POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Attacchi al protocollo TLS: dalla teoria alla pratica

Relatori
Prof. Antonio Lioy
Dr. Ing. Diana Berbecaru

Candidato
Davide Giorda

Anno accademico 2020-2021

† A mio nonno Gino
† A mio nonno Adolfo

Sommario

Ogni giorno, ognuno di noi tramite il proprio cellulare o il proprio computer si collega e naviga sulla rete Internet. Questo porta alla creazione di innumerevoli connessioni, le quali devono essere create in modo sicuro al fine di proteggere i nostri dati.

Il Transport Layer Security, o TLS, è il protocollo crittografico, al momento, più diffuso al mondo. Viene utilizzato nel campo delle telecomunicazioni e dell'informatica per fornire la privacy e la sicurezza dei dati delle comunicazioni Internet. Il caso d'uso principale di questo protocollo sono le comunicazioni tra applicazioni web e server, sebbene TLS sia ampiamente utilizzato anche in altri campi come le e-mail, la messaggistica e Voice-over-IP. Tuttavia, l'evoluzione di TLS va di pari passo con l'evoluzione degli attacchi informatici, che mirano a sfruttare le vulnerabilità e i bug di progettazione di questo protocollo.

Questa tesi analizza le principali funzioni del protocollo TLS e buona parte dei possibili attacchi che possono essere messi in atto, sfruttando le vulnerabilità che possono essere ancora presenti nei vari sistemi informatici. Viene fatta luce sia sulle problematiche che rendono, tutt'oggi, TLS bersaglio degli attacchi e vengono spiegate quali possono essere le contromisure da prendere per mitigare questi attacchi o prevenirli del tutto.

Per poter rilevare i sistemi informatici ancora affetti da queste vulnerabilità è stata fatta una ricerca di possibili strumenti e tool utili sia difensivi sia offensivi. Per il monitoraggio delle connessioni sono stati analizzati alcuni IDS come *Zeek* e *Suricata*, i quali forniscono importanti informazioni che possono essere utilizzate per scovare alcune vulnerabilità. Per quanto riguarda la parte di attacco, sono stati presi in considerazione sia tool commerciali come *Metasploit*, sia tool meno noti come *TLS-Attacker*, il cui codice è recuperabile su GitHub. Questi tool ci hanno permesso di rilevare la presenza o meno delle vulnerabilità sui sistemi informatici e, successivamente, anche di eseguire veri e propri attacchi.

Infine, vengono mostrati passo dopo passo gli attacchi informatici che sono stati eseguiti contro i sistemi vulnerabili. É stata definita una strategia che permettesse di creare un processo in cui gli strumenti per il rilevamento delle vulnerabilità collaboravano con quelli per eseguire gli attacchi. Sono stati eseguiti tre attacchi: un attacco Man-In-The-Middle per il furto delle credenziali di accesso ad un social network, un attacco Heartbleed per il furto di informazioni dalla memoria del sistema informatico vittima ed un attacco Padding Oracle per la decifrazione del testo crittografato di una connessione. Questi esperimenti mettono in luce quanto sia relativamente semplice reperire e sfruttare tool che si trovano in rete per eseguire attacchi informatici contro il protocollo TLS.

Indice

Capitolo 1

Transport Layer Security - TLS

Nel primo capitolo della mia tesi viene fatta una panoramica sul protocollo TLS, nonchè oggetto di studio, mettendo in mostra l'evoluzione del protocollo dalla sua nascita fino ai giorni nostri ed il suo funzionamento. Successivamente verranno analizzate alcune categorie di attacchi contro questo protocollo che sono stati scoperti nel corso degli anni, spiegando inoltre le contromisure che possono essere prese per evitarli.

1.1 Introduzione a TLS

Il protocollo TLS, o "Transport Layer Security", è un protocollo crittografico utilizzato per le comunicazioni sicure sulle reti TCP/IP come Internet. La necessità di mantenere la comunicazione confidenziale è, tutt'ora, uno scenario estremamente comune, in particolar modo quando le informazioni che vengono trasmesse sono dati sensibili come password, informazioni bancarie o dati della carta di credito. Questi dati, quando vengono trasmessi in rete, possono essere soggetti ad intercettazione ad opera di malintenzionati, i quali entrerebbero in possesso di informazioni riservate. Perciò, si è ritenuto necessario impiegare le tecnologie utili per assicurarsi che questi dati non fossero comprensibili a terze parti e non potessero essere alterati durante il trasporto. Dunque, il protocollo TLS fornisce le seguenti proprietà:

Autenticazione: tutti i partecipanti alla comunicazione devono verificare l'identità delle controparti. Un utente malintenzionato potrebbe impersonare l'identità di un interlocutore e ricevere informazioni riservate, perciò è necessario certificare l'identità delle parti remote.

Integrità: è necessario introdurre la possibilità di verificare se i dati hanno subito modifiche durante il trasporto, che siano causate da errori durante la trasmissione oppure da terze parti che intercettano e modificano i dati della comunicazione.

Riservatezza: i dati devono essere illeggibili per chiunque non sia un partecipante alla comunicazione, solo loro devono essere in grado di interpretare i dati ricevuti

Il protocollo TLS si trova al livello 5 all'interno della pila ISO/OSI, ovvero tra il livello del protocollo dell'applicazione e il livello TCP/IP. In questo modo ha la possibilità di proteggere i dati dell'applicazione ed inviarli al livello di trasporto ed in più, grazie alla posizione in cui si trova, ha anche la possibilità di supportare diversi protocolli del livello applicazione. Sebbene il protocollo applicativo più diffuso al momento sia HTTP, esistono anche altri protocolli, tuttavia meno conosciuti, come SMTP, FTP, TELNET, IMAP4 e POP3.

1.1.1 Da SSL a TLS

Prima di prendere il nome di Transport Layer Security (TLS), il protocollo si chiamava Secure Sockets Layer o SSL.

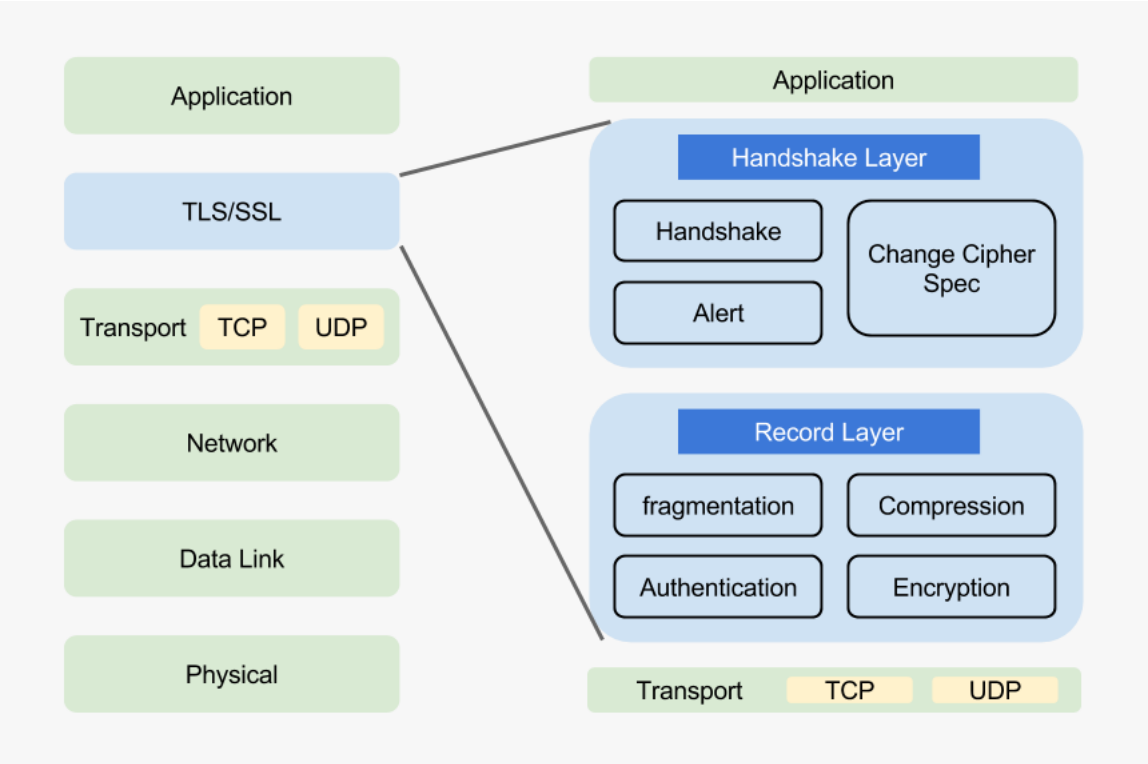

Figura 1.1. Protocollo TLS all'interno della pila ISO/OSI.

