



Università di Catania

Progetto Internet Security A.A 22/23 A02:2021-Cryptographic Failures

Realizzato dallo studente:

Vincenzo Barba

1000002746

Docente di riferimento:

prof. Giampaolo Bella



Sommario

| | |
|---|-----------|
| 1. Introduzione | 3 |
| 2. Attacco 1: dati a riposo | 5 |
| 2.1. Ipotesi | 5 |
| 2.2. Premesse tecniche | 5 |
| 2.3. Demo attacco | 8 |
| 3. Attacco 2: dati in transito | 13 |
| 3.1. Ipotesi | 13 |
| 3.2. Premesse tecniche | 14 |
| 3.3. Demo Attacco | 16 |
| 3.4. Possibili difese | 21 |
| 4. Fonti | 22 |

1. Introduzione

Nel campo della cybersecurity, il problema dei fallimenti crittografici è uno dei più rilevanti, soprattutto negli ultimi anni; infatti, secondo l'*OWASP Foundation*, è oggi al secondo posto assoluto tra le varie vulnerabilità nel mondo informatico.

Mentre fino a un decennio fa circa la crittografia era ancora diffusa poco a causa degli elevati costi e della mancanza di comprensione delle sue funzionalità e vantaggi, oggi invece si sta cominciando a pensare di criptare tutto ciò che riguarda un prodotto digitale *by default*. Questo sì per una maggiore accessibilità in termini di costo, ma soprattutto perché ricercatori ed esperti nel campo crittografico, tramite i loro approfonditi studi su quello che è ormai un ramo molto vasto dell'informatica, hanno convinto il mondo scientifico dell'importanza e la necessità della crittografia. Inoltre, con l'avvento delle leggi sulla privacy dei dati, ad esempio il *GDPR*, la protezione di ogni genere di dati digitali è ormai diventata un diktat aziendale.

Un sistema crittografico è genericamente composto da una coppia di algoritmi e una chiave. L'algoritmo di *encryption* serve per rendere indeducibile il dato in input, mentre quello di *decryption* serve per trasformare l'input da indeducibile in deducibile. Entrambi, per essere effettuati con successo, necessitano della chiave (che può anche essere diversa per un'operazione rispetto l'altra). L'obiettivo della crittografia è quello di preservare i tre pilastri della sicurezza informatica su un certo dato: segretezza, autenticazione e integrità.

A causa però dell'evoluzione tecnologica e del continuo scontro fra difesa e attacco, il cui unico limite è la disponibilità economica e quindi di risorse, ad oggi molti sistemi crittografici sono sconsigliati e in disuso, perché contenenti vulnerabilità ormai conosciute e sfruttabili da eventuali attaccanti. È fondamentale inoltre ricordare che un sistema crittografico non sempre necessita o si pone l'obiettivo di garantire tutte e tre le proprietà fondamentali della sicurezza; in ogni caso per ottenere le proprietà volute, necessita che le assunzioni fatte a priori, che differiscono in base all'algoritmo, affinché esso funzioni debbano valere, tra cui su tutte la segretezza, confidenzialità della/e chiave/i. Però, per rompere un sistema crittografico non si necessita per forza della conoscenza della chiave, per via della crittoanalisi: essa è una scienza ormai collaudata, il cui scopo è quello di eludere la sicurezza di algoritmi crittografici tramite canali laterali, bypassando la sicurezza della chiave.

Secondo l'*OWASP*, i fallimenti crittografici comportano l'esposizione di dati sia a riposo, che in transito; più nel dettaglio, i principali problemi dovuti alla mancanza o l'inadeguata crittografia sono i seguenti:

- Utilizzo di algoritmi o protocolli crittografici vecchi e obsoleti, che trasmettono dati in chiaro tra cui HTTP, SMTP, FTP, o attaccabili facilmente tramite *brute-force*, come DES.
- Salvataggio di password con funzioni hash deprecated, tra cui soprattutto MD5 e SHA1, o anche con salt mancante o inadeguato;
- Dati sensibili, tra cui soprattutto password o anche chiavi crittografiche, codificati in chiaro nel codice sorgente (secondo la *CWE-259: Use of Hard-coded Password*);
- Mancanza di rotazioni periodiche delle chiavi, utilizzo di chiavi deboli o riutilizzate;
- Problemi durante la validazione dei certificati della *chain of trust* di un server.
- Mancanza di entropia nelle funzioni di randomness utilizzate per la creazione di chiavi;
- Utilizzo di crittografia semplice quando invece si necessita l'abbinamento di essa con l'autenticazione.

Proseguendo sono state descritte, come da consegna, due esemplificazioni di attacco sfruttando determinate vulnerabilità legate ai fallimenti crittografici. In particolare, è stato effettuato un attacco alla crittografia su dati a riposo, e uno su dati in transito. Sono stati anche riportati punti di forza e di debolezza degli attacchi, ed eventuali fix in difesa che hanno permesso di rendere vano l'attacco correlato.

