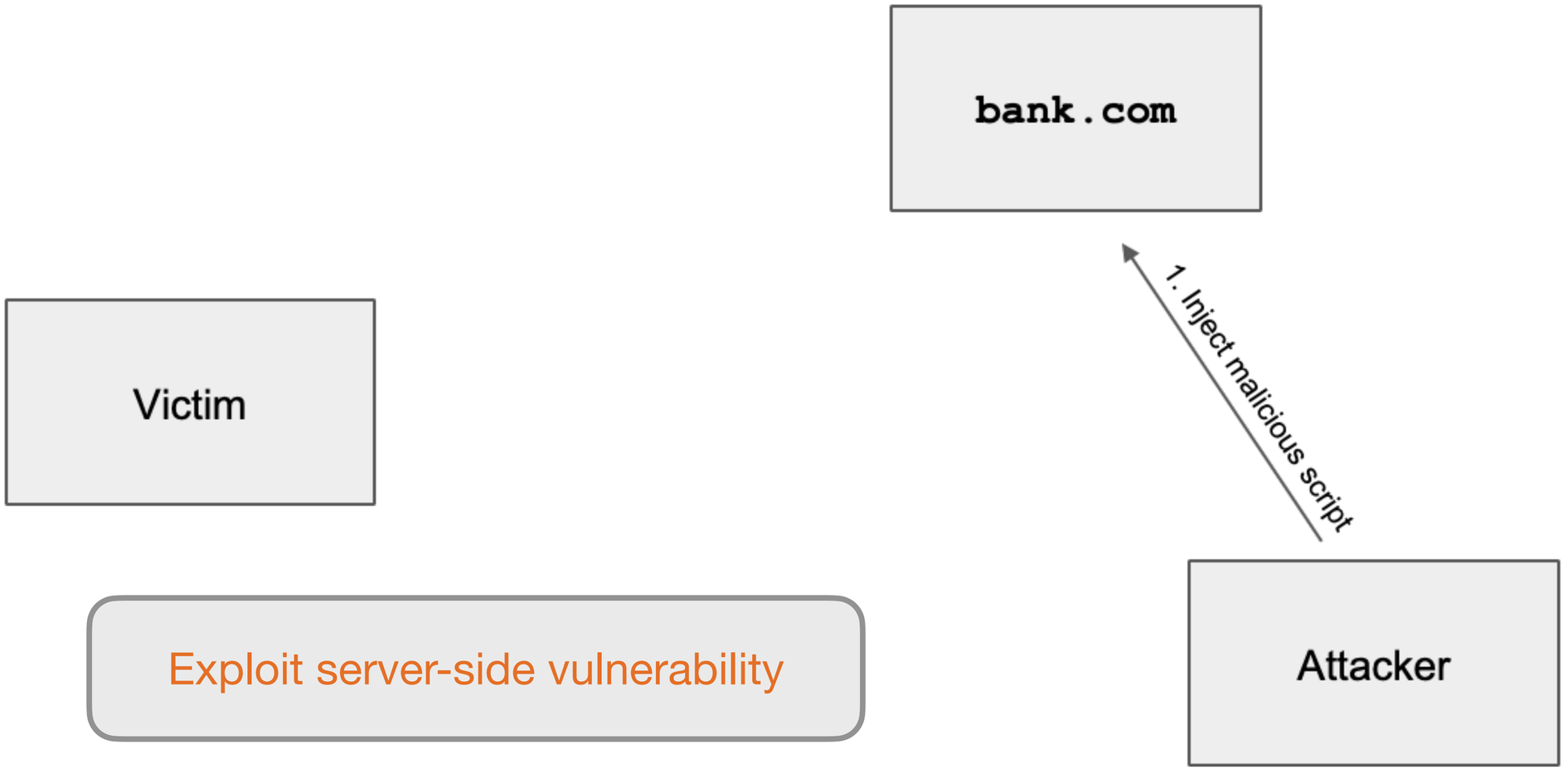
Cross-Site Scripting (XSS)