Con il nome di SSL, la versione 1.0 di questo protocollo non fu mai rilasciata al pubblico a causa di molteplici problemi di sicurezza. Perciò nel 1995, la prima versione del protocollo ad essere rilasciata fu SSL 2.0. Tuttavia, furono riscontrati problemi di sicurezza anche in questa versione che fu prontamente sostituita l'anno successivo, il 1996, da SSL 3.0. Purtroppo anche questa versione non ebbe una sorte migliore, infatti dopo poco tempo l'IETF ("Internet Engineering Task Force"), ovvero il responsabile dell'evoluzione tecnica e tecnologica di Internet, respinse SSL 3.0 a causa della presenza di nuove falle di sicurezza.

Nasce così nel 1999 la prima versione ufficiale del protocollo TLS, chiamata TLS 1.0. Come dichiarato nel rispettivo RFC 2246 [1]:

> "Le differenze tra questo protocollo e SSL 3.0 non sono drastiche, ma sono significative abbastanza che TLS 1.0 e SSL 3.0 lavorano insieme (sebbene TLS 1.0 incorpori un meccanismo per cui un'implementazione TLS può tornare a SSL 3.0)"

Rispetto alla versione 3.0 di SSL, TLS 1.0 migliora particolarmente in quattro campi:

sicurezza crittografica: in modo da poter essere utilizzato per stabilire connessioni sicure tra due parti;

interoperabilità: per permettere ai programmatori di sviluppare applicazioni utilizzando TLS, che permetterà di scambiare parametri crittografici senza dover conoscere un altro codice;

estensibilità: cercando di fornire un framework in cui è possibile incorporare nuove chiavi pubbliche e metodi di crittografia di massa, se necessario;

efficienza relativa: le operazioni crittografiche tendono a consumare molta CPU, in particolare quelle a chiave pubblica. Il protocollo TLS incorpora uno schema di cache di sessione opzionale per ridurre il numero di connessioni che devono essere stabilite da zero.

La prima versione di TLS dura 7 anni, fino a quando viene sostituita da TLS 1.1 nel 2006. Questa nuova versione non introduce grandi cambiamenti rispetto alla prima, tant'è che passano solamente due anni prima che esca nuovamente un versione più recente del protocollo. In particolare, la versione TLS 1.1 [2]effettua dei piccoli miglioramenti sull'utilizzo del vettore di inizializzazione (IV) e sulla gestione degli errori nella modalità CBC per proteggersi maggiormente dagli attacchi contro questa modalità; ed infine, aggiunge qualche informazioni su diversi attacchi scoperti contro TLS.

Ma i cambiamenti maggiori si hanno con il rilascio di TLS 1.2 nel 2008 [3], il quale offre una maggiore sicurezza contro gli attacchi informatici e consente alle applicazioni una flessibilità maggiore per quanto riguarda le cipher suites utilizzate. Le principali migliorie introdotte da questa versione possono esser riassunte così:

- aggiunta del supporto per la crittografia autenticata con modalità dei dati aggiuntive;
- descrizione per evitare gli attacchi di Bleichenbacher e Klima;
- TLS_RSA_WITH_AES_128_CBC_SHA obbligatoria per implementare la suite di cifratura;
- aggiunta delle suite di cifratura HMAC-SHA256;
- rimozione delle suite di cifratura IDEA e DES;
- il supporto per la retrocompatibilità con SSLv2 passa da DOVREBBE a PUÓ, con la possibilità di diventare NON DOVREBBE.

Nel frattempo, nel 2011 e nel 2015, vengono ufficialmente deprecate rispettivamente le versioni SSL 2.0 e SSL 3.0, poiché considerate non sicure e viene dunque sconsigliato il loro utilizzo.

Infine nel 2018 esce l'ultima versione di TLS, ovvero TLS 1.3. L'RFC 8446 appartenente a questa versione [4], oltre a descrivere le differenze rispetto alla versione 1.2, introduce alcuni miglioramenti che interessano proprio la versione 1.3, tra questi sono presenti:

- la descrizione di un meccanismo di protezione contro il downgrade di versione;
- la definizione degli schemi probabilistici di firma chiamati RSASSA-PSS;
- l'apertura a usare l'estensione *supported_versions* del *ClientHello* per negoziare la versione di TLS da utilizzare, in preferenza al campo *legacy_version*;
- la possibilità di utilizzare, da parte del client, l'estensione *signature_algorithms_cert* per indicare gli algoritmi di firma che può validare nei certificati X.509.

1.1.2 TLS 1.3

Ad oggi, dicembre 2021, la versione più recente del protocollo TLS è la 1.3, rilasciata al pubblico tre anni fa. L'obiettivo principale che puntava a raggiungere l'IETF con l'introduzione di questa nuova versione era di ridurre le varie fasi operative riducendo notevolmente i tempi di caricamento delle pagine per i dispositivi mobili e non solo. Inoltre il protocollo fornisce miglioramenti decisi nelle aree della sicurezza, delle performance e della privacy. Sebbene la versione precedente del protocollo fosse considerata sicura, diverse vulnerabilità sfruttavano parti opzionali del protocollo e algoritmi obsoleti. TLS 1.3 ha deciso di rimuovere molte di queste problematiche e fornisce supporto solo per gli algoritmi che non presentano vulnerabilità note.

Nella sezione precedente abbiamo già visto quali sono i miglioramenti che introduce rispetto alla versione 1.2 del protocollo. Ma le novità principali introdotte per la versione 1.3 sono ancora di più e riguardano diversi aspetti del protocollo. La rimozione del supporto agli algoritmi obsoleti è una delle più grandi; infatti viene aggiornata la lista degli algoritmi a cifratura simmetrica supportati dal protocollo rimuovendo tutti quelli considerati "legacy". Tutti quelli che continuano a far parte di questa lista sono algoritmi AEAD ("Authenticated Encryption with Associated Data"), ovvero a crittografia autenticata. Inoltre viene modificato il concetto di suite di cifratura,

al fine di separare il meccanismo dell'autenticazione e di scambio delle chiavi dall'algoritmo di protezione dei record e un hash da utilizzare sia con la funzione di derivazione delle chiavi sia con il Message Authentication Code (MAC) dell'handshake. Proprio la funzione di derivazione delle chiavi subisce delle modifiche che migliorano le proprietà di separazione delle chiavi, permettendo un'analisi più semplice da parte dei crittografi. La funzione utilizzata come primitiva sottostante è la HKDF, ovvero la funzione di derivazione delle chiavi Extract-and-Expand basata su HMAC. Un ultimo cambiamento apportato è la rimozione delle ciphersuite RSA e Diffie-Hellman statiche, in questo modo ora tutti i meccanismi di scambio delle chiavi basati sulle chiavi pubbliche forniscono segretezza in avanti.

Per quanto riguarda il protocollo di handshake è stata aggiunta la modalità 0-RTT (Zero Round-Trip Time), che consente di risparmiare un viaggio di andata e ritorno nella configurazione della connessione per alcuni dati applicazione. Inoltre, da questa versione, tutti i messaggi dell'handshake successivi al messaggio *ServerHello* sono cifrati e il nuovo messaggio *EncryptedExtensions* permette a diverse estensioni, che precedentemente erano inviate in chiaro nel *ServerHello*, di godere di questa protezione. Viene inoltre ristrutturata significativamente la macchina a stati finiti dell'handshake per renderla più consistente e rimuovere alcuni messaggi superflui come il messaggio *ChangeCipherSpec*.

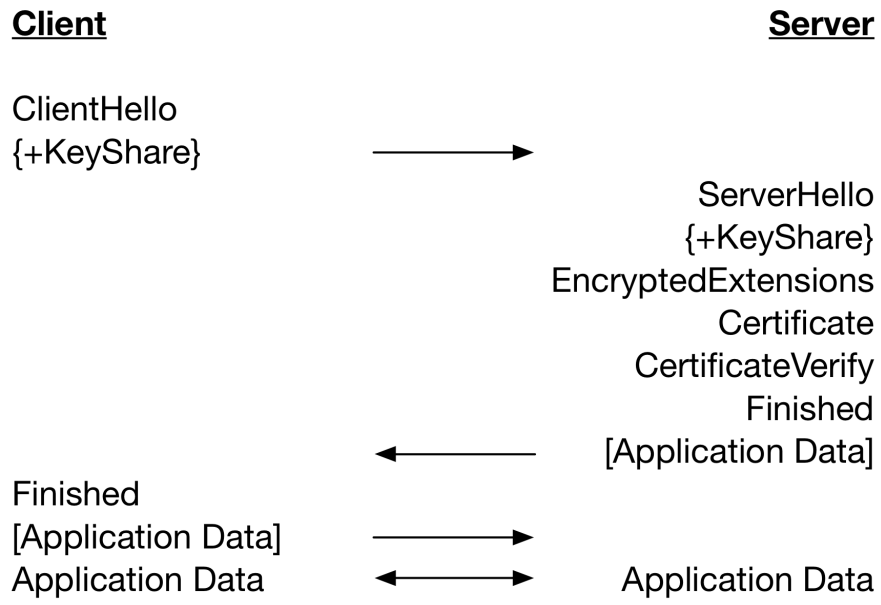

Figura 1.2. Flusso di messaggi per un handshake TLS 1.3 completo.