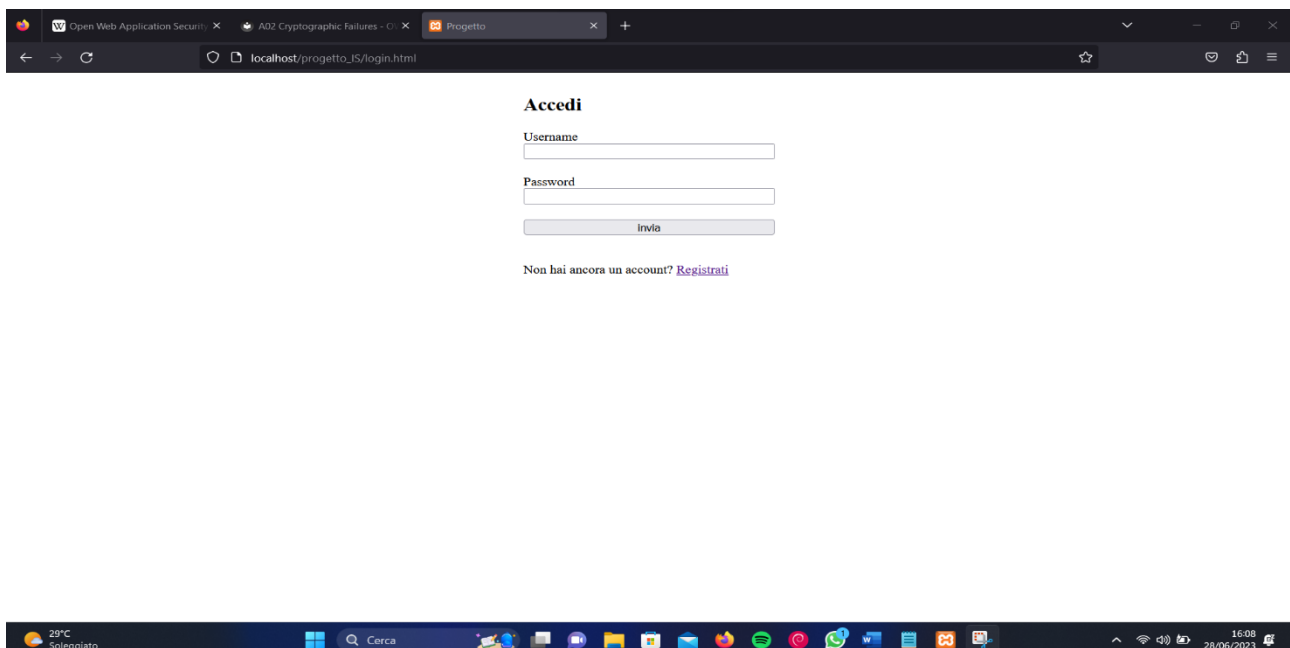
2. Attacco 1: dati a riposo

2.1. Ipotesi

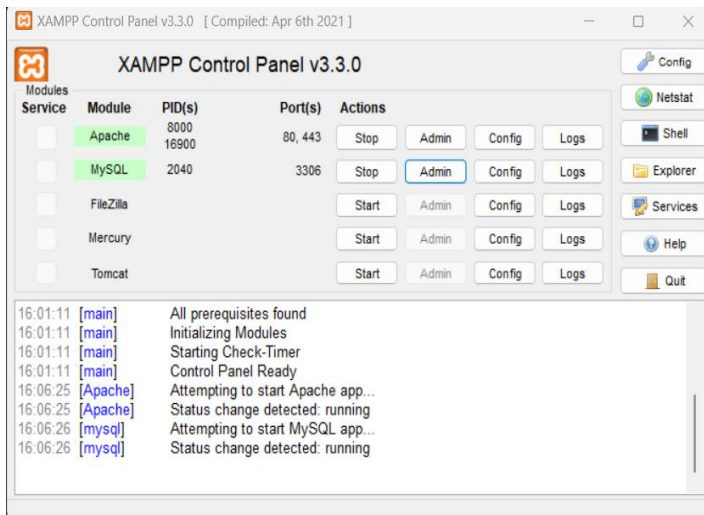
Il caso di ricerca analizzato è stato quello in cui si è supposto di disporre già di un DB di password per l'autenticazione a un web-server, eventualmente in seguito a un attacco *SQL Injection* o una falla nel caricamento dei dati. Per l'esperimento, le password non sono state salvate in chiaro, bensì ne sono stati salvati i corrispondenti valori hash. Nonostante le funzioni hash siano di natura invertibile, e quindi dovrebbe essere impossibile partire da una password hashata e ricondursi alla password originaria, esistono funzioni hash deboli, senza *salt* o con *salt* debole, per cui ciò è possibile, e che sono state appositamente scelte e testate durante questo esperimento. Le vulnerabilità sfruttate fanno riferimento alla *CWE-327: Use of a Broken or Risky Cryptographic Algorithm*. L'obiettivo preposto è stato quello di provare a ricavare le password dai rispettivi hash tramite un *attacco vocabolario*, cioè un file di testo contenente milioni di parole alle quali è stato applicato l'hash e poi confrontato con quello target.

2.2. Premesse tecniche

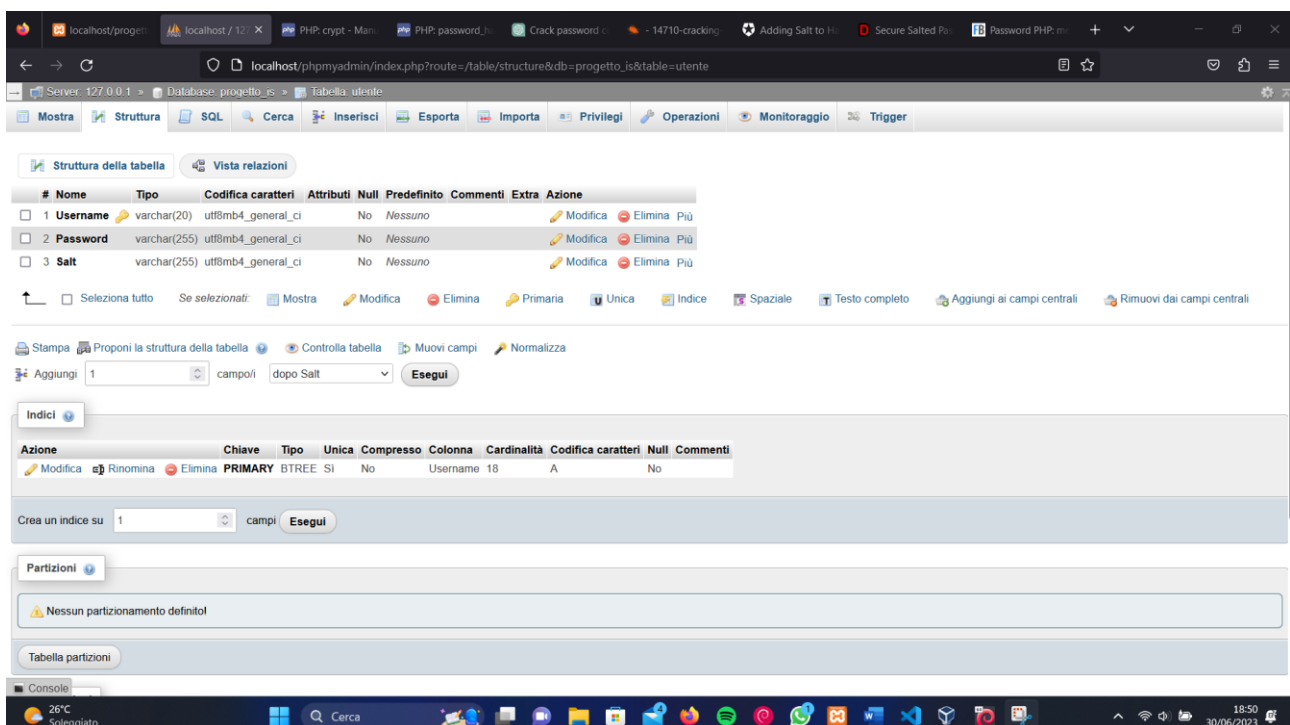
Per la costruzione del laboratorio, al fine di testare al meglio le vulnerabilità sopra elencate e in un ambiente di testing sicuro, onde evitare danni o problemi a siti reali, è stato realizzato un server web locale per la registrazione tramite username e password, e ovviamente per tentare l'autenticazione tramite una schermata di login.



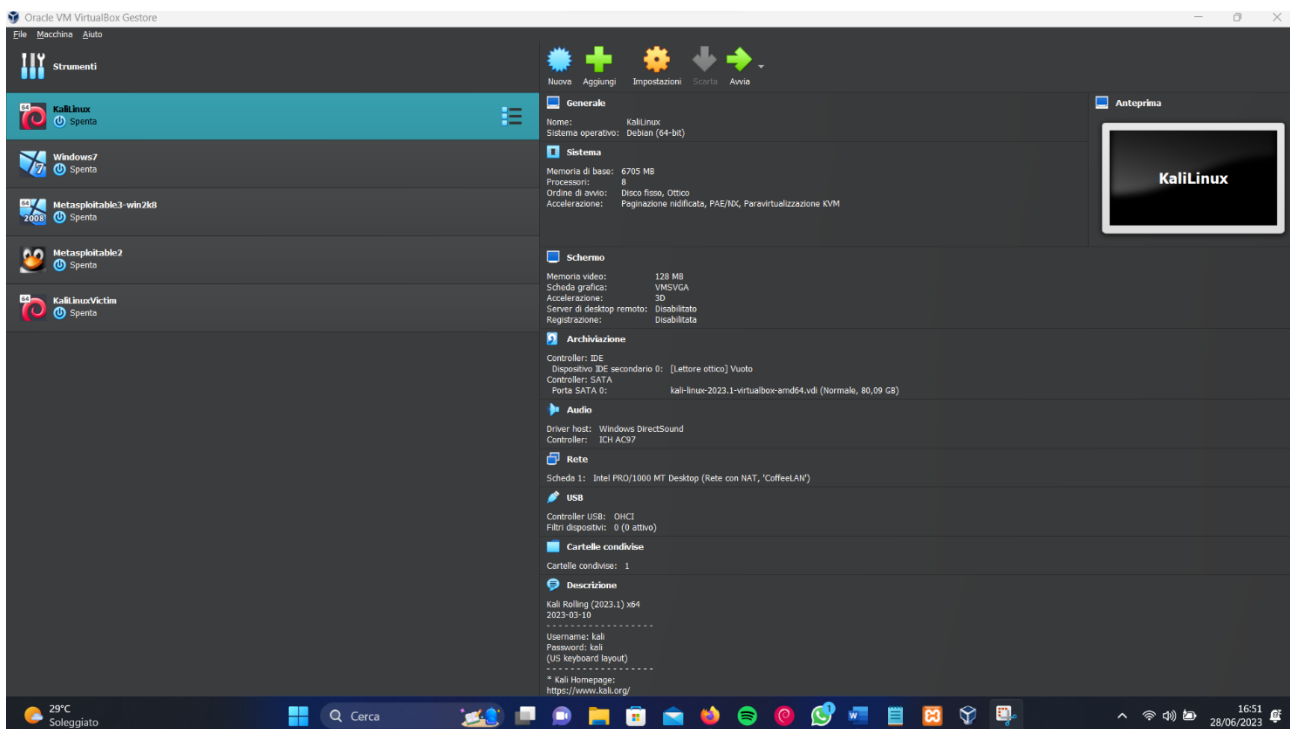
I file html e PHP allegati, realizzati tramite l'editor Visual Studio Code, permettono il corretto layout delle pagine web, il collegamento col DB mySQL e l'invio di query per tentare l'accesso e registrarsi. Il tutto è stato realizzato tramite *XAMPP*, scelto poiché permette una semplice installazione e configurazione di un ambiente di sviluppo web locale. Esso è stato avviato nella macchina host per questioni di efficienza e flessibilità.



