DOCUMENTAZIONE LABORATORIO 3
FONDAMENTI DI CYBERSECURITY

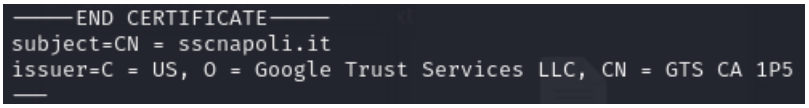
Manuel Castiglia
Thomas Westerman

ESERCIZIO 1

Come prima cosa, una volta aver capito quali fossero e come funzionassero i vari comandi, ci siamo messi alla ricerca di un sito che utilizzasse Let's Encrypt.
Al primo tentativo siamo riusciti a trovare un sito che non utilizzasse Let's Encrypt e come primo tentativo abbiamo provato con il sito della SSC Napoli (https://sscnapoli.it/) tramite il comando

'openssl s_client -connect www. sscnapoli.it:443 2>/dev/null | openssl x509 -noout -text'

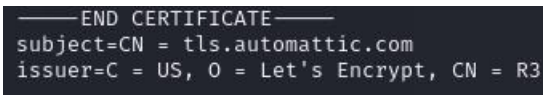
il quale ha evidenziato che l'ente certificatore del sito è Google Trust Services



Discorso diverso invece per l'altro caso, in quanto sono serviti un paio di tentativi prima di riuscire a trovare il primo sito web che utilizzasse Let's Encrypt, ma alla fine abbiamo trovato il sito della Polisportiva Michelangelo ASD (https://polisportivamichelangelo.com/) tramite il comando

'openssl s_client -connect www.polisportivamichelangelo.com:443 2>/dev/null | openssl x509 -noout -text'

che soddisfava la nostra ricerca



Il passo successivo è stato pensare a una lista di 10 siti web e controllare effettivamente se utilizzano Let's Encrypt oppure no. I risultati della nostra ricerca sono stati:

1) MEDIOLANUM (https://www.bancamediolanum.it/)
   - Utilizza come ente certificatore Digicert (https://www.digicert.com/it)
2) RALLYLINK (https://www.rallylink.it/home_it.htm)
   - Utilizza come ente certificatore Digicert (https://www.digicert.com/it)
3) RSE ITALIA (https://www.rseitalia.it/)
   - Utilizza come ente certificatore Let's Encrypt (https://letsencrypt.org/it/)
4) USL TOSCANA SUD EST (https://www.uslsudest.toscana.it/)
   - Utilizza come ente certificatore Global Sign (https://www.globalsign.com/en)
5) RUZZA OROLOGI (https://ruzzaorologi.com/)
   - Utilizza come ente certificatore CloudFlare (https://www.cloudflare.com/it)
6) CORA SPA (https://www.coraitaly.com/)
   - Utilizza come ente certificatore Global Sign (https://www.globalsign.com/en)
7) COMUNE DI CAPRESE MICHELANGELO()
   - Utilizza come ente certificatore Let's Encrypt (https://letsencrypt.org/it/)
8) BREMBO (https://www.brembo.com/it)
   - Utilizza come ente certificatore Digicert (https://www.digicert.com/it)

9) M20 (https://www.m2o.it/)
    - Utilizza come ente certificatore Amazon (https://aws.amazon.com/)
10) TICKET SMS (https://www.ticketsms.it/it/)
    - Utilizza come ente certificatore Digicert (https://www.digicert.com/it)

Come si può osservare dalla lista, è stato più semplice trovare siti web che non utilizzassero Let's Encrypt piuttosto che siti che lo utilizzano

Il passo successivo è stato quello di prendere i siti di SSC Napoli e Polisportiva Michelangelo e vedere quali fossero le versioni di TLS che supportassero. Per fare questo controllo abbiamo usato il seguente comando