Gli algoritmi a curve ellittiche entrano a far parte ufficialmente delle specifiche di base del protocollo, e vengono inclusi nuovi algoritmi per la firma digitale come EdDSA ("Edwards-curve Digital Signature Algorithm"). In più, TLS 1.3 rimuove la negoziazione per il formato punto in favore di un singolo formato punto per ogni curva. Vengono inoltre effettuati ulteriori miglioramenti a livello crittografico, incluso il cambiare il padding RSA, per usare lo schema di firma probabilistico RSA (RSASSA-PSS), e la rimozione della compressione dell'algoritmo di firma digitale (DSA) e dei gruppi personalizzati "Ephermeral Diffie-Hellman" (DHE).

Il meccanismo della negoziazione della versione tipico di TLS 1.2 è stato deprecato in favore di una lista di versioni all'interno di un'estensione, in modo da incrementare la compatibilità con i server esistenti che implementavano, erroneamente, la negoziazione della versione. Infine, il meccanismo di recupero della sessione, con e senza stato, lato server e le ciphersuite basate su Pre Shared Key (PSK) delle versioni precedenti sono stati rimpiazzati da un nuovo singolo scambio PSK.

1.1.3 Protocollo di handshake

Una parte molto importante di TLS è il protocollo di handshake, un processo automatizzato che permette a due parti di negoziare i parametri di sicurezza di una connessione. All'inizio di una comunicazione è necessario che il client e il server si mettano d'accordo sulla versione del protocollo da utilizzare e sulla scelta degli algoritmi crittografici, opzionalmente si autentichino l'uno con l'altro, e usino tecniche di cifratura a chiave pubblica per generare i segreti condivisi. Come accennato nella sezione precedente, questo protocollo ha subito dei cambiamenti con l'introduzione della versione 1.3 di TLS, perciò ora verrà mostrato come è mutato nel tempo lo scambio di messaggi all'interno del protocollo, soffermandoci sulle motivazioni che hanno portato a questi cambiamenti.

Originariamente, nelle versioni precedenti a TLS 1.3, il protocollo di handshake prevedeva i seguenti passaggi:

- scambio dei messaggi di hello per mettersi d'accordo su algoritmi, scambio dei valori casuali, e verificare la ripresa della sessione;
- scambio dei parametri crittografici necessari al client e al server per mettersi d'accordo sul *PreMaster Secret*;
- scambio dei certificati e delle informazioni crittografiche per permettere al client e al server di autenticarsi;
- generazione del *Master Secret* a partire dal *PreMaster Secret* e dai valori casuali scambiati;
- fornire i parametri di sicurezza al livello record;
- permettere al client e al server di verificare che le parti abbiano calcolato gli stessi parametri di sicurezza e che l'handshake sia avvenuto senza manomissioni da parte di un attaccante.

Dunque, lo scambio dei messaggi cominciava come segue: il client invia un messaggio *ClientHello* al quale il server deve rispondere con un messaggio *ServerHello*, altrimenti avverrà un errore fatale e la connessione fallirà. Questi due messaggi vengono utilizzati per stabilire la versione del protocollo, l'identificativo di sessione, la ciphersuite e il metodo di compressione. Inoltre, vengono generati e scambiati due valori casuali chiamati *ClientHello.random* e *ServerHello.random*. In seguito ai messaggi di hello, nel caso debba essere autenticato, il server invia il suo certificato in un messaggio *Certificate* e può inoltre mandare un messaggio *ServerKeyExchange* se richiesto (per esempio, se non possiede un certificato o se il certificato è utilizzato solo per la firma). Nel caso avvenga l'autenticazione del server, questo può richiedere anche al client di autenticarsi mediante un messaggio *CertificateRequest*. Tuttavia, questi ultimi tre messaggi descritti sono opzionali e situazionali, perciò non sempre sono inviati. A questo punto il server invia il messaggio *ServerHelloDone* per indicare che la fase dei messaggi di hello è conclusa e si mette in attesa di una risposta dal client.

Se il server ha richiesto l'autenticazione del client, il primo messaggio della risposta deve coincidere con il messaggio *Certificate*. Prosegue con l'invio del messaggio *ClientKeyExchange*, il cui contenuto è dipendente dall'algoritmo a chiave pubblica selezionato nella fase di hello. Nel caso sia stato obbligato ad autenticarsi ed abbia inviato un certificato con capacità di firma, allora il client è tenuto ad inviare un messaggio *CertificateVerify* firmato digitalmente, in questo modo verifica esplicitamente di possedere la chiave privata nel certificato. La risposta si conclude con l'invio del messaggio *ChangeCipherSpec*, che porta il client a copiare la *Cipher Spec* in sospeso all'interno della *Cipher Spec* corrente, e il messaggio *Finished*, che permette di verificare che lo scambio delle chiavi e il processo di autenticazione abbiano avuto successo. In risposta, il server invia il proprio messaggio *ChangeCipherSpec*, copiando anch'esso la *Cipher Spec* in sospeso all'interno della *Cipher Spec* corrente, e il messaggio *Finished* che conclude l'handshake. A questo punto il client e il server possono iniziare scambiarsi dati applicazione in sicurezza.

Nel caso in cui si voglia recuperare una sessione precedente, il client inizia l'handshake indicando all'interno del *ClientHello* l'identificativo della sessione da recuperare. Il server a quel punto controlla la sua cache delle sessioni e, se trova la precedente sessione con il client, risponde con

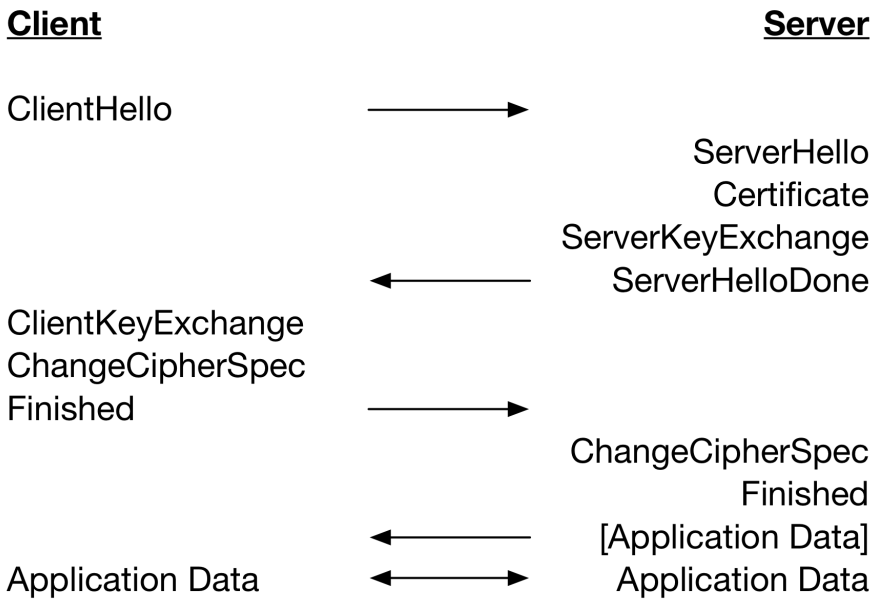

Figura 1.3. Flusso di messaggi per un handshake TLS 1.2 completo.

il messaggio *ServerHello* indicando il medesimo identificativo. Il client e il server devono inviare immediatamente i loro messaggi *ChangeCipherSpec* e *Finished* e poi possono iniziare ad inviare dati applicazione. Ciò che è stato descritto è un tipico esempio di handshake abbreviato, che può essere utilizzato in tutti i quei casi in cui sia il client sia il server intendono recuperare una precedente sessione. Tuttavia, nel caso in cui il server non riesca a trovare l'identificativo della sessione all'interno della sua cache, è necessario procedere ad un nuovo handshake completo senza poter usufruire della sua versione abbreviata.

Con l'aggiornamento all'ultima versione di TLS, il protocollo di handshake ha subito delle modifiche volte a renderlo più veloce e più sicuro. In breve, il protocollo di handshake delle versioni precedenti a TLS 1.3 richiede due "round trip" (letteralmente "andata e ritorno") prima della conclusione dell'handshake e la possibilità di scambiare dati applicazione. Nell'ultima versione invece è necessario un unico "round trip" per concludere l'handshake, che a questo punto può essere suddiviso in sole tre fasi differenti:

Scambio delle chiavi: viene stabilito il materiale condiviso per le chiavi e vengono selezionati i parametri crittografici. Qualunque messaggio dopo questa fase è crittografato.