All'interno del DB è stata inserita una semplice tabella 'utenti', contenente i campi 'Username', scelto come chiave primaria per evitare che più utenti si registrino con lo stesso, 'Password', con lunghezza massima 255, in modo da poterla salvare dopo l'hashing senza alcun troncamento imprevisto, e infine il campo 'Salt', per salvare il sale nei casi in cui si è optato per l'hashing con sale manuale.



Per tentare il *cracking* delle password in versione hash, si è utilizzata una macchina virtuale Kali Linux, poiché già configurata con tool offensivi. Essa è stata installata nella macchina host (Lenovo IdeaPad 3, Windows 11, CPU Intel Core i5-10210U @1.60GHz 2.11GHz, RAM 8 GB, processore x64) tramite il tool Oracle VM VirtualBox, con le seguenti caratteristiche:



Infine, in Kali Linux è stato utilizzato il tool preinstallato *hashcat*, poiché esso permette di effettuare attacchi offline di *cracking* delle password, disponendo un'ampia scelta tra attacchi dizionario e attacchi brute-force con eventuali mask (pattern specifici); inoltre è in grado di *crackare* una grande quantità di funzioni hash diverse e sfrutta anche la potenza di calcolo delle schede grafiche (GPU) e dei processori multicore per accelerare il processo. È bene sottolineare infatti, che per simulare al meglio attacchi di questo genere, è necessario conferire alla macchina virtuale la maggiore quantità possibile di CPU, e l'accesso alle GPU.

L'hashing delle password, sia per il login che per la registrazione, è stato effettuato nel lato server invece che nel DB, al fine di avere a disposizione una grande varietà di funzioni hash, deboli e forti, e poterle settare manualmente in modo più semplice. Questa scelta è sempre consigliata sia per questioni di flessibilità, che di sicurezza.

2.3. Demo attacco

In primis, è stato riempito il DB di utenti, in particolare un utente 'enzo' con password molto semplice 'enzo', e un utente 'prof' con password molto complessa 'InternetSecurity23!'. Inoltre, essi sono stati registrati tante volte quanti sono gli algoritmi testati in figura (parte del codice PHP della pagina di registrazione):

```
//$options = ['salt' => "salt"];
$options = ['cost' => 19];
$salt = base64_encode(random_bytes(16));
$salted_password = $Password.$salt;

$salted = false;

//$hashed_password = password_hash($Password, PASSWORD_DEFAULT); //bcrypt
//$hashed_password = md5($Password); //cracked
//$hashed_password = md5($salted_password); //cracked, non presente in t
//$hashed_password = sha1($Password); //cracked
//$hashed_password = hash("sha256",$Password); //cracked
//$hashed_password = hash("sha512",$Password); //cracked
//$hashed_password = hash("sha512",$salted_password); //cracked, sia pas
//$hashed_password = crypt($Password); //errore, necessita sale per php
//$hashed_password = crypt($Password,'$1$rasmusle$'); //md5salt, cracked
//$hashed_password = crypt($Password,'$6$rounds=5000$thisisaweak_salt$');
$hashed_password = password_hash($Password, PASSWORD_BCRYPT, $options);
```

Nella figura seguente si possono notare i risultati. È da considerare però che in un DB reale un utente non avrà mai la funzione hash concatenata al proprio username, ma qui è stato fatto per questioni di chiarezza e per velocizzare le operazioni di testing. In

alternativa a ciò, analizzando la lunghezza del *digest* o cercando altre caratteristiche univoche di una funzione hash, ci si potrebbe ricondurre ad essa.

| | | | Username | Password | Salt |
|--------------------------|----------|-------|----------|--------------------|---|
| <input type="checkbox"/> | Modifica | Copia | Elimina | admin | \$2y\$10\$qbvBzXWJ8Ly1Nq3MAVfruoPXmB195tZ0LJVT1Jzq... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzobcrypt | \$2y\$19\$1pyyWz9Q5l55r02g8eChmeULhO6Hqcl5Nv97alAcivY... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzomd5 | 6b5b0dd03c9c85725032ce5f3a0918ae |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzomd5saltauto | \$1\$rasmusle\$dqBvG90iJ9/YuffWibQGS1 |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzomd5saltman | 8d5e633296900151bfcee75e7596d425 XpZsEAYGr7zEO7rBZEhl8A== |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzosha1 | a0e5523a645bb1b2c5fd697cbee31025a8f2b788 |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzosha256 | 605306b83fe54de0ab9373e98b9fd30d0a44da6e57487f1962... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzosha512 | 69c5691c6ab5131f885f33b9c8e569a6e9a6d79070ca7b9e80... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzosha512saltauto | \$6\$rounds=5000\$thisisaweak_salt\$2v0kewDH9osk9OwtLj... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | enzosha512saltman | a0caaf197e9c4cf7507eb45c809b7e0d05301cb8da8c4950e3... /uvmlTV5kFetK7y0mE9ldA== |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profbcrypt | \$2y\$19\$ctOSIN0CRImPRhACbFxdluKEJ9JEdrbRI3sri4uno5T... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profm5 | 0091493dafd4390684e9fc80148a3423 |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profm5saltaut | \$1\$rasmusle\$segWD2LHZAXaMsv3HpkKB/ |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profm5saltman | 6828cc36091d12a818b412cae03daef4 x4SM+3o99Dv1sZWZAnbvJQ== |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profsha1 | 4f0b8211f1c3bd9ab010ad4a6ea9f477d116776c |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profsha256 | ad539d1c1a0dabddb19edfbee49fcbbebbf5b749795f904ce74... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profsha512 | 52e57ff6495be30749a935c38f409b32fe45c40dbd0928d051... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profsha512saltaut | \$6\$rounds=5000\$thisisaweak_salt\$WfBSLNJcWduFx5po... |
| <input type="checkbox"/> | Modifica | Copia | Elimina | profsha512saltman | f37d323ecd05ab8522207d966ecdf16818c9dddb4da1df6589... m7DIV/UBpzZtIAKd1RXOavg== |

In seguito, come detto in precedenza, supponendo di disporre già del DB delle password e, ove usato, del salt manuale in chiaro, si è tentato di *crackare* tutte le password sulla macchina virtuale Kali Linux, tramite il tool *hashcat*. La sintassi utilizzata è stata la seguente:

```
-(kali@kali)-[~]
$ hashcat -a 0 -m 0 hash.txt rockyou.txt --show
```

Il parametro '*a*' serve per selezionare la modalità con la quale si vuole eseguire l'attacco, con possibilità di scelta tra attacco dizionario, brute-force, combinato o mask. Durante ogni test è stato scelto il parametro 0 corrispondente ad un attacco dizionario. In conseguenza dell'attacco scelto la sintassi prevede: il parametro '*m*' tramite il quale si sceglie l'algoritmo di hash con il quale la password è stata codificata (vedere documentazione *hashcat* per tutti gli algoritmi supportati tramite il comando *hashcat -h*); il successivo parametro deve essere l'hash che si vuole provare a invertire, inserendolo direttamente sotto forma di stringa, oppure è possibile anche inserire più valori hash da *crackare* contemporaneamente all'interno di un file di testo (in questo caso *hash.txt*); infine il parametro che permette di selezionare il vocabolario, contenente la lista di parole indiziate ad essere la password.