- **Cross-site scripting (XSS)**: Injecting JavaScript into websites that are viewed by other users
  - Cross-site scripting subverts the same-origin policy
- Two main types of XSS
  - Stored XSS
  - Reflected XSS

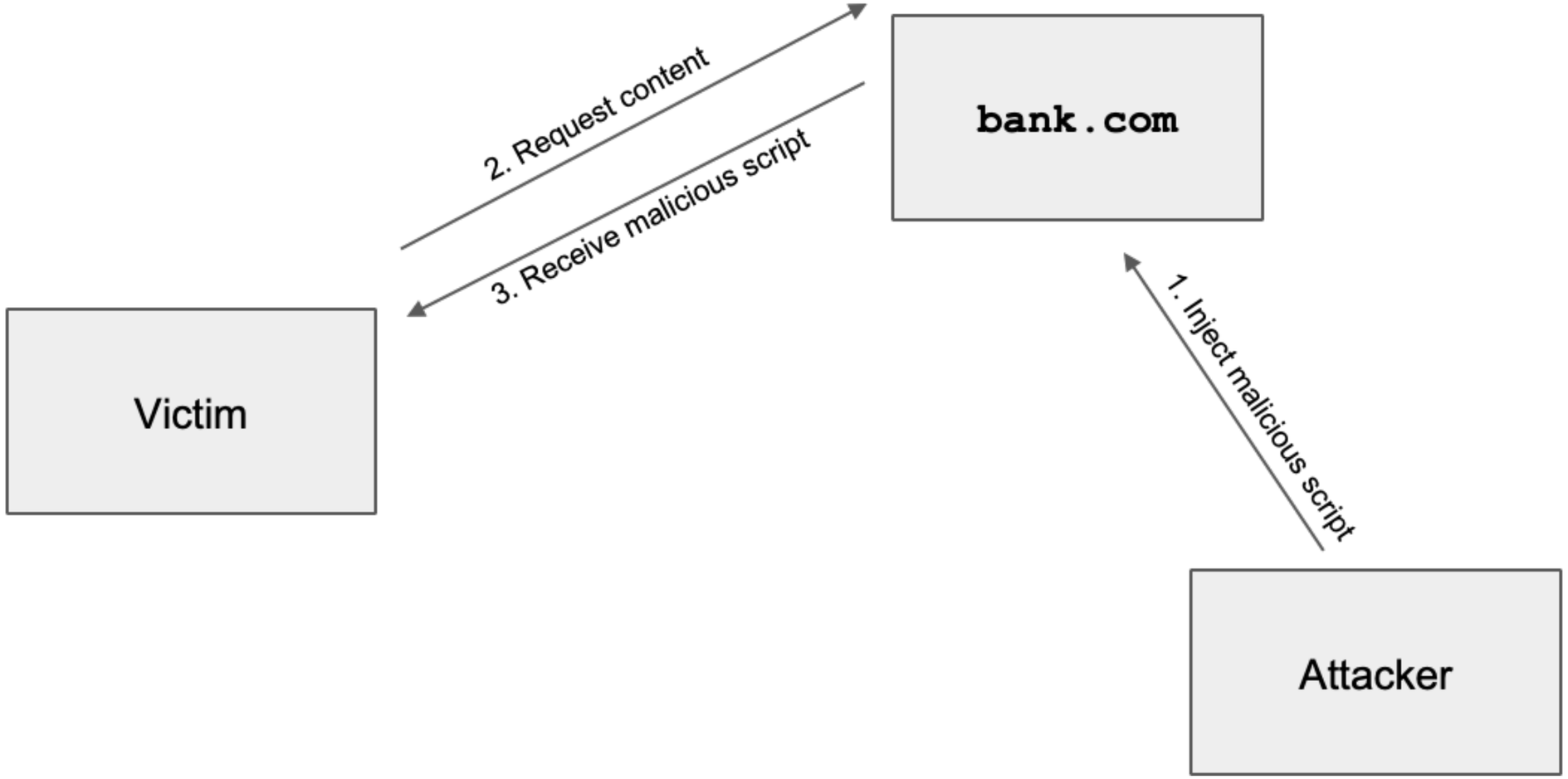
Stored XSS

- **Stored XSS (persistent XSS)**: The attacker's JavaScript is stored on the legitimate server and sent to browsers
- Classic example: Facebook pages