Parametri del server: vengono stabiliti gli altri parametri dell'handshake (se il client è autenticato, il supporto del protocollo a livello applicazione, ...)

Autenticazione: autenticazione del server (e, opzionalmente, del client) e viene fornita la conferma delle chiavi e l'integrità dell'handshake.

La prima fase comincia, esattamente come prima, con l'invio da parte del client del messaggio di *ClientHello*. Questo messaggio conterrà un valore casuale *ClientHello.random*, la sua versione del protocollo offerta, una lista di coppie cipher simmetriche/hash HDKF e poi o un insieme di chiavi Diffie-Hellman condivise, o un insieme di etichette di chiavi pre-condivise, o entrambe. Il server processerà questo messaggio e determinerà i parametri crittografici per la connessione; al che, risponderà con il messaggio *ServerHello*, indicando i parametri. La combinazione dei due messaggi di hello determina le chiavi condivise. Il messaggio di hello del server può essere accompagnato da due estensioni: la prima, *key_share*, nel caso in cui siano in uso le chiavi (EC)DHE, conterrà la condivisione DH effimera del server; la seconda, *pre_shared_key*, nel caso in cui siano in uso le chiavi PSK, indicando quale delle PSK offerte dal client è stata scelta.

Successivamente il server invia due messaggi per stabilire i parametri del server: il messaggio *EncryptedExtensions* contenente le risposte alle estensioni *ClientHello* non necessarie, al fine di determinare i parametri crittografici, diversi da quelli specifici dei singoli certificati, e il messaggio *CertificateRequest* per richiedere l'autenticazione del client.

Infine, il client e il server si scambiano i messaggi di autenticazione *Certificate*, *CertificateVerify* e *Finished* che mantengono le stesse funzioni spiegate precedentemente. A questo punto, l'handshake è completo e le due parti possono derivare il materiale per le chiavi richiesto dal livello record per poter scambiare dati applicazione protetti da cifratura autenticata. Sebbene i dati applicazioni non possano essere inviati prima del messaggio *Finished*, il server può inviarli prima di ricevere i messaggi di autenticazione del client; questo vuol dire che qualunque dato inviato in quel punto, sarà inviato ad una parte non autenticata.

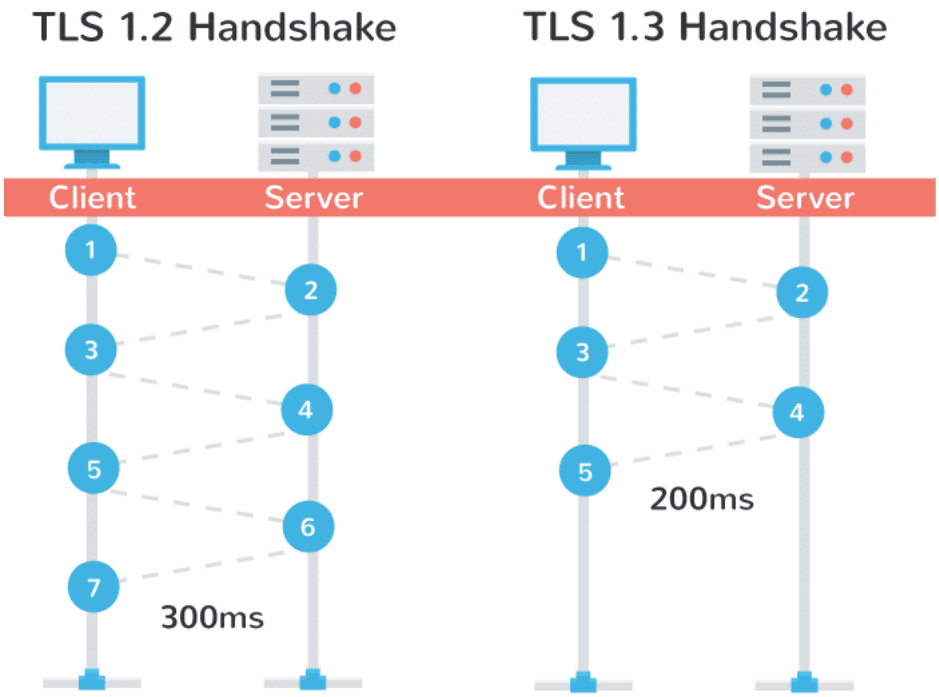

Figura 1.4. Confronto tra un handshake TLS 1.2 e un handshake TLS 1.3.

Come è possibile notare anche dalla Fig. 1.4, l'handshake nella versione 1.3 di TLS aiuta le connessioni crittografate ad essere più veloci e dimezza anche la latenza. Inoltre, capire il funzionamento di questo processo e le modifiche che sono state fatte in TLS 1.3, ci aiuterà a comprendere meglio i meccanismi utilizzati negli attacchi al protocollo di handshake, come l'attacco *Early CCS* o l'attacco *Early Finished*.

1.1.4 Protocollo di registrazione

Il protocollo di registrazione di TLS (in inglese "TLS Record Protocol") fornisce protezione ai dati dell'applicazione mediante l'utilizzo delle chiavi create durante la fase di handshake. Questo protocollo è perciò responsabile della protezione e della verifica dell'integrità e dell'origine di questi record, gestendo ciò che segue:

- suddivisione dei messaggi in uscita in blocchi gestibili e riassemblamento dei messaggi in arrivo;
- applicazione di un *Message Authentication Code* (MAC) ai messaggi in uscita e verifica dei messaggi in arrivo mediante MAC;

- crittografia dei messaggi in uscita e decrittografia dei messaggio in arrivo;
- compressione dei blocchi in uscita e decompressione dei blocchi in ingresso [opzionale].

A meno che non ne vengano negoziati di nuovi mediante estensioni, il contenuto dei record può essere di quattro tipi differenti: *handshake*, *application_data*, *alert* e *change_cipher_spec*. Se un'implementazione riceve un tipo di record inaspettato, è obbligata a terminare la connessione con un avviso *unexpected_message*. Lo stesso messaggio deve essere utilizzato per terminare la connessione nel caso in cui venga ricevuto un record *change_cipher_spec* protetto oppure con un qualsiasi valore differente da 0x01. Infatti, questa tipologia di record non deve essere cifrata, deve contenere un singolo byte di valore 0x01 e deve esser ricevuta in qualunque momento dopo l'invio o la ricezione del primo messaggio *ClientHello* e prima della ricezione del messaggio *Finished*. Nel caso non rispettasse una qualunque di queste regole, l'implementazione deve trattare il messaggio come un tipo di record inaspettato e, perciò, lanciare un avviso *unexpected_message*.

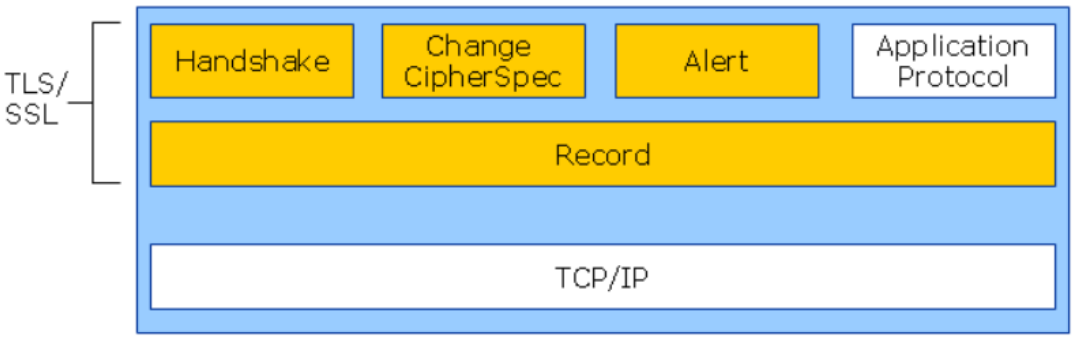

Figura 1.5. Composizione del protocollo TLS.

Il livello *record* si occupa della frammentazione dei blocchi di informazioni in record chiamati *TLSPlaintext*, i quali conterranno i dati in dimensione massima di 2^{14} byte. I limiti del messaggio sono gestiti in modo differente in base al tipo di contenuto dei messaggi. Ad esempio, i messaggi *handshake* possono essere riuniti in un singolo record *TLSPlaintext* o frammentati tra record separati, a condizione che venga rispettato ciò che segue:

- i messaggi *handshake* non devono essere interlacciati con altri tipi di record. Ovvero, se un messaggio di handshake è suddiviso tra due o più record, non possono esserci altri record tra di loro;
- i messaggi *handshake* non devono subire la modifica delle chiavi. Le implementazioni devono verificare che tutti i messaggi che precedono un cambiamento di chiave siano allineati con un limite di record; in caso contrario, sono obbligate a terminare la connessione con un avviso *unexpected_message*.