Durante le prove è stato quindi cambiato opportunamente il parametro '*m*' in funzione dell'algoritmo testato; in *hash.txt* sono stati salvati gli hash da rompere; per la lista delle parole, si è scelto un vocabolario già presente all'interno di Kali Linux e contenente circa 14 milioni di parole, '*rockyou.txt*', ricavabile dal percorso

'usr/share/wordlists/'. Le password ricercate, cioè 'enzo' e 'InternetSecurity23!' sono state opportunamente aggiunte in quanto non presenti. In ogni caso, per la quantità di parole contenute, è un buon modo per testare un reale attacco vocabolario.

I risultati ottenuti sono stati i seguenti (uguali sia con la password semplice che con quella complessa):

- con MD5, sia senza salt che con salt manuale (16 caratteri casuali), è stato possibile ricondursi alla password originaria istantaneamente;
- con SHA1, SHA256, SHA512 senza salt la password è stata trovata in pochissimi secondi, ugualmente per SHA512 con salt manuale (16 caratteri casuali);
- usando MD5 e SHA512 con salt gestito tramite la funzione *crypt(..)* di PHP i risultati sono stati identici. Inoltre, è stato provato il parametro round di *crypt(..)* per SHA512 con un valore di 5000 (esso indica quante volte la funzione hash deve essere applicata, le iterazioni), e in pochi secondi è stato possibile trovare la password;
- con bcrypt tramite la funzione *password_hash(..)*, e il parametro cost pari a 10. Il cost in questo caso è l'esponente da applicare alla base 2, il cui risultato sarà il numero di iterazioni dell'algoritmo di hashing. In questo caso quindi, bcrypt è stato eseguito 1024 volte, ma anche in questo caso è stato possibile *crackare* la password in pochi secondi.

Ecco l'output del tentativo di *crack* della password 'InternetSecurity23!' hashata tramite 5000 iterazioni di SHA512, in un tempo complessivo di 1 secondo:

```
$6$weak_salt$iyGZbxbN35i.7dpIK0xHgIkdkTxw1qYwK9W60XbjRNpDhi5JDc6lTTtQnbTeLddu28yJwIZSuC.Q0m5/t.u7j1:InternetSecurity23!

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target.....: $6$weak_salt$iyGZbxbN35i.7dpIK0xHgIkdkTxw1qYwK9W60X ... t.u7j1
Time.Started.....: Fri Jun 30 16:09:37 2023 (1 sec)
Time.Estimated...: Fri Jun 30 16:09:38 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 795 H/s (12.78ms) @ Accel:512 Loops:128 Thr:1 Vec:2
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 512/14344389 (0.00%)
Rejected.....: 0/512 (0.00%)
Restore.Point....: 0/14344389 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4992-5000
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 → garcia
Hardware.Mon.#1..: Util: 40%

Started: Fri Jun 30 16:09:34 2023
Stopped: Fri Jun 30 16:09:40 2023
```

Ecco l'output del tentativo di *crack* della password 'InternetSecurity23!' hashata con 1024 iterazioni di bcrypt, in un tempo complessivo di 2 secondi:

```
$2y$10$FpCEJZ/cZJYuZyS1gIliYeL7C1w0QkxxtB6vgk6mpMLKEu.F1b4zC:InternetSecurity23!
Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target.....: $2y$10$FpCEJZ/cZJYuZyS1gIliYeL7C1w0QkxxtB6vgk6mpMLK ... F1b4zC
Time.Started.....: Fri Jun 30 16:42:17 2023 (2 secs)
Time.Estimated...: Fri Jun 30 16:42:19 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Usage:1%.....: 56 H/s (7.10ms) @ Accel:8 Loops:8 Thr:1 Vec:1
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 128/14344389 (0.00%)
Rejected.....: 0/128 (0.00%)
Restore.Point...: 64/14344389 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:1016-1024
Candidate.Engine.: Device Generator
Candidates.#1....: hottie → richard
Hardware.Mon.#1..: Util: 51%

Started: Fri Jun 30 16:42:12 2023
Stopped: Fri Jun 30 16:42:20 2023
```

In conclusione, è stato possibile confermare che MD5 e SHA1 su tutti, sono funzioni hash troppo deboli e assolutamente da evitare, in quanto troppo veloci e molto soggetti ad *attacchi vocabolario*, *forza bruta* e *rainbow attack*. Anche SHA256 e SHA 512 si sono rivelati molto rapidi e facili da rompere, nonostante il tentativo di aumentare drasticamente le iterazioni di essi. Infine, anche bcrypt, considerato molto solido, se impostato con un parametro 'cost' non abbastanza alto, risulta abbastanza debole.

L'inserimento del salt, nel caso in cui esso sia stato salvato in chiaro nel DB, e quindi visibile all'attaccante, si è rivelato insignificante. Un'altra cosa da evitare è sicuramente l'utilizzo di salt costante, poiché comporterebbe vulnerabilità ai rainbow-attack, e nel caso in cui due utenti abbiano la stessa password, allora avranno lo stesso hash, rendendo più semplice per un attaccante dedurre che le due password sono identiche. Il lato positivo dell'utilizzo di sale randomico è che un eventuale attaccante dovrebbe costruire tabelle hash con ogni salt possibile, anche se da un altro lato, bisogna necessariamente salvare il salt insieme all'hash, per poter consentire il corretto matching durante il login.

E' stata testata la funzione *crypt()* di PHP senza alcun parametro, se non la password da hashare, che di default usa DES con salt automatico, ma non è più supportata in quanto si necessita l'inserimento di un sale manuale, diverso da quello automatico che invece era debole.

2.4. Possibili difese

Per mitigare attacchi *brute-force*, *dizionario* o *rainbow table*, sarebbe opportuno utilizzare algoritmi di hashing abbastanza stabili come Bcrypt o Argon2, opportunamente settati con un parametro di ‘cost’ abbastanza alto in modo da rendere troppo costoso e temporalmente impossibile il *cracking* della password da parte di un attaccante. Come si potrebbe erroneamente pensare, non è la lunghezza dell’hash generato a determinare la sua forza, ma il costo computazionale ovvero le risorse (cicli di CPU e RAM) necessarie a calcolare l’hash. In ciò Argon2 si rivela più sicuro di Bcrypt in quanto impiega anche un’elevata quantità di RAM. La tecnica appena descritta su cui si basano questi due algoritmi prende il nome di *key stretching*. Anche con una GPU o HW personalizzato, se settati opportunamente bene, rendono vani i tentativi di attacchi *forza bruta* e *dizionario*. Ovviamente bisogna scegliere il parametro ‘cost’ in maniera opportuna in base alle risorse computazionali del server che ha il compito di applicare la funzione hash ad ogni registrazione o login. Esso, infatti, che in Bcrypt è settabile tra il range 4-31, se troppo alto rispetto le proprie risorse computazionali, comporta un netto peggioramento delle prestazioni; quindi, è importante trovare un buon raccordo tra funzionalità e sicurezza. Esso genera anche un salt randomico e unico.

Considerando che l’attaccante è avvantaggiato dalla conoscenza del salt e dei round (cost), in quanto salvati insieme alla password, potrebbe essere una buona idea, anche se complessa, quella di salvare essi separatamente. Ciò necessiterebbe l’uso di funzioni di manipolazione delle stringhe in PHP per estrarre i componenti corretti dalla stringa di hash, e per poi assemblarli in fase di login per controllo. Questa tecnica in realtà non è consigliata a causa della sua complessità di gestione dell’efficacia e di eventuali aggiornamenti del salt. In alternativa si potrebbero anche crittografare gli hash nel DB, per esempio tramite AES.