  - An attacker puts some JavaScript on their Facebook page
  - Anybody who loads the attacker's page will see JavaScript (with the origin of Facebook)
- Stored XSS requires the victim to load the page with injected JavaScript
- Remember: Stored XSS is a **server-side vulnerability**!

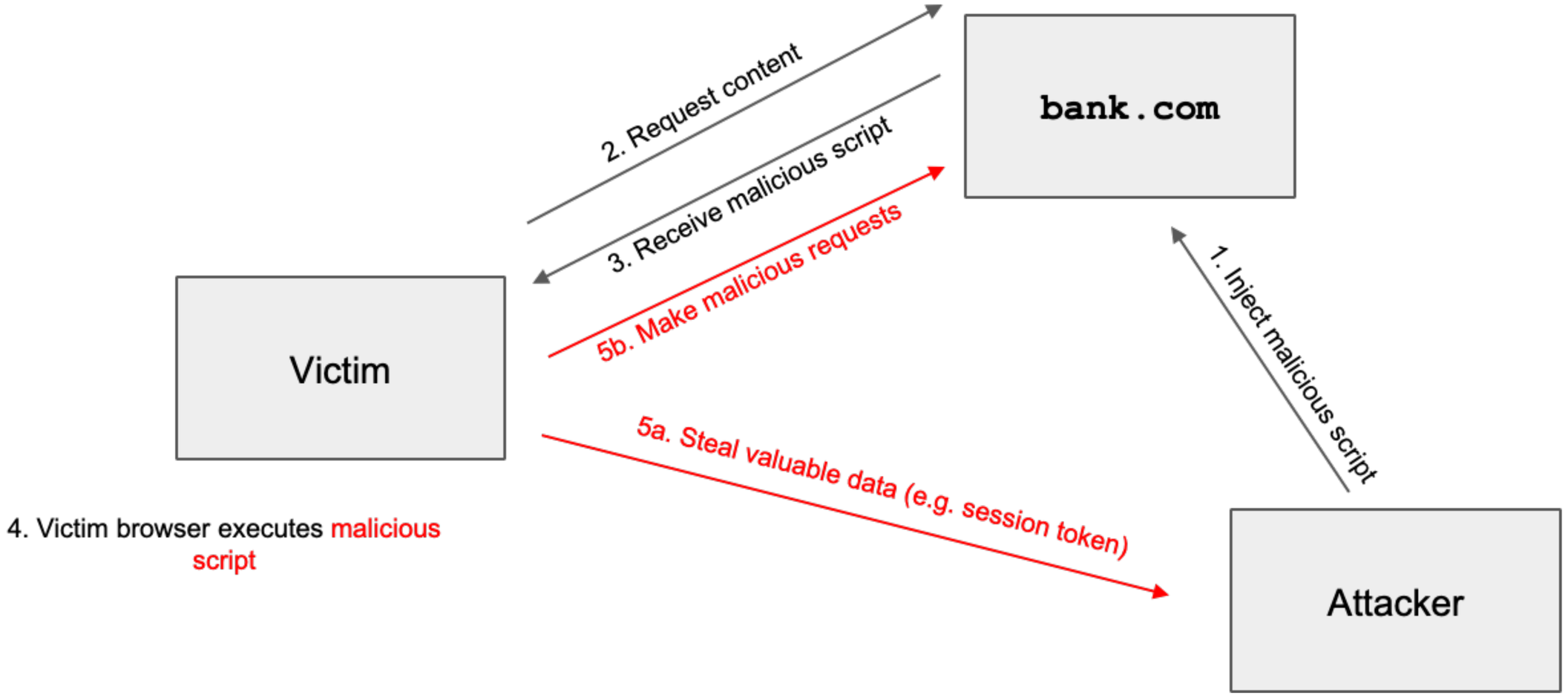
Stored XSS


bank.com
1. Inject malicious script
Victim
Exploit server-side vulnerability
Attacker