Inoltre, le implementazioni non devono inviare frammenti di tipo *handshake* che abbiano una lunghezza pari a zero, anche se quei frammenti contengono padding. Invece, i messaggi *alert* non possono essere frammentati tra i record e nemmeno possono essere riuniti più messaggi in un singolo record *TLSPlaintext*. Perciò, ogni record con contenuto di tipo *alert* deve contenere esattamente un solo messaggio. Al contrario, i messaggi *application_data* possono essere suddivisi attraverso record multipli o riuniti in un record singolo. Tuttavia, questi messaggi sono sempre protetti e quindi il loro contenuto risulta essere opaco a TLS. É inoltre possibile inviare anche frammenti di questo tipo con lunghezza pari a zero, poiché potrebbe essere potenzialmente utile come contromisura alle analisi del traffico.

Le funzioni di protezione dei record traducono le strutture *TLSPlaintext* in strutture *TLSCiphertext*, mentre le funzioni di deprotezione eseguono il processo inverso. A differenza di tutte le versioni precedenti, in TLS 1.3 tutte le cifrature sono modellate come "Authenticated Encryption with Associated Data" (AEAD), le cui funzioni forniscono un'operazione di crittografia e

autenticazione unificata che trasforma il testo in chiaro in testo cifrato e autenticato, e viceversa. Ogni record crittografato consiste in un *header* di testo in chiaro seguito da un *body* crittografato, contenente a sua volta un tipo e del padding opzionale.

Gli algoritmi AEAD ricevono come input una chiave singola, un numero casuale, il testo in chiaro e un "dato addizionale" da includere nel controllo dell'autenticazione. La chiave può essere sia la *client_write_key* sia la *server_write_key*, mentre il numero casuale viene generato a partire dal numero di sequenza e dal *client_write_iv* o dal *server_write_iv*. Infine, il "dato addizionale" di input è l'header del record. Per effettuare la decifrazione e la verifica, vengono presi come input, la chiave, il numero casuale, il dato addizionale e il valore *AEADEncrypted*. Il risultato prodotto sarà o il testo in chiaro corretto oppure un errore che indica che la decifrazione è fallita; in quest'ultimo caso, il ricevente deve terminare la connessione con un avviso *bad_record_mac*.

Un'ultima funzione che ci fornisce questo protocollo è la possibilità di aggiungere del padding a tutti i record TLS cifrati, in modo da aumentare la dimensione del *TLSCiphertext* e nascondere la dimensione del traffico ad un eventuale osservatore. Quando viene generato un record *TLSCiphertext*, l'implementazione può decidere se utilizzare il padding oppure no. Nel caso in cui decida di utilizzarlo, è obbligata ad inserirlo prima che avvenga l'operazione di cifratura. Il padding è semplicemente una stringa di byte dal valore uguale a zero aggiunti al campo *Content-Type*. In fase di decifrazione, il padding inviato viene verificato automaticamente dal meccanismo di protezione dei record e, una volta effettuata l'operazione di decifrazione, l'implementazione scansiona l'apposito campo a partire dal fondo fino a quando non viene trovato un ottetto diverso da zero; quell'ottetto è il *ContentType*. Nel caso in cui un'implementazione non riesca a trovare un ottetto diverso da zero nel testo in chiaro, allora deve terminare la connessione inviando un avviso *unexpected_message*.

1.1.5 Protocollo di alert

Il protocollo di alert permette a TLS di utilizzare un tipo di contenuto per i messaggi, chiamato "Alert", per indicare informazioni riguardo la chiusura della connessione e gli errori. Come altri messaggi, anche i messaggi di alert sono crittografati come specificato dallo stato di connessione corrente. Questi messaggi di alert trasportano una descrizione dell'avviso e, al loro interno, contengono un campo "legacy" che, nelle versioni di TLS precedenti a 1.3, indica il livello di sicurezza del messaggio. In TLS 1.3 questo campo può essere ignorato, poiché il livello di severità del messaggio è implicito nel tipo di avviso trasportato. Gli avvisi sono divisi in due categorie: gli avvisi di chiusura e gli avvisi di errore.

Avvisi di chiusura

Il messaggio *close_notify* viene utilizzato per indicare una chiusura ordinata di una connessione in una direzione. Dopo aver ricevuto un messaggio di questo tipo, l'implementazione TLS deve indicare all'applicazione la fine dei dati. Il client e il server devono condividere tra di loro le informazioni riguardo ad una chiusura della connessione, in modo da evitare che venga messo in atto un "truncation attack" [10]. In realtà, esistono due diversi avvisi di chiusura leggermente differenti l'uno dall'altro:

***close_notify*:** il ricevente viene avvisato che il mittente non invierà più alcun dato sulla connessione. Qualsiasi dato ricevuto dopo questo messaggio deve essere ignorato.

***user_canceled*:** il ricevente viene avvisato che il mittente sta annullando l'handshake per qualche ragione slegata dagli errori di protocollo. Questo messaggio deve comunque essere seguito da un messaggio *close_notify*. Il livello di allerta di questo messaggio generalmente è "warning".

Nel caso una delle due parti voglia iniziare la chiusura del suo lato di connessione relativo alla scrittura dei dati, deve prima inviare un messaggio *close_notify* all'altra parte. Questo può non succedere nel caso in cui, la parte che vuole chiudere la connessione, abbia già inviato in precedenza un messaggio di errore. Tuttavia, l'invio di questo messaggio non ha effetto sul proprio lato di connessione relativo alla lettura dei dati. Inoltre, se una parte riceve una chiusura a livello

trasporto prima di ricevere un messaggio *close_notify*, allora non potrà sapere se tutti i dati che gli sono stati inviati sono anche stati ricevuti. Questo comportamento è cambiato rispetto alle versioni di TLS precedenti alla 1.3, nelle quali le implementazioni TLS dovevano reagire ad una *close_notify* scartando tutte le scritture pendenti e inviando a loro volta un messaggio *close_notify*. Questo comportamento poteva causare un troncamento nel lato relativo alla lettura dei dati. Ora, nella versione TLS 1.3, entrambe le parti non devono più attendere che venga ricevuta una *close_notify* prima di chiudere il loro lato di lettura, sebbene questo introduca la possibilità di troncamento.

Avvisi di errore

I messaggi di errore, invece, indicano una chiusura della connessione non corretta. Dopo aver ricevuto un messaggio di questo tipo, l'implementazione TLS deve indicare un errore all'applicazione e non deve permettere che alcun dato sia più ricevuto o inviato sulla connessione. Inoltre, il server e il client devono dimenticare i valori dei segreti e delle chiavi stabilite nella connessione che è fallita, ad eccezione delle PSK associate ai ticket di sessione, le quali devono essere scartate, se possibile.

La gestione degli errori in TLS è molto semplice, quando viene rilevato un errore, la parte che ne ha rilevato la presenza deve inviare un messaggio all'altra parte. In seguito alla ricezione o alla trasmissione di un messaggio di avviso con livello di allerta uguale a "fatal", entrambe le parti devono chiudere immediatamente la connessione senza inviare o ricevere più altri dati. Inoltre, tutti gli avvisi di tipo sconosciuto devono essere considerati come messaggi di errore. Gli avvisi che elencheremo qui di seguito, solo una parte della totalità di quelli esistenti, hanno tutti un livello di allerta uguale a "fatal" e devono essere trattati come messaggi di errore quando vengono ricevuti, indipendentemente dal livello di allerta indicato al loro interno:

***unexpected_message*:** indica che è stato ricevuto un messaggio non appropriato (es. il messaggio di hanshake sbagliato, dati applicazione prematuri, etc...).

***handshake_failure*:** indica che il mittente non è in grado di negoziare un insieme accettabile di parametri di sicurezza, in base alle opzioni disponibili.

***bad_certificate*:** indica che un certificato è stato corrotto, contiene una firma non verificata correttamente o altro.

***certificate_revoked, certificate_expired, certificate_unknown*:** indicano rispettivamente che un certificato è stato revocato, è scaduto oppure è sorto qualsiasi altro problema nella gestione del certificato che l'ha reso inaccettabile.

***unknown_ca*:** indica che il certificato non è stato accettato perché non è stato possibile individuare l'autorità di certificazione.

***decode_error, decrypt_error*:** indicano rispettivamente che un messaggio non può essere decodificato, a causa di qualche campo fuori posto o la lunghezza del messaggio non corretta, oppure che un'operazione crittografica dell'handshake è fallita, come la verifica della firma o la validazione del messaggio *Finished*.

***protocol_version*:** indica che la versione del protocollo che la parte ha provato a negoziare è riconosciuta ma non supportata.

***insufficient_security*:** viene ritornato questo messaggio invece di *handshake_failure* quando una negoziazione è specificatamente fallita perché il server richiede parametri più sicuri di quelli supportati dal client.