Ecco riportato un tentativo di cracking di una password hashata con Bcrypt con valore di cost = 19, cioè con un totale di 542.288 iterazioni di Bcrypt (2^{19}):

```
Session.....: hashcat
Status.....: Running
Hash.Mode.....: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target.....: $2y$19$1pyyWz9Q5I55r02g8eChmeULh06HqcI5Nv97aIAcivYK ... Upf2W.
Time.Started.....: Fri Jun 30 17:12:12 2023 (16 secs)
Time.Estimated...: Fri Oct 25 11:54:27 2030 (7 years, 116 days)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 0 H/s (7.16ms) @ Accel:8 Loops:8 Thr:1 Vec:1
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 0/14344389 (0.00%)
Rejected.....: 0/0 (0.00%)
Restore.Point....: 0/14344389 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:14056-14064
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 → elizabeth (DCC1, MS cache
Hardware.Mon.#1..: Util: 61%
```

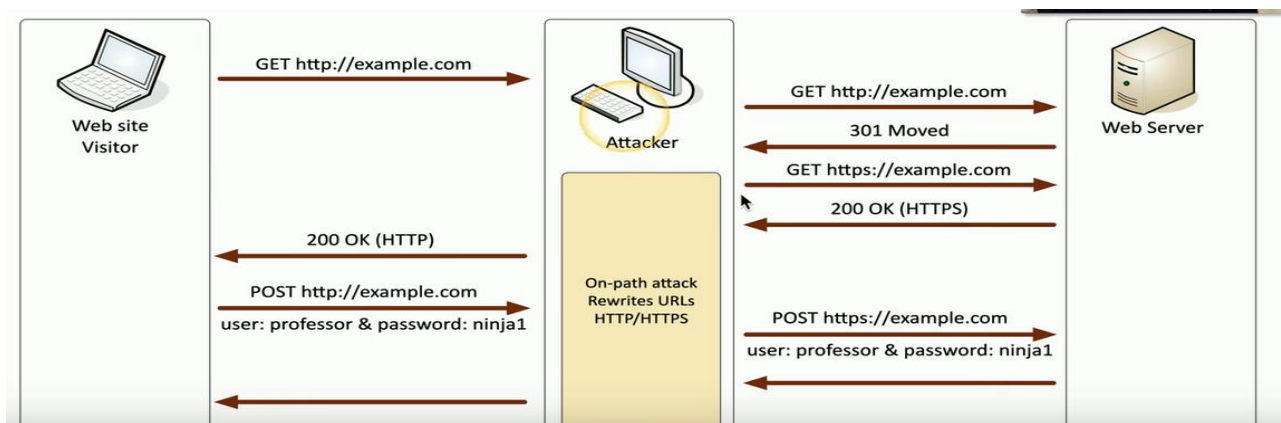
3. Attacco 2: dati in transito

3.1. Ipotesi

In questo caso di studio, è stato analizzato un attacco alla crittografia mancante o non sufficiente su dati in transito, cioè in rete. In particolare, si è supposto che la vittima navigasse sul web tramite una rete wireless insicura. Un attaccante collegato alla stessa rete, in seguito ad un attacco di *ARP poisoning*, intercetta tutto il traffico tra la vittima e il router. Più precisamente, se la vittima dovesse visitare siti HTTP, l'attaccante riuscirebbe a vedere tutti i pacchetti in chiaro, inclusi eventuali dati sensibili come le credenziali di accesso; nel caso in cui la vittima visiti invece siti HTTPS, in alcuni casi, cioè quelli in cui il browser usato è vecchio o usa versioni obsolete oppure il sito web visitato è datato o non aggiornato, non sarà presente l'header HSTS.

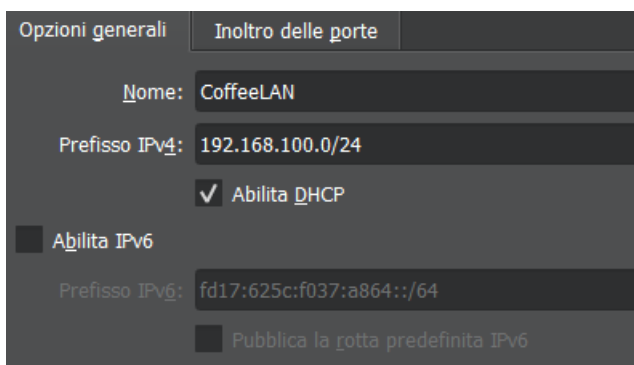
L'header HSTS (HTTP Strict Transport Security), per i siti che lo supportano, viene inviato dal web-server al web-browser che ha fatto una richiesta, obbligandolo, se esso è compatibile ad HSTS, a forzare qualsiasi richiesta da HTTP ad HTTPS. Nel caso in cui, l'implementazione di questo header manchi anche solo in una delle due parti, allora sarà possibile effettuare un attacco di SSL Stripping. Esso consiste nel *downgrade* di un sito web da HTTPS a HTTP, e di conseguenza lo *sniffing* di pacchetti. Ovviamente ciò necessita un precedente attacco *Man-In-The-Middle* per intercettare il traffico e reindirizzare il traffico tra vittima, router e server. L'obiettivo è stato quello di provare ad applicare SSL Stripping al variare dei browser e dei siti-web e successivamente di dimostrare quanto è debole un protocollo senza crittografia come HTTP, in cui i dati sono in chiaro. Se il server è impostato per accettare la prima richiesta in HTTP per stabilire la connessione, quindi sulla porta 80, l'attaccante può sfruttare questa vulnerabilità per creare un canale sicuro col web-server, lo decifrerà e ne creerà uno non cifrato con la macchina vittima.

Eccone un esempio grafico:

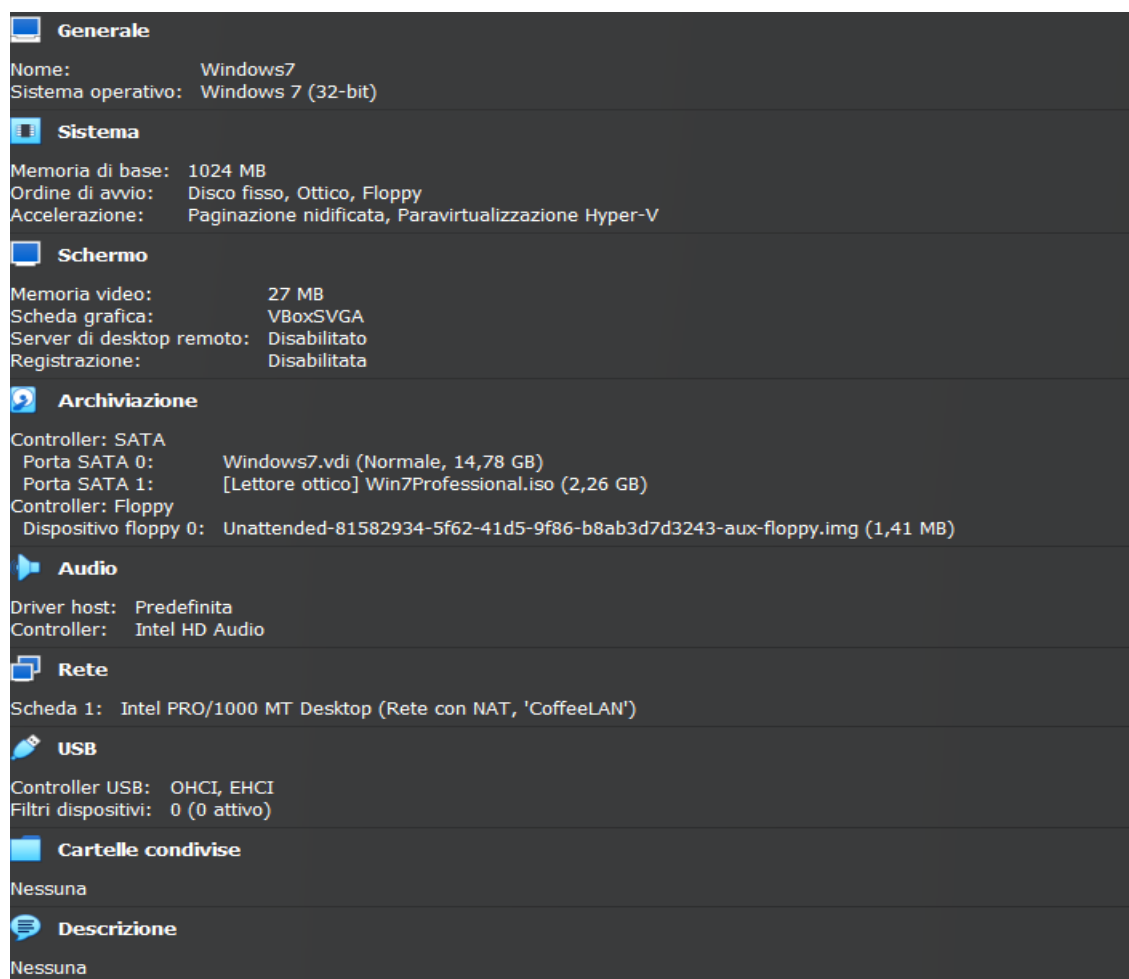


3.2. Premesse tecniche

Per la costruzione del laboratorio, sono state utilizzate tre macchine virtuali tramite VM Virtual Box: la macchina attaccante Kali Linux, e due macchine vittima, Kali Linux (di nome KaliLinuxVictim) e Windows 7. È importante sottolineare come la macchina vittima e attaccante devono essere all'interno della stessa rete affinché si possa tentare l'attacco, e soprattutto deve essere insicura, per esempio senza l'utilizzo di VLAN; infatti, le schede di rete di tutte e tre le macchine sono state impostate come 'Rete con NAT'. Questo ha permesso sia alle macchine vittima di accedere ad Internet e di inviare richieste a vari siti web, sia di creare una sottorete virtuale interna e condivisa tra tutte e tre le macchine, simulando una LAN reale. In particolare, esse sono state collegate alla rete con NAT di nome 'CoffeeLAN', appositamente creata in VM Virtual Box; ciò è stato possibile andando su *file->strumenti-> gestore di rete*. Ecco le impostazioni della sottorete virtuale 'CoffeeLAN':



Le impostazioni delle macchine Kali Linux sono rimaste invariate rispetto all'attacco agli hash delle password sopra citato. La macchina Windows 7 è stata invece installata con le seguenti specifiche tecniche:

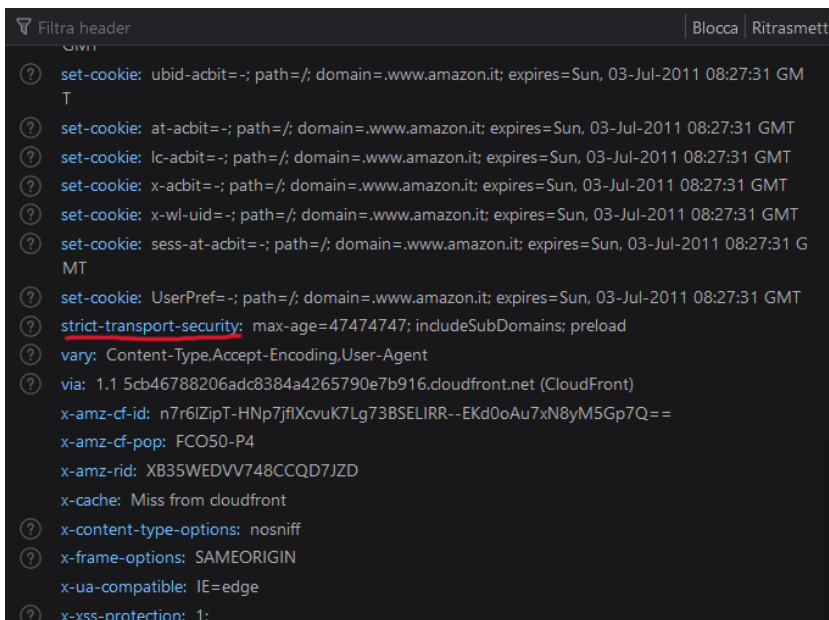


Nella macchina KaliLinuxVictim sono stati utilizzati i browser Mozilla (preinstallato) e Microsoft Edge aggiornati. La macchina Windows 7 è stata invece utilizzata appositamente per usare Internet Explorer 8.0. Questo al fine di visitare lo stesso sito web in seguito ad un attacco di SSL Stripping tramite più browser, per verificare eventuali variazioni di comportamento.

Nella macchina Kali Linux attaccante sono stati utilizzati i tool preinstallati *Ettercap* e *Wireshark*. *Ettercap* ha permesso di effettuare l'*ARP poisoning* della macchina vittima, e successivamente *Wireshark* è stato utilizzato per sniffing e analisi del traffico dati. È stato inoltre installato *mitmproxy* tramite il comando digitato su shell '*apt-get install mitmproxy*'. Esso ha consentito di modificare il normale comportamento del proxy dell'attaccante, sfruttando un particolare script personalizzato in python '*sslstrip.py*', scaricato dalla rete al link annesso alle fonti in calce alla relazione; esso in sostanza ha lo scopo di fare il *downgrade* della richiesta da HTTPS ad HTTP.

3.3. Demo Attacco

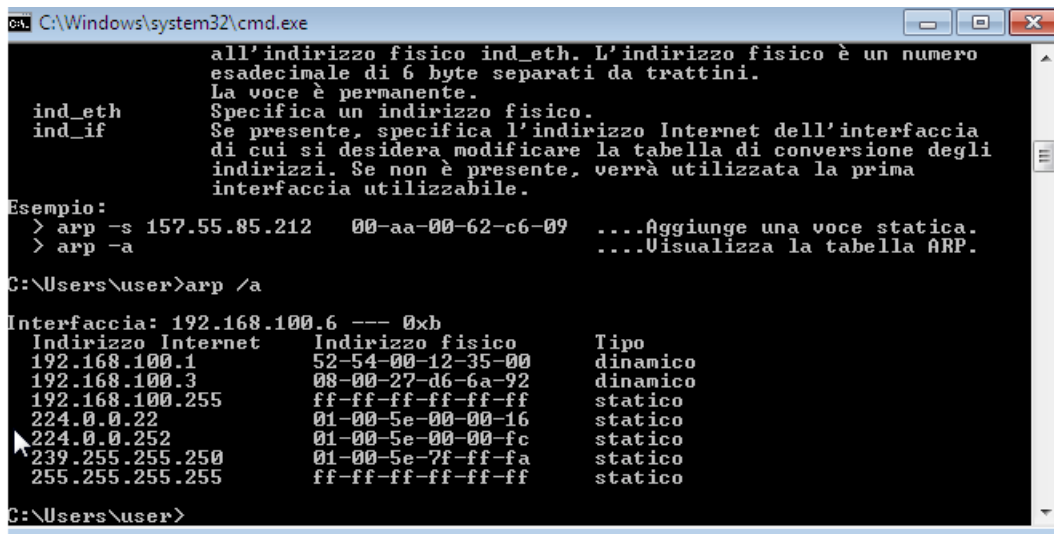
In primis sono state avviate le macchine virtuali da VM VirtualBox, in particolare Kali Linux attaccante e Windows 7 vittima. Prima di eseguire l'attacco ci si è accertati che sulla macchina vittima, i siti web usati per il test fossero con protocollo HTTPS e che supportassero HSTS. Per il primo basta controllare l'URL, mentre per il secondo bisogna seguire la seguente procedura: entrare negli strumenti di sviluppo del browser tramite F12 (mentre si è nel sito web in questione), andare nella sezione 'Rete', aggiornare la pagina e cercare la richiesta del sito verso il browser dell'URL principale (html), andare nella sezione Header e cercare *strict-transport-security*, se esso è presente allora significa che il sito usa l'header HSTS. Ecco in figura un esempio:



A questo punto è stato verificato che le due macchine appartenessero alla stessa sottorete e che usassero lo stesso gateway. In particolare, la macchina attaccante ha indirizzo IP 192.168.100.4 (col comando *ifconfig*), mentre la macchina vittima sta sull'indirizzo 192.168.100.6 (comando *ipconfig*). Il gateway sta invece per entrambi sull'indirizzo 192.168.100.1 (su Kali Linux bisogna cercarlo esplicitamente tramite il comando *route*).