Stored XSS


bank.com
2. Request content
3. Receive malicious script
Victim
1. Inject malicious script
Attacker

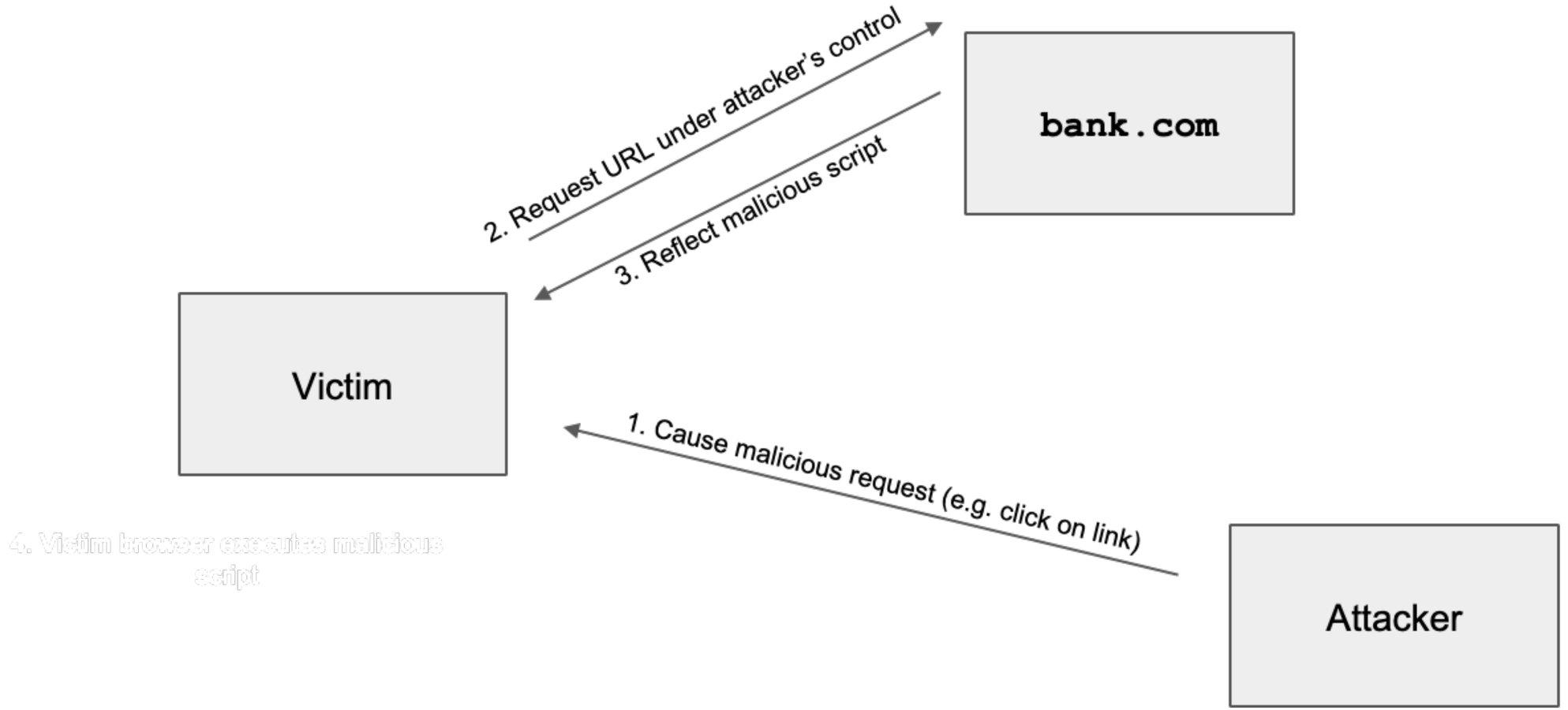
Stored XSS


bank.com
2. Request content
3. Receive malicious script
5b. Make malicious requests
1. Inject malicious script
Victim
5a. Steal valuable data (e.g. session token)
Attacker
4. Victim browser executes malicious script