***missing_extension, unsupported_extension*:** indicano rispettivamente la ricezione di un messaggio di handshake senza un'estensione considerata obbligatoria o contenente un'estensione considerata proibita.

***bad_certificate_status_response*:** inviata dal client quando viene fornita una risposta OCSP non valida o inaccettabile dal server, tramite l'estensione *status_request*.

***unknown_psk_identity*:** inviata dal server quando le PSK sono richieste ma il client non ha fornito un'identità PSK accettabile.

***certificate_required*:** inviata dal server quando il certificato del client è richiesto ma non è stato fornito.

1.2 Certificati X.509

Un certificato X.509 è un particolare documento digitale che viene utilizzato per legare l'identità assoluta del richiedente con la sua chiave pubblica ed è implementato in modo da essere interpretato automaticamente dai computer. L'infrastruttura a chiave pubblica definisce un insieme di tecnologie e di politiche per la creazione e l'uso di questi certificati digitali. L'efficacia di questi sistemi è data dal lavoro cooperativo tra la crittografia a chiave pubblica e questo insieme di politiche, che devono essere create e gestite con molta attenzione

La maggior parte dei servizi Internet effettuano uno scambio di questi certificati che sono emessi da un'autorità di certificazione, in modo da potersi scambiare le chiavi pubbliche. Per definire il formato standard di tali certificati e le norme che devono essere seguite dalle varie componenti che costituiscono l'infrastruttura generale di certificazione, nel 1988 è stato presentato al mondo dal ITU-T lo standard internazionale X.509, come parte dello standard X.500. Questo standard, dal 1996 in versione 3, viene utilizzato per la certificazione delle chiavi pubbliche. Questa certificazione avviene mediante la codifica della chiave pubblica del richiedente con la chiave privata della autorità di certificazione (CA). Tuttavia, lo standard non impone l'uso di un particolare algoritmo simmetrico per effettuare questa codifica, sebbene sia ampiamente utilizzato RSA. Inoltre, lo standard prevede anche l'uso di un digest, in modo da produrre una firma dell'intero certificato, ma anche in questo caso non sono previste imposizioni su quale algoritmo vada utilizzato.

Come detto, il formato dei certificati X.509 è ben definito [5], ed infatti tutti i certificati iniziano con un campo *Version*, il quale indica la versione dello standard a cui si riferisce il certificato. Il numero di versione è molto utile perché fornisce indicazioni su come debba essere effettuato il parsing dei campi successivi, poiché sono previsti campi differenti in base alla versione a cui fa riferimento il certificato. Il valore di questo campo è un numero intero compreso tra 0 e 2: il valore 0 indica la versione 1, il valore 1 la versione 2 e il valore 2 la versione 3. In questo modo, in futuro, sarà possibile utilizzare altri valori in riferimento ad altre versioni.

Il campo successivo è il *Certificate Serial Number*, un numero intero che permette di identificare univocamente il certificato all'interno della CA che lo ha emesso. Ne consegue che, secondo quanto previsto dallo standard, un'autorità di certificazione non può mai emettere due certificati che abbiano lo stesso numero seriale.

Il campo *Signature Algorithm Identifier* indica quale algoritmo è stato utilizzato per produrre la firma finale del certificato. Inoltre, in base all'algoritmo indicato in questo campo, sono necessari, e perciò presenti, uno o più parametri di inizializzazione. Questo campo è utile per identificare sia l'algoritmo di hash utilizzato per la produzione del digest (*MD5*, *SHA*, *HMAC*, etc...) sia l'algoritmo utilizzato per lo scambio delle chiavi (*RSA*, *DSS*).

Il campo *Issuer Name* contiene l'identificativo dell'autorità di certificazione che ha emesso il certificato. Lo standard X.509 prevedere l'uso di identificativi univoci anche per le CA. Questo identificativo assume la forma di un *Distinguish Name* (DN), i cui campi principali sono:

O: identifica l'organizzazione;

OU: identifica un'unità specifica interna all'organizzazione;

CN: è il *Common Name*, ovverio il nome del soggetto o dell'entità;

L: indica la località, generalmente è il nome di una città;

ST: indica lo stato o la provincia;

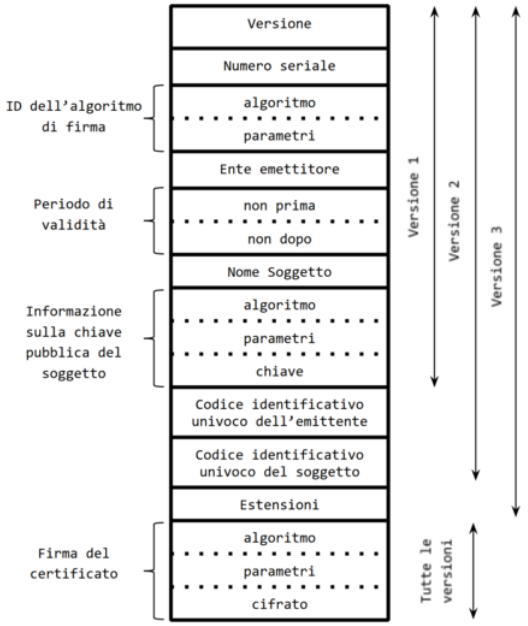

Figura 1.6. Formato di un certificato X.509.

C: indica la nazione di appartenenza del soggetto o dell'entità.

Tutti i campi di un DN sono opzionali e possono anche comparire più di una volta.

Il campo *Validity* stabilisce il periodo temporale di validità del certificato. Ad ogni certificato sono generalmente associate due date differenti. La prima indica la data a partire dalla quale il certificato risulta valido, la seconda indica la data dopo la quale il certificato non è più valido. Tuttavia, per una serie di motivi, un certificato può essere revocato dall'autorità che lo ha emesso anche prima dell'effettiva data di scadenza.

Il campo *Subject* contiene al suo interno le generalità del soggetto o dell'entità che ha fatto la richiesta del certificato. Lo standard X.509 prevede l'uso di identificativi univoci per ogni richiedente. L'identità del richiedente stabilisce, di fatto, chi è il possessore della chiave privata corrispondente alla chiave pubblica contenuta all'interno del certificato. Il formato del *Subject*, come per il campo *Issuer Name*, assume la forma di un DN, ad esempio:

- *C=Italy, O=Polito, OU=DAUIN, CN=Davide Giorda*

Il valore del *CN* può variare in base all'entità che ha richiesto il certificato o allo scopo per cui il certificato viene rilasciato. Tuttavia, lo standard X.509 non pone alcun vincolo sul valore che deve essere inserito nel *Common Name*. Perciò, nel caso di persona fisica, il *CN* può essere il suo nome e cognome, mentre nel caso di un server web può essere il suo nome DNS. In quest'ultimo caso, il compromesso raggiunto oramai a livello globale è quello di inserire il *Domain Name* all'interno del *CN* del *Subject*, in questo modo il client ha la possibilità di confrontarlo con il nome del dominio a cui si sta collegando. Inoltre, i certificati X.509 sono abbastanza costosi da ottenere, o per lo meno lo sono quelli che vengono emessi da autorità di certificazione affidabili. Questo ha fatto sorgere alcuni problemi. Prendiamo, come esempio, un sito di e-commerce che controlla tre domini differenti: *orders.site.com*, *shop.site.com* e *purchase.site.com*. L'amministratore dovrebbe richiedere un certificato per ognuno di questi tre domini, dovendo tenere anche traccia di tutte le date di scadenza, in modo da garantirsi che i certificati vengano emessi nuovamente in un breve periodo di tempo. Di conseguenza, l'amministratore di *site.com* preferirebbe dover richiedere un unico certificato che autentichi tutti i server del suo dominio (anche quelli virtuali). Per questo motivo, all'interno del *CN* del *Subejct* del certificato, viene consentito l'utilizzo del carattere "*", in questo modo l'amministratore può richiedere un unico certificato per il dominio **.site.com*.

Questo porta ad altri problemi, tra cui la possibilità di essere vittime di un attacco Man-In-The-Middle, una tipologia di attacco che vedremo più avanti. Nel 1998, lo standard X.509 estende il *Subject Name*, introducendo la possibilità di esplicitamente le componenti di un *Domain Name* suddivise in base alla gerarchia DNS. Tuttavia, questa estensione non è ancora particolarmente utilizzata e la maggior parte dei siti continua ad utilizzare il campo *CN* per identificare il proprio dominio.

Il campo *Subject Public Key* contiene tutte le informazioni relative alla chiave pubblica del soggetto che ha richiesto il certificato. Questo campo fornisce la possibilità di stabilire quale sia l'algoritmo asimmetrico a cui la chiave fa riferimento e gli eventuali parametri di inizializzazione aggiuntivi, oltre a contenere una sequenza binaria corrispondente alla chiave pubblica.