Dopo di che è stato eseguito il comando '*arp -a*' sulla macchina vittima, per vedere l'elenco delle corrispondenze IP-MAC attualmente memorizzate nella cache ARP del dispositivo. Qui è bene sottolineare che l'indirizzo IP serve per comunicare con dispositivi esterni alla rete locale; invece, l'indirizzo MAC identifica la scheda di rete di un preciso dispositivo all'interno di una rete locale, e viene usato per comunicare

con dispositivi della stessa rete. Il risultato dell'esecuzione del comando sul prompt dei comandi è stato il seguente:



```

C:\Windows\system32\cmd.exe

all'indirizzo fisico ind_eth. L'indirizzo fisico è un numero
esadecimale di 6 byte separati da trattini.
La voce è permanente.
ind_eth Specifica un indirizzo fisico.
ind_if Se presente, specifica l'indirizzo Internet dell'interfaccia
di cui si desidera modificare la tabella di conversione degli
indirizzi. Se non è presente, verrà utilizzata la prima
interfaccia utilizzabile.

Esempio:
> arp -s 157.55.85.212 00-aa-00-62-c6-09 ....Aggiunge una voce statica.
> arp -a ....Visualizza la tabella ARP.

C:\Users\user>arp /a

Interfaccia: 192.168.100.6 --- 0xb
Indirizzo Internet Indirizzo fisico Tipo
192.168.100.1 52-54-00-12-35-00 dinamico
192.168.100.3 08-00-27-d6-6a-92 dinamico
192.168.100.255 ff-ff-ff-ff-ff-ff statico
224.0.0.22 01-00-5e-00-00-16 statico
224.0.0.252 01-00-5e-00-00-fc statico
239.255.255.250 01-00-5e-7f-ff-fa statico
255.255.255.255 ff-ff-ff-ff-ff-ff statico

C:\Users\user>

```

Si può notare come all'indirizzo IP del gateway corrisponde attualmente l'indirizzo MAC '52-54-00-12-35-00'. L'obiettivo della macchina attaccante è quello di eseguire un attacco di *ARP Spoofing* tramite *ettercap*. Esso consiste nella modifica delle *ARP Table* sia del gateway che della macchina vittima tramite l'invio di pacchetti ARP falsificati. Lo scopo è quello di far credere alla macchina vittima che la macchina attaccante sia il gateway, e al gateway che la macchina attaccante sia invece la macchina vittima. In altre parole, la macchina attaccante fa esattamente da *Man-In-The-Middle*, interponendosi tra macchina vittima e gateway. Nella normale esecuzione le richieste eseguite dalla macchina vittima vengono inviate al gateway e viceversa il gateway invierà le risposte direttamente alla macchina vittima; se l'attacco invece ha avuto successo la macchina attaccante fa da collante in maniera del tutto trasparente: prende le richieste della macchina vittima e le invia al gateway, e viceversa.

Affinché esso funzioni sia bene che si stia su una rete insicura, cioè che non implementi VLAN o che non abbia i firewall opportunamente configurati per limitare il traffico di ARP sospetto. Inoltre, essendo l'*ARP table* una cache, al riavvio del sistema della macchina vittima le corrispondenze IP-MAC torneranno a quelle di default.

Per eseguire l'attacco di *ARP Spoofing* è stato avviato *ettercap* all'interno della macchina attaccante. È stata selezionata l'interfaccia di rete utilizzata dalla sottorete virtuale 'CoffeeLAN', in questo caso '*eth0*'. Dopo di che è stata eseguito uno scan degli host nella rete '*eth0*', ottenendo la lista di tutte le macchine collegate ad essa, andando in *opzioni* -> *Scan for Hosts*. Sempre in *opzioni* si trova la casella *host list*, dentro la quale è adesso possibile prendere visione dell'analisi fatta in precedenza:

| Host List x | | |
|---|-------------------|-------------|
| IP Address | MAC Address | Description |
| 192.168.100.1 | 52:54:00:12:35:00 | |
| 192.168.100.2 | 52:54:00:12:35:00 | |
| 192.168.100.3 | 08:00:27:D6:6A:92 | |
| 192.168.100.5 | 08:00:27:0E:86:42 | |
| fe80::959f:dd8:8b5d:2474 | 08:00:27:BF:37:4F | |
| 192.168.100.6 | 08:00:27:BF:37:4F | |
| <div> Delete Host Add to Target 1 Add to Target 2 </div> | | |
| Randomizing 255 hosts for scanning... Scanning the whole netmask for 255 hosts... 5 hosts added to the hosts list... DHCP: [08:00:27:C7:E1:36] REQUEST 192.168.100.4 DHCP: [192.168.100.3] ACK : 192.168.100.4 255.255.255.0 GW 192.168.100.1 DNS 192.168.1.254 | | |

Come è possibile notare sono presenti l'indirizzo del gateway e della macchina vittima. Adesso, per eseguire l'*ARP Spoofing*, detto anche *ARP Poisoning*, è stato selezionato l'indirizzo del gateway e aggiunto al target 1, l'indirizzo della macchina vittima e aggiunto al target 2. Fatto ciò, tramite la sezione MITM, è stato possibile eseguire l'attacco tramite la casella apposita *ARP Poisoning*. Per verificare che l'attacco sia andato a buon fine è stata controllata l'*ARP Table* nella macchina vittima: l'indirizzo MAC del gateway corrispondente al suo indirizzo IP è cambiato, in particolare corrisponde a quello della macchina attaccante '*08-00-27-c7-e1-36*':

```
Interfaccia: 192.168.100.6 --- 0xb
Indirizzo Internet    Indirizzo fisico    Tipo
192.168.100.1         08-00-27-c7-e1-36  dinamico
192.168.100.3         08-00-27-d6-6a-92  dinamico
192.168.100.4         08-00-27-c7-e1-36  dinamico
192.168.100.255      ff-ff-ff-ff-ff-ff  statico
```

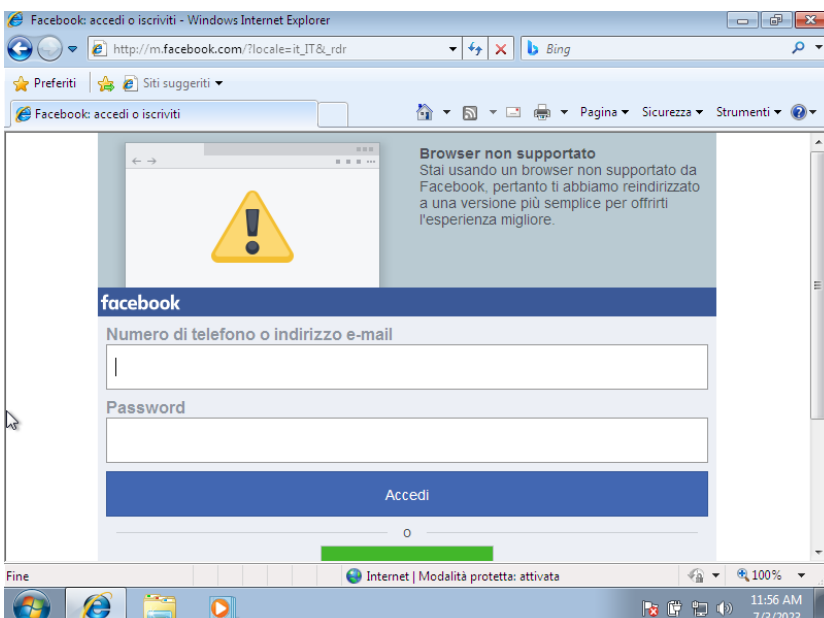
Fatto ciò, la macchina attaccante era già in grado di intercettare il traffico tra la macchina vittima e i server a cui essa facesse richiesta, ma essendo ormai la maggior parte dei siti in HTTPS, il traffico è risultato cifrato; quindi, si è optato nel tentativo di effettuare il downgrade dei siti HTTPS in HTTP.