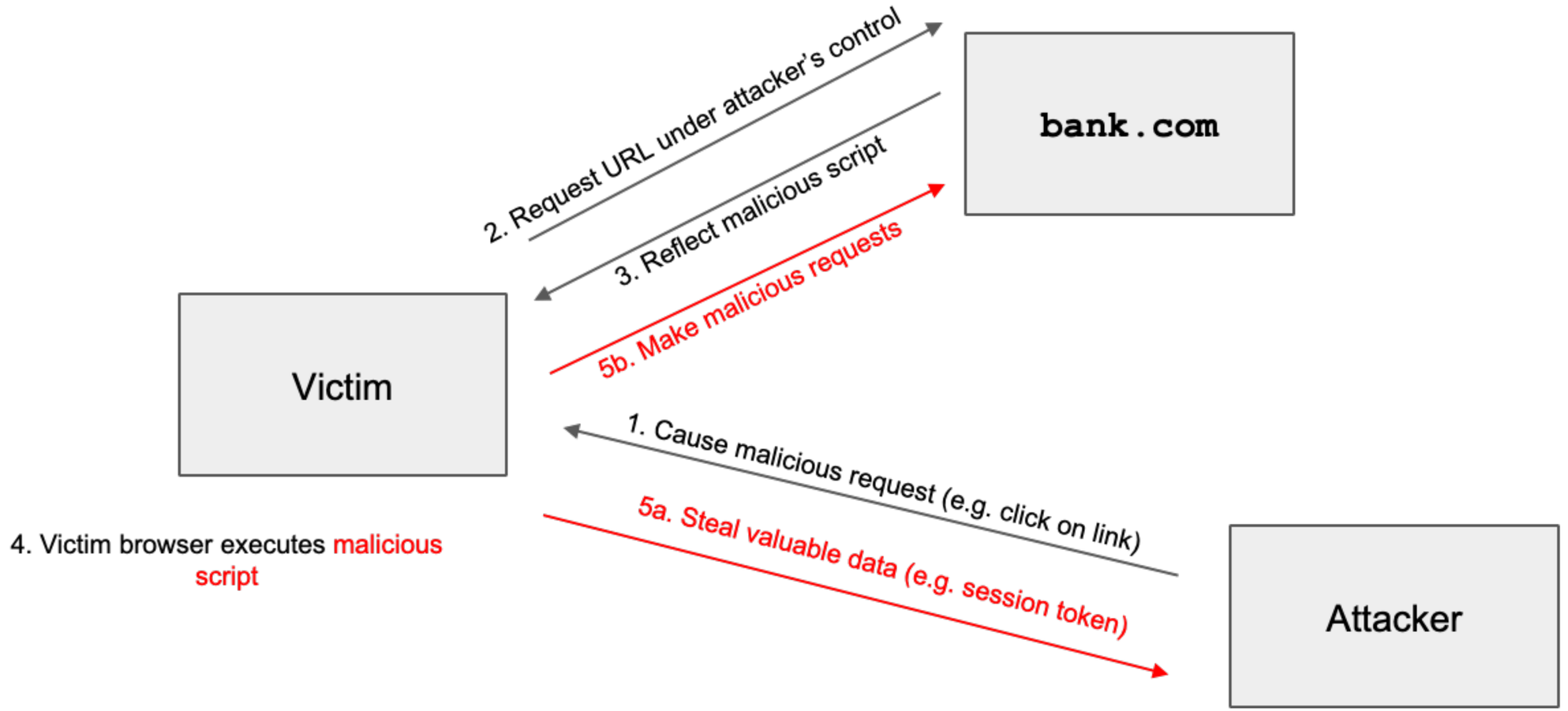
Reflected XSS

- **Reflected XSS:** The attacker causes the victim to input JavaScript into a request, and the content is **reflected (copied)** in the response from the server
  - Classic example: Search
  - If you make a request to http://google.com/search?q=bot, the response will say “10,000 results for bot”
  - If you make a request to http://google.com/search?q=<script>alert(1)</script>, the response will say “10,000 results for <script>alert(1)</script>”
- Reflected XSS requires the victim to make a request with injected JavaScript