L'ultimo campo presente all'interno del certificato è il campo *Signature*, il quale contiene la versione firmata del risultato ottenuto dalla funzione di hashing prestabilita, applicata all'intero corpo del certificato. Questa firma viene effettuata mediante l'utilizzo della chiave privata della CA; così facendo, la chiave pubblica del soggetto richiedente viene trasmessa in chiaro in ogni certificato e può essere estratta dal messaggio da qualsiasi utente, anche senza essere in possesso della chiave pubblica della CA che ha emesso il certificato. Questa firma ha lo scopo di garantire l'integrità e l'autenticità del certificato emesso. Se un utente vuole verificare l'originalità della chiave pubblica che si trova all'interno del certificato, deve procedere in questo modo:

1. deve procurarsi la chiave pubblica dell'autorità di certificazione che ha emesso il certificato;
2. deve decodificare la firma del certificato;
3. deve applicare la funzione di hash al corpo del certificato;
4. deve confrontare i digest precedenti per stabilire l'autenticità della chiave pubblica.

A differenza dello standard X.509v1, gli standard X.509v2 e X.509v3 prevedono due ulteriori campi chiamati *Issuer Unique Identifier* e *Subject Unique Identifier*, i quali sono inseriti immediatamente prima della firma finale, in modo da consentire l'utilizzo dei certificati anche quando i *DN* del soggetto richiedente e della CA vengono usati per identificare molteplici entità, ovvero quando gli identificativi non sono univoci.

Nonostante ogni certificato abbia una validità temporale limitata, può accadere che sia necessario revocare uno o più certificati in anticipo rispetto alla data di scadenza. La revoca del certificato può avvenire solamente per mano della CA che lo ha emesso. Per questo motivo, lo standard X.509 prevede che ogni autorità di certificazione mantenga una lista aggiornata dei certificati revocati. Siccome ogni certificato è identificato in modo univoco attraverso un numero seriale assegnatogli dalla CA, è sufficiente che, per ogni certificato revocato, all'interno della lista venga indicato il numero di serie del certificato e la data di revoca. Tuttavia, è responsabilità degli utenti richiedere periodicamente, alle diverse CA, una lista aggiornata dei certificati revocati. Inoltre, per garantire che i messaggi contenenti questa lista non vengano alterati durante la trasmissione, è presente una firma della lista stessa.

Il sistema dei certificati X.509 introduce dei benefici reali solamente se non rappresenta una decisione presa da un singolo server o da un singolo amministratore. I benefici effettivi di un sistema efficiente di firma digitale e crittografia asimmetrica si ottengono se i certificati X.509 vengono considerati come parte integrante nella realizzazione di una evoluta infrastruttura per lo scambio delle chiavi pubbliche come la "Public Key Infrastructure" (PKI). Una PKI oltre ad includere la autorità di certificazione, gli utenti che la ritengono fidata e i certificati X.509 che vengono scambiati tra le parti, prevede l'attivazione di una o più "Registration Authority" (RA). Le autorità di registrazione sono quelle entità responsabili di intrattenere i rapporti con gli utenti finali ed i loro sistemi, sia da un punto di vista informatico sia fisicamente. Una RA ha diverse responsabilità, tra cui quella di distribuire in assoluta sicurezza il certificato contenente la chiave pubblica della CA, verificare l'identità di colui che richiede un certificato prima di autorizzarne l'emissione ed, infine, regolare il rapporto con gli utenti finali dal punto di vista amministrativo e contrattuale. L'attivazione di più PKI mutuamente compatibili, che siano in grado di gestire i diversi servizi Internet, richiede che prima venga realizzata un'architettura globale che possa regolare i rapporti di fiducia tra le varie CA. Lo standard implementa una struttura gerarchica a

più livelli, in cui ogni autorità di certificazione è responsabile della certificazione sia degli utenti finali sia delle altre CA. Per entrare a far parte di questa infrastruttura, ogni autorità locale deve procurarsi un certificato che sia valido per la propria chiave pubblica e che sia emesso dalla CA con cui ha un rapporto diretto di parentela. L'architettura X.509 prevede che le CA di più alto livello si scambino mutuamente i certificati riguardanti le proprie chiavi pubbliche. Successivamente questi certificati vengono memorizzati da ogni CA all'interno di una directory consultabile pubblicamente.

Il processo che risulta essere più critico, e che spesso è sottovalutato dagli utenti, è la validazione di un certificato. Infatti sono comprese le seguenti operazioni:

- il controllo crittografico della firma del certificato;
- il controllo delle varie estensioni e dei campi del certificato;
- il controllo dello stato di revoca del certificato.

Poiché le CA sono all'interno di una struttura gerarchica, per validare il certificato dell'utente finale è spesso necessario effettuare la verifica di tutti i certificati della catena. Perciò la validazione di una catena di certificati implica che, potenzialmente, è necessario processare iterativamente ogni certificato fino a raggiungere quello della *Root CA*. Se durante questo processo, anche solo uno dei certificati della catena risulta non valido, allora l'applicazione non deve permettere di autenticare il soggetto. Inoltre, l'aumentare del numero di autorità intermedie aumenta anche il numero possibile di punti da cui si può essere attaccati. L'intero meccanismo dei certificati X.509 si basa sulla fiducia che gli utenti ripongono nelle *Certification Authority*.

1.2.1 Autorità di certificazione

L'ente incaricato del rilascio di un certificato digitale viene definito *Certification Authority* (CA, tradotto "autorità di certificazione"). Si tratta di una organizzazione di terza parte, pubblica o privata, abilitata al rilascio dei certificati digitali, in modo che garantiscano l'associazione biunivoca fra un particolare utente e una coppia di chiavi, rendendo disponibile la chiave pubblica. Questo ente conferma anche l'autenticità dei certificati che vengono rilasciati, assicurando che l'identità contenuta all'interno del certificato corrisponda realmente al possessore della chiave pubblica associata. All'interno del certificato, la CA inserisce i dati identificativi dell'utente, la sua chiave pubblica e la firma della CA stessa, quest'ultima serve ad assicurarne l'autenticità ed impedirne un eventuale manomissione. Per poter eseguire tutte queste operazioni, a sua volta anche la CA deve essere in possesso di un certificato che la associ ad una coppia di chiavi e che sia rilasciato da un'autorità superiore oppure dalla stessa. L'utente finale non si limita a presumere che il certificato ricevuto sia autentico, ma ne controlla l'autenticità presso la CA che lo ha rilasciato, verificando che questa sia realmente affidabile. Nel caso non sia possibile autenticare l'identità dell'autorità di certificazione, ne consegue che non è possibile nemmeno autenticare l'identità dell'utente o del computer remoto con cui sta avvenendo la conversazione.

Generalmente le autorità di certificazione sono organizzate in modo gerarchico. In questa struttura, ogni CA madre è chiamata *Root CA* ed ha la possibilità di firmare un certificato relativo ad una CA di livello gerarchico più basso, chiamata *Intermediate CA*, garantendone l'affidabilità. La creazione di questa struttura gerarchica permette alle CA di delegare il processo di identificazione ad altre autorità, il che è particolarmente utile per tutte quelle CA che operano su scala globale, poiché gli viene fornita la possibilità di distribuire il carico di lavoro sugli enti locali, i quali risultano anche più competenti nelle procedure di identificazione. I certificati intermedi non devono necessariamente identificare altre CA, ma possono essere utilizzati anche per motivi di sicurezza, permettendo ad una CA di firmare un certificato intermedio con il proprio certificato principale. Perciò, il certificato intermedio viene utilizzato per l'emissione di nuovi certificati ed è possibile che venga sostituito o revocato in qualsiasi momento, mentre il *Root Certificate* viene mantenuto al sicuro offline, al di fuori della rete. Un ultimo possibile utilizzo dei certificati intermedi è per la firma di altri certificati intermedi, in modo da creare la cosiddetta "catena di certificazione". Quando risulta necessario validare una catena di certificati, è necessario partire

dal basso e validare ogni certificato tramite quello che lo precede nella catena, fino a raggiungere l'estremo superiore. Ogni CA di livello differente dal primo, oltre a possedere una certificazione della propria chiave emessa da una CA superiore, deve mantenere un certificato inverso che certifichi la chiave pubblica della CA di livello più alto. Questi certificati inversi permettono ad ogni entità, utente o CA, di risalire la struttura gerarchica e di verificare la chiave pubblica degli altri partecipanti. In generale, non esiste nessun modo automatico per avere una garanzia sicura dell'affidabilità delle autorità di certificazione intermedie, così come di quella della *Root CA*. Inoltre, se il numero di autorità intermedie e dei relativi certificati aumenta, di conseguenza aumenta anche il numero dei possibili punti da cui è possibile subire un attacco.