'openssl s_client -connect www.sscnapoli.it:443 -servername www.sscnapoli.it -tls1_3' per il sito SSC Napoli

e

'openssl s_client -connect www.polisportivamichelangelo.com:443 -servername www.polisportivamichelangelo.com -tls1_3' per il sito Polisportiva Michelangelo

andando di volta in volta a cambiare l'ultimo parametro, per verificare le varie versioni di TLS

Qui riportiamo i risultati:

- Polisportiva Michelangelo
    - Non supporta la versione TLS 1.0
    - Non supporta la versione TLS 1.1
    - Supporta la versione TLS 1.2
    - Supporta le versioni superiori a TLS 1.2
- SSC Napoli
    - Supporta la versione TLS 1.0
    - Supporta la versione TLS 1.1
    - Supporta la versione TLS 1.2
    - Supporta le versioni superiori a TLS 1.2

Come ultima cosa, siamo andati ad analizzare uno dei siti sopra citati, sul sito SSLabs. Abbiamo scelto il sito Polisportiva Michelangelo e il risultato è stato il seguente:

- Il suo punteggio generale è A+
- Supporta tutte le versioni di TLS superiori alla 1.2 (compresa)
- Utilizza come ente certificatore Let's Encrypt
- Il sito utilizza la tecnologia "Certificate Transparency" che garantisce la trasparenza e la verificabilità dei certificati SSL / TLS emessi per il dominio
- Ha HSTS abilitato, con max-age = 31536000
    - HSTS è un meccanismo che protegge i siti web da attacchi Man In The Middle, facendo comunicare il sito solo tramite HTTPS anziché HTTP
    - Max-age sta a significare che il browser dell'utente che visita la pagina forzerà una connessione HTTPS verso il sito, per un anno dal momento in cui è stato visitato il sito
- Il sito non ha la misura di sicurezza HSTS preloading che permette di forzare la connessione HTTPS anche se l'utente inserisce http anziché https
- Ha la Downgrade Attack Prevenction
    - Sono misure di sicurezza implementate per proteggere una connessione TLS da possibili attacchi che mirano a ridurre il livello di sicurezza di una connessione TLS

- Supporta la Forward Secrecy con la maggior parte dei Browser
    - Significa che il sito protegge le comunicazioni online da potenziali attacchi di decifratura nel caso di compromissione futura della chiave privata del server
- Il sito non supporta la versione 49 di Chrome su Windows XP SP3 in quanto molto vecchia
- La suite di cifratura che utilizza AES_128_CBC_SHA e la suite di cifratura che utilizza AES_256_CBC_SHA sono considerate deboli in quanto utilizzano SHA-1 anziché algoritmi di hash più recenti come sha-256

ESERCIZIO 2

Una volta scaricati i file da virtuale e aver capito quali fossero i comandi da utilizzare siamo passati a decifrare il primo file. Il primo passo è stato utilizzare il comando

'hashcat -a 0 '9e02fdbbc6df842939b75a98e92e98317d29046c' /usr/share/wordlists/rockyou.txt'

per trovare l' hash mode da utilizzare, in quanto questo comando restituisce una serie di valori da passare al parametro -m per poter passare alla decifratura dell'hash.

Siamo passati poi a scaricare il ruleset InsidePro-PasswordsPro.rule da mettere come parametro al comando hashcat, e dopo un paio di tentativi abbiamo trovato l'hash mode corretto per la decifratura, che era -m 100, che ha fornito come password 'spaghettibolognase1987'

```
9e02fdbbc6df842939b75a98e92e98317d29046c:spaghettibolognase1987

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 100 (SHA1)
Hash.Target......: 9e02fdbbc6df842939b75a98e92e98317d29046c
Time.Started.....: Thu May  2 14:01:37 2024 (25 mins, 8 secs)
Time.Estimated ...: Thu May  2 14:26:45 2024 (0 secs)
Kernel.Feature ...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Mod........: Rules (/home/kali/Downloads/InsidePro-PasswordsPro.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  7767.4 kH/s (6.09ms) @ Accel:256 Loops:64 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 11649592320/46389741090 (25.11%)
Rejected.........: 0/11649592320 (0.00%)
Restore.Point....: 3601920/14344385 (25.11%)
Restore.Sub.#1 ...: Salt:0 Amplifier:1216-1280 Iteration:0-64
Candidate.Engine.: Device Generator
Candidates.#1....: spain1111961 → spaceorb12024
Hardware.Mon.#1..: Util: 76%

Started: Thu May  2 14:01:37 2024
Stopped: Thu May  2 14:26:47 2024
```