Reflected XSS



Reflected XSS



Reflected XSS: Making a Request

- How do we force the victim to make a request to the legitimate website with injected JavaScript?
  - Trick the victim into visiting the attacker's website, and include an embedded iframe that makes the request
    - Can make the iframe very small (1 pixel x 1 pixel), so the victim doesn't notice it:

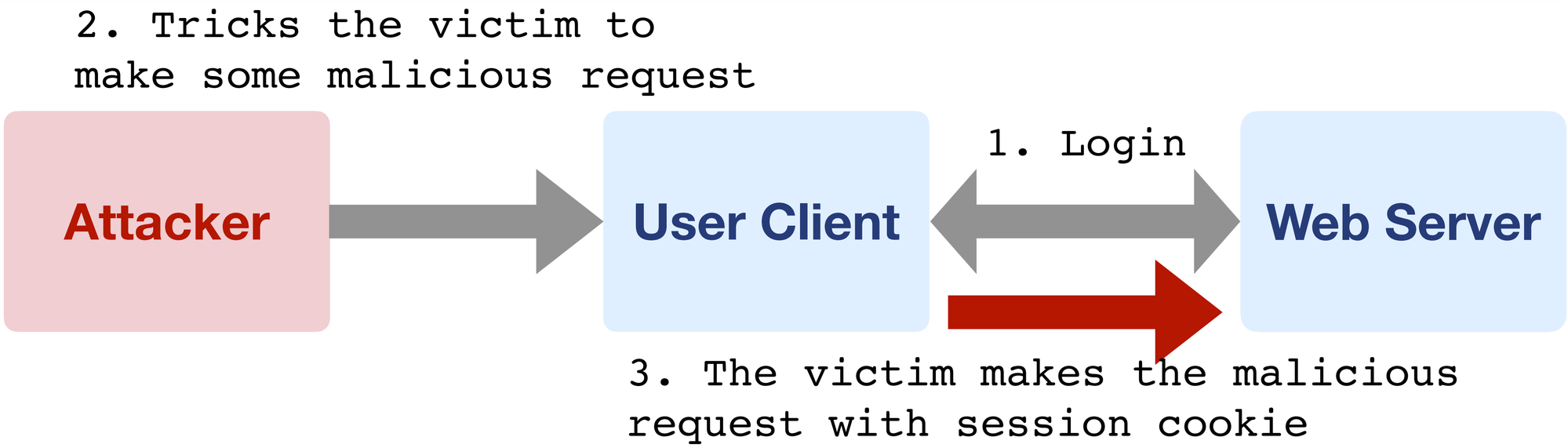
      ```
      <iframe height=1 width=1 src="http://google.com/search?
      q=<script>alert(1)</script>">
      ```
  - clicking a link (e.g. posting on social media, sending a text, etc.)
  - visiting the attacker's website, which redirects to the reflected XSS link
  - …

Reflected XSS is not CSRF

- Reflected XSS and CSRF both require the victim to make a request to a link
- Reflected XSS: An HTTP response contains maliciously inserted JavaScript, executed on the client side
- CSRF: A malicious HTTP request is made (containing the user's cookies), executing an effect on the server side