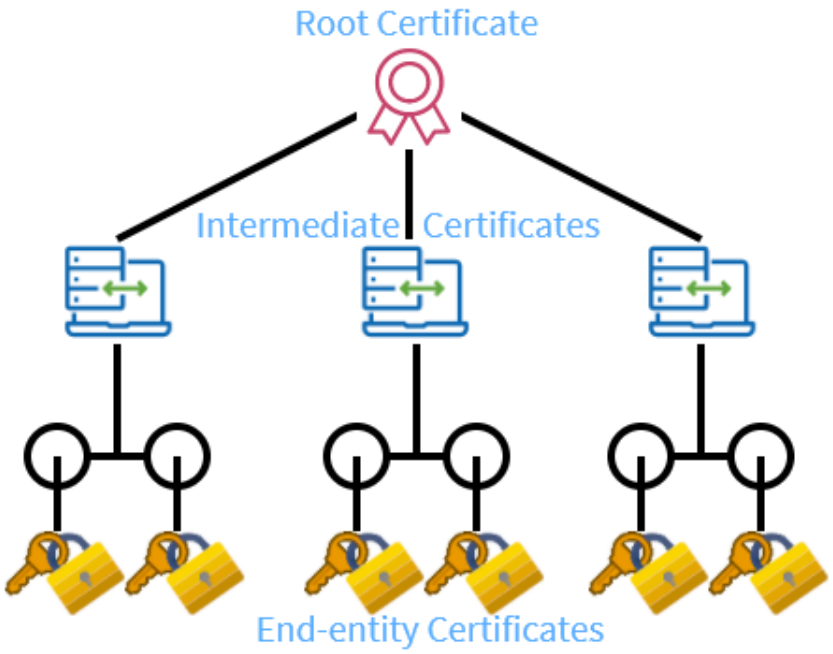

Figura 1.7. Struttura gerarchica della catena di certificazione.

Qualsiasi certificato appartenente ad una *Root CA* viene auto-firmato, ovvero la chiave pubblica è auto certificata mediante la corrispondente chiave privata della stessa CA a livello root. Questi certificati root sono generalmente distribuiti insieme ai browser o al sistema operativo, perciò non possono essere facilmente aggiornati, portando la gestione delle chiavi root a seguire l'evoluzione delle versioni e delle distribuzione degli stessi browser. L'unica possibilità per un utente di mantenere aggiornati i *Root Certificate* è quella di tenere i browser allineati alle versioni più recenti. In più, dal momento che risulterebbe impossibile modificare la moltitudine di copie di chiavi sui browser sparsi in giro per il mondo, è praticamente impossibile effettuare la revoca di un certificato root. Proprio per questo motivo le *Root CA* hanno un'organizzazione estremamente solida e sicura dal punto di vista degli attacchi informatici.

1.2.2 Estensioni dei certificati

Per poter associare ulteriori attributi alle chiavi pubbliche sono state introdotte le estensioni ai certificati. Inoltre, l'ultima versione dello standard X.509 permette alle aziende di definire delle estensioni private, in modo da poter aggiungere informazioni particolari. Ogni estensione può essere definita "critica" o "non critica". Un qualsiasi sistema che utilizza i certificati X.509 può ignorare un'estensione non critica che non viene riconosciuta, mentre se a non essere riconosciuta è un'estensione critica allora il sistema deve rigettare il certificato. É perciò importante fare molta attenzione quando si adottano estensioni critiche nei certificati, poiché potrebbero sorgere problemi nel loro utilizzo. Le principali estensioni standard dei certificati X.509 sono le seguenti:

Authority Key Identifier: viene utilizzata quando chi ha emesso il certificato possiede più di una chiave per la firma. Questa estensione risulta essere un mezzo per identificare la chiave pubblica corrispondente alla chiave privata che è stata utilizzata per la firma. L'identificazione può basarsi sia sull'identificatore della chiave, sia sul nome della CA e sia sul numero di serie.

Basic Constraints: in questa estensione viene specificato se il soggetto del certificato è un'autorità di certificazione ed, eventualmente, viene indicata la lunghezza della catena di certificati che può sussistere attraverso questa CA. Se questa estensione è presente, il campo *pathLenConstraint* al suo interno (il quale indica il numero massimo di certificati CA che formano la "certificate chain"), deve esser maggiore o uguale a zero. Se invece non è presente, non ci sono limiti alla lunghezza della catena di certificati.

Certificate Policy: contiene la sequenza delle politiche in base a cui il certificato è stato emesso e i propositi per i quali può essere utilizzato.

CRL Distribution Points: specifica dove si possono ottenere le informazioni riguardo le Certificate Revocation List (CRL). Può essere un indirizzo e-mail, una URL o una entry di una directory.

Extendend Key Usage: al suo interno vengono indicati uno o più scopi per i quali la chiave pubblica può essere utilizzata, in aggiunta o al posto di quelli base indicati nell'estensione *Key Usage*.

Issuer Alternative Name: utilizzata per associare le rispettive identità, che possono essere più di una, a chi effettua l'emissione del certificato.

Key Usage: permette di definire lo scopo della chiave contenuta nel certificato. É possibile utilizzare questa estensione quando è necessario limitare l'utilizzo di una chiave. Ad esempio, una chiave pubblica può essere utilizzata per firmare certificati, per criptare le chiavi o per lo scambio delle chiavi. Non è vietato utilizzare la chiave per scopi differenti da quelli specificati nel certificato, ma è compito di chi verifica quello di accertarsi che sia rispettato il "key usage", bloccando le operazioni non consentite.

Name Constraints: è possibile utilizzarla solamente in un certificato CA ed indica le limitazioni a cui sono sottoposti tutti i nomi dei soggetti nei certificati successivi all'interno della catena.

Policy Contraints: può essere utilizzata nei certificati emessi per una CA, in modo da poter restringere la validazione della catena di certificati.

Private Key Usage Period: permette a chi emette il certificato di specificare un periodo di validità per la chiave privata, differente da quello del certificato.

Subject Alternative Name (SAN): fornisce la possibilità di collegare identità aggiuntive al soggetto del certificato. In questa estensione vengono inserite tutte le informazioni che non rientrano nel *DN* del campo *Subject*, ad esempio:

- Directory Name
- Domain Name System
- Indirizzi IP
- Indirizzi di posta elettronica
- Indirizzi X.400
- Uniform Resource Identifier (URI)

Esistono altre opzioni, incluse quelle completamente locali. É possibile utilizzarla tutte le volte in cui un certificato deve includere host con nomi multipli o host con istanze multiple per ogni nome. Questa estensione è sempre critica se il *Subject Name* è vuoto; inoltre, le CA devono verificarne tutte le parti, poiché il *Subject Alternative Name* è legato alla chiave pubblica.

Subject Key Identifier: viene utilizzata per l'identificazione di un utente che ha ottenuto più certificati da più di una singola CA. L'identificazione avviene mediante la sua chiave pubblica. Inoltre, per facilitare la costruzione della catena dei certificati, questa estensione deve essere presente in tutti i certificati che sono conformi allo standard.

1.2.3 CRL e OCSP

I certificati digitali possiedono una data di scadenza, proprio come qualsiasi documento d'identità cartaceo. Per conoscere se un certificato risulta ancora valido, è necessario verificarne il termine temporale. Tuttavia, anche se il certificato risulta ancora temporalmente valido, potrebbe essere stato revocato per un altro motivo, come:

- la chiave segreta e il certificato dell'utente sono stati compromessi;
- l'utente non è più certificato dalla CA;
- il certificato è stato emesso erroneamente.

In caso di furto, un ladro potrebbe continuare ad usare il vecchio certificato e la chiave privata per firmare qualsiasi documento, impersonando il legittimo proprietario del certificato. La data di scadenza è stata introdotta proprio per evitare questo: se il proprietario del certificato non è a conoscenza del furto, ad un certo punto il certificato scadrà e ne verrà generato uno nuovo. Invece, se il proprietario scopre che il suo certificato è stato compromesso, deve avvisare immediatamente la CA, la quale procederà a revocarlo. Esistono due meccanismi che sono stati introdotti per verificare se un certificato è stato revocato o meno:

- **Certificate Revocation List (CRL)**
- **Online Certificate Status Protocol (OCSP)**

Ogni autorità di certificazione è tenuta a pubblicare una lista dei certificati che sono stati revocati, chiamata "Certificate Revocation List". Al suo interno vengono indicati tutti i numeri di serie dei certificati che, per qualsiasi motivo, non devono essere più utilizzati, sebbene non siano ancora scaduti. É responsabilità dell'utente finale quella di controllare questa lista, in modo da poter confrontare il numero di serie di ogni certificato ricevuto con l'elenco dei numeri di serie dei certificati revocati. Spesso le CRL risultano particolarmente pesanti, dal punto di vista del download, a causa del gran numero di revoche effettuate. Perciò è possibile che, invece delle tradizionali CRL, una CA distribuisca una CRL "delta" contenente solamente i certificati revocati dopo la pubblicazione dell'ultima CRL. In questo modo potrà essere scaricata più velocemente dall'utente, il quale