Una volta trovata la password, siamo passati a decifrare il primo file tramite questo comando

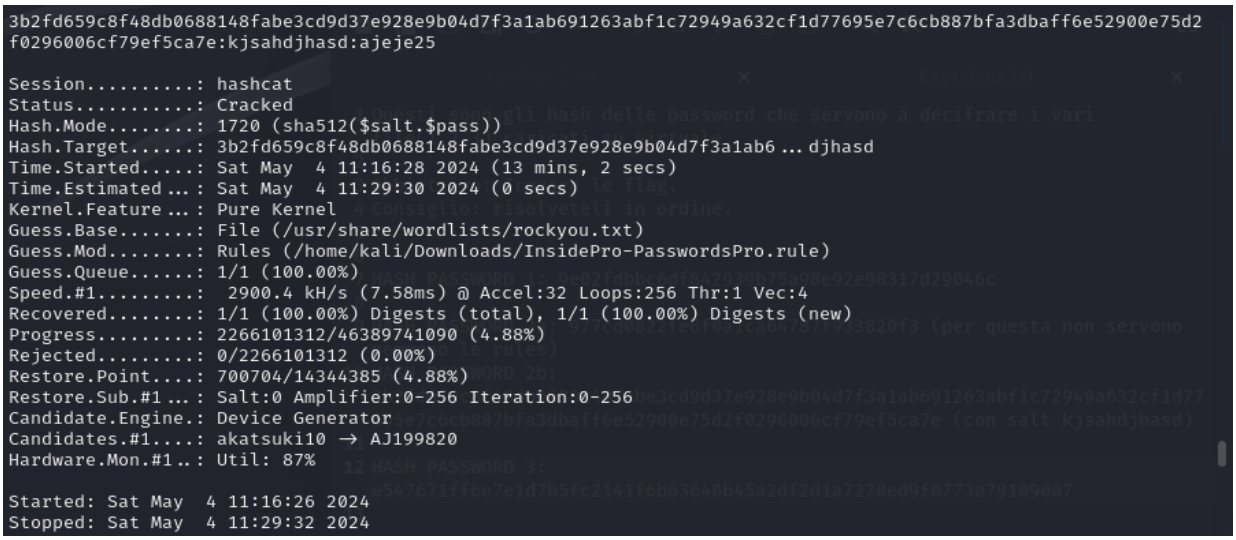
'openssl aes-256-ecb -d -in file1.aes-256-ecb.txt -out file1.aes-256-ecb-decrypted.txt'

e una volta inserita la password, ci ha generato un file di output che conteneva il primo flag: !bella_vez09567!

Completato il primo file siamo passati al secondo, che si componeva di due hash da decifrare.

Abbiamo usato lo stesso approccio del primo file per entrambe le stringhe, aggiungendo semplicemente il sale infondo all'hash, ossia utilizzando il comando senza il parametro -m per trovare la lista di hash da poter utilizzare. Abbiamo trovato il valore di -m corretto e abbiamo trovato la prima password, ossia 'ajeje'

Successivamente abbiamo replicato gli stessi passaggi per andare a decifrare il secondo hash, trovando come hash mode corretto da usare il valore 1720, trovando la password 'ajeje25'



Una volta trovate le due password, siamo passati a decifrare il secondo file. A differenza del precedente, abbiamo dovuto utilizzare due volte il comando

'openssl aes-256-cbc -d -in file2.aes-256-cbc.txt -out file2.aes-256-ecb-decrypted1.txt'

'openssl aes-256-cbc -d -in file2.aes-256-ecb-decrypted1.txt -out file2.aes-256-ecb-decrypted2.txt'

la prima volta mettendo come file di input il file scaricato da virtuale e utilizzando la password 'ajeje', mentre la seconda volta abbiamo messo come file di input il file output generato prima e abbiamo utilizzato la password ajeje25. Fatta questa procedura è stato generato un secondo file di output contente il secondo flag 'kjahgsdjkhkjashd0' e dei consigli per trovare la password del file 3.

Per decifrare il file tre, abbiamo utilizzato i suggerimenti che abbiamo ottenuto dal file precedente. Siamo andati a creare un ruleset ‘regole.rule’ dove abbiamo inserito tutte le possibili combinazioni di numeri e punti che si potevano trovare al termine delle parole, e abbiamo inoltre creato una wordlist ‘regole.txt’ dentro la quale abbiamo messo le possibili parole che, combinate con le regole definite prima, avrebbero formato la password.

La struttura del ruleset è la seguente:

```
$2 $9 $9 $.
$3 $9 $9 $.
$4 $9 $9 $.
$5 $9 $9 $.
$6 $9 $9 $.
$7 $9 $9 $.
$8 $9 $9 $.
$9 $9 $9 $.
```

```
$0 $0 $0 $!
$1 $0 $0 $!
$2 $0 $0 $!
$3 $0 $0 $!
$4 $0 $0 $!
$5 $0 $0 $!
$6 $0 $0 $!
$7 $0 $0 $!
```

Mentre la struttura del dizionario è questa:

```
1 bella_vez
2 bella_vecchio
3 bolognese
4 dammi_tiro
5 butta_rusco
6
```

Fatto ciò, abbiamo ripercorso lo stesso procedimento applicato in precedenza, andando a trovare la password ‘dammi_tiro579.’ tramite il comando

‘hashcat -m 1400 -a 0 -r /home/kali/Desktop/regole.rule 'e547671ff6e7e1d7b5fc2141f6b63648b45a2df2d1a7270ed9f0773a79109007' /home/kali/Desktop/regole.txt’

```
e547671ff6e7e1d7b5fc2141f6b63648b45a2df2d1a7270ed9f0773a79109007:dammi_tiro579.

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: e547671ff6e7e1d7b5fc2141f6b63648b45a2df2d1a7270ed9f ... 109007
Time.Started.....: Mon May  6 14:50:37 2024 (0 secs)
Time.Estimated ...: Mon May  6 14:50:37 2024 (0 secs)
Kernel.Feature ...: Pure Kernel
Guess.Base.......: File (/home/kali/Desktop/regole.txt)
Guess.Mod........: Rules (/home/kali/Desktop/regole.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   820.2 kH/s (1.08ms) @ Accel:16 Loops:256 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 5120/10000 (51.20%)
Rejected.........: 0/5120 (0.00%)
Restore.Point....: 0/5 (0.00%)
Restore.Sub.#1 ...: Salt:0 Amplifier:768-1024 Iteration:0-256
Candidate.Engine.: Device Generator
Candidates.#1....: bella_vez867. → butta_rusco320!
Hardware.Mon.#1..: Util: 35%

Started: Mon May  6 14:50:36 2024
Stopped: Mon May  6 14:50:40 2024
```

Una volta trovata la password, siamo passati a decifrare il file 3 tramite il comando

‘openssl aes-256-cbc -d -in /home/kali/Desktop/file3.enc.txt -out /home/kali/Desktop/file3.dec.txt’

che ci ha restituito un file output contente il terzo ed ultimo flag ‘congrats!23042024’ oltre il messaggio ‘Congratulazioni! Hai trovato la terza flag!’