Steps of a CSRF Attack

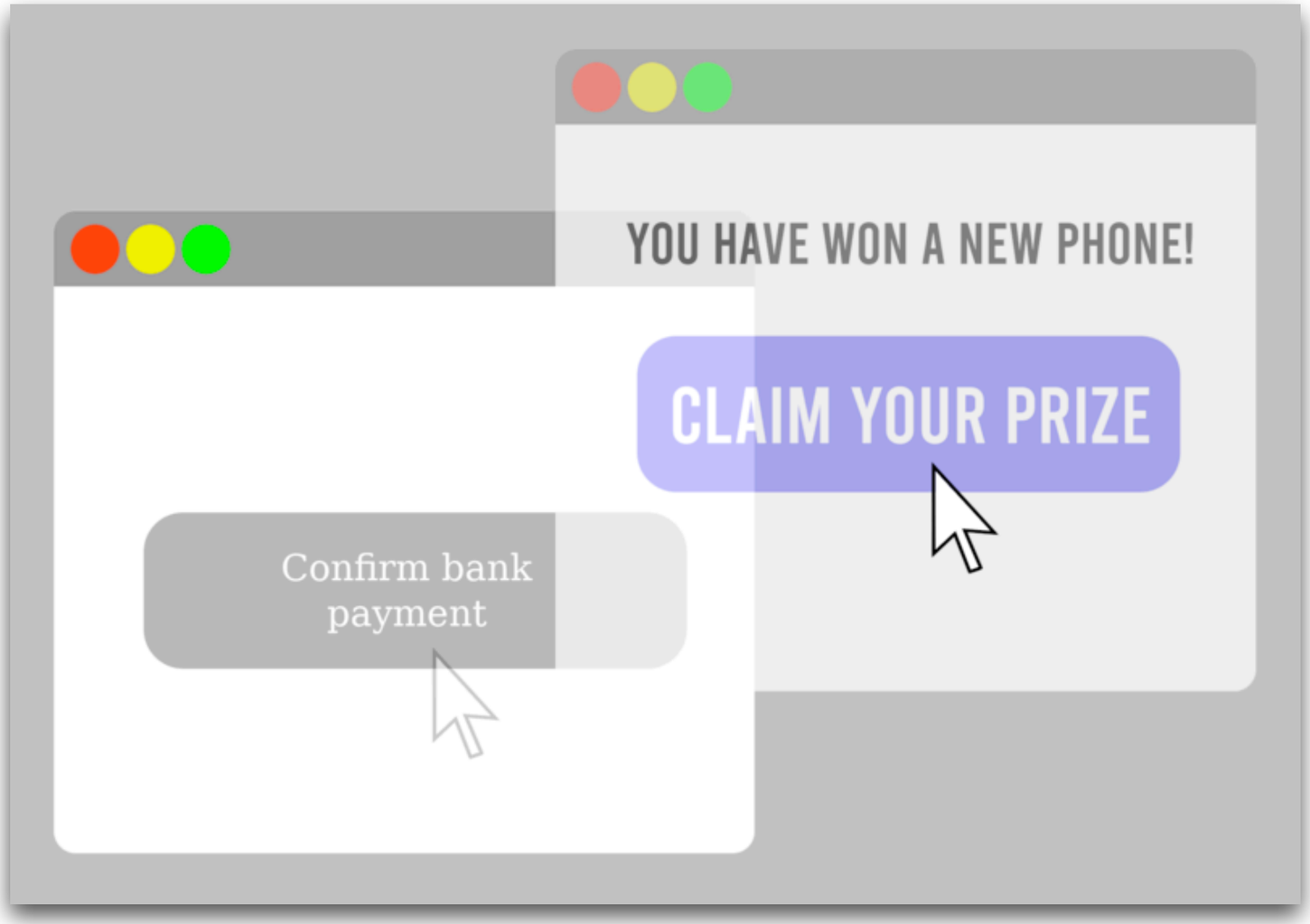
1. User authenticates to the server, receives a **cookie** with a valid **session token**
2. Attacker **tricks** the victim into making a malicious request to the server
3. The victim **makes the malicious request**, attaching the cookie, server accepts it



Clickjacking: Download Buttons

- Which is the real download button?
- What if the user clicks the wrong one?

Invisible iframe Variant #1



- Frame the legitimate site invisibly, over visible, enticing content
- Victims think they are clicking on the enticing site, but they click on the legitimate site, e.g., pay the attacker's account