Per fare ciò è stato utilizzato lo script `'sslstrip.py'`. In particolare, il proxy adesso presente nella macchina attaccante è stato settato opportunamente affinché modificasse le richieste ricevute dalla macchina vittima e dal server, tramite il comando `'mitmdump -s sslstrip.py -m transparent'`. Tramite il comando `'-s'` è stato selezionato lo script da eseguire sulla richiesta, mentre `'-m transparent'` ha permesso di specificare il modo di funzionamento di mitmproxy: in questo caso transparent è una modalità tramite la quale è possibile intercettare e analizzare il traffico senza richiedere alcuna configurazione aggiuntiva sul client.

```
(kali@kali)~[/Downloads]
$ mitmdump -s sslstrip.py -m transparent
[05:30:49.785] Loading script sslstrip.py
[05:30:49.797] transparent proxy listening at *:8080.
```

Fatto ciò, è stato necessario configurare il sistema operativo della macchina attaccante in modo che le richieste in uscita del client vengano reindirizzate a Mitmproxy usando regole di reindirizzamento del traffico, in particolare le *iptables*. È stato quindi aperto un altro terminale sul quale è stato eseguito il comando `'sudo iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-ports 8080'`. Il parametro `-t` con opzione `nat` è stato usato per agire sulla tabella NAT; il parametro `-A PREROUTING` consente di instradare le richieste in arrivo su una porta verso un'altra specifica porta, prima dell'arrivo alla porta standard; il parametro `-p tcp` indica il protocollo utilizzato; con i parametri seguenti è stato invece impostato che tutte le richieste alla porta 80 venissero reindirizzate alla porta 8080, che è quella su cui il Mitmproxy è in ascolto. Per eseguire questa manipolazione è necessario essere utente root.

Infine, l'attacco è stato compiuto a termine utilizzando *wireshark*, il quale ha permesso di visionare il traffico di rete della macchina attaccante. È stata selezionata l'interfaccia di rete `'eth0'`. Nel momento in cui la macchina vittima si è collegata ad un sito HTTPS, è stato subito notato come questa si fosse presentata invece in HTTP:



Una volta che la vittima ha inserito le sue credenziali per eseguire l'accesso al sito, l'attaccante ha potuto fare *sniffing* di pacchetti, in particolare usando il filtro di *wireshark* solo sulle richieste HTTP, è stato trovato il pacchetto esatto di richiesta di login da parte della macchina vittima:

```

1134 208.508628821 192.168.100.6 31.13.86.36 HTTP 914 POST /login/device-base
1172 213.856242843 31.13.86.36 192.168.100.6 HTTP 189 HTTP/1.1 302 Found
1174 213.866432746 192.168.100.6 31.13.86.36 HTTP 718 GET /login/?email=enzo@gmail.com
1190 213.984899912 31.13.86.36 192.168.100.6 HTTP/X... 726 HTTP/1.1 200 OK
1196 213.991895709 192.168.100.6 157.240.231.35 HTTP 574 GET /security/hsts-pixel
1225 214.170956840 157.240.231.35 192.168.100.6 HTTP 472 HTTP/1.1 302 Found
1229 214.174234966 192.168.100.6 157.240.231.35 HTTP 494 GET /security/hsts-pixel
1258 214.425691924 157.240.231.35 192.168.100.6 HTTP 468 HTTP/1.1 302 Found
1262 214.428403932 192.168.100.6 157.240.231.35 HTTP 490 GET /security/hsts-pixel
1294 214.663707751 157.240.231.35 192.168.100.6 HTTP 122 HTTP/1.1 200 OK (GIF89a)

[HTTP request 1/2]
[Response in frame: 1172]
[Next request in frame: 1174]
File Data: 176 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "lsd" = "AVoXpVH9jQc"
  Form item: "jazoest" = "2963"
  Form item: "m_ts" = "1688378181"
  Form item: "li" = "RZuiZE9rH9rmBjThbXWDtt-L"
  Form item: "try_number" = "0"
  Form item: "unrecognized_tries" = "0"
  Form item: "email" = "enzo@gmail.com"
  Form item: "pass" = "InternetSecurity23!"
  Form item: "login" = "Accedi"
  Form item: "bi_xrwh" = "0"
  
```

Analizzando il seguente pacchetto, accedendo alla sezione del form HTML, è stato possibile notare come le credenziali siano in chiaro.

L'attacco, in questo caso, è andato a buon fine perché il browser Internet Explorer non supporta l'header HSTS, nonostante la pagina web visitata dalla vittima lo avesse richiesto. Questo perché, come detto nel paragrafo 'Ipotesi', se anche solo una delle due parti non supporta l'header, non è possibile farne uso e quindi forzare la richiesta in HTTPS. Di conseguenza qualsiasi sito web visitato tramite Internet Explorer in seguito ad un attacco di SSL Stripping, porterebbe ad effettuare richieste HTTP. È risultato con molta evidenza come navigare con browser che non supportano HSTS, o in alternativa visitare siti che non lo supportano o che direttamente siano su HTTP, comporti un importante problema di segretezza della comunicazione a causa della mancanza o insufficiente crittografia.

Le stesse procedure sono state testate con la seconda macchina vittima, KaliLinuxVictim, con i browser Mozilla e Microsoft Edge. In entrambi i casi si è notato che, essendo dei browser odierni e aggiornati, supportano gli header HSTS. In particolare, l'attacco e quindi la connessione in HTTP della vittima sono riusciti solo in due particolari siti, *hotmail.com* e *hotels.com*, ma solo al primo accesso; cioè, cercando di accedere nuovamente agli stessi link con lo stesso browser, le richieste

venivano poi inviate in HTTPS. Questo perché, in seguito alla prima richiesta, l'HSTS viene salvato nella cache del browser (in quanto compatibile) e le seguenti richieste venivano direttamente fatte in HTTPS. Inoltre, l'accesso alla pagina HTTP riguardava solamente le schermate iniziali in cui non era presente nessun form in cui poter inserire dati sensibili; proseguendo da questa schermata ad un'altra non principale, esse sono state ricevute in HTTPS.

3.4. Possibili difese

Innanzitutto, è bene sottolineare che oggi una grandissima percentuale dei siti supportano l'header HSTS, a meno di siti obsoleti o opportunamente realizzati con intenti malevoli. Lo stesso ragionamento può essere fatto per i browser principali, come Mozilla, Opera, Microsoft Edge e altri. Infatti, per dimostrare la vulnerabilità è stato necessario utilizzare un browser ormai obsoleto, Internet Explorer. Per evitare un attacco del genere quindi, a parte la sensibilizzazione dell'utente sull'attenzione dell'URL per ogni sito visitato, basta utilizzare browser moderni e aggiornati. Inoltre, la maggior parte dei siti web comunica ormai esclusivamente sulla porta 443, così che nemmeno la prima richiesta sulla porta 80 venga accettata, ma reindirizzata alla 443, vanificando l'attacco. La ragione principale dietro questa possibilità era che il meccanismo di HSTS richiede che il browser riceva l'header HSTS dal server per la prima volta durante la comunicazione iniziale. Se la comunicazione iniziale avviene su HTTP, per questioni di compatibilità, invece di HTTPS, l'attaccante può intercettare la richiesta e rimuovere o modificare l'header HSTS. Di conseguenza, il browser non riceverebbe l'indicazione di utilizzare sempre HTTPS per il sito specifico.

4. Fonti

OWASP: https://owasp.org/Top10/it/A02_2021-Cryptographic_Failures/

CWE correlate: <https://cwe.mitre.org/data/definitions/259.html>,
<https://cwe.mitre.org/data/definitions/327.html>,
<https://cwe.mitre.org/data/definitions/331.html>

Scribbr: <https://www.scribbr.com/category/research-paper/>

Attacco 1: <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>,
<https://crackstation.net/hashing-security.htm>,
<https://www.exploit-db.com/docs/english/14710-cracking-salted-hashes.pdf>

Attacco 2: <https://cybersudo.org/downgrade-https-to-http--man-in-the-middle-attack/>,
https://owasp.org/www-pdf-archive/SSL_Spoofing.pdf,
<https://www.geeksforgeeks.org/what-is-arp-spoofing-attack/>

Script sslstrip.py: <https://cybersudo.org/wp-content/uploads/2022/04/sslstrip.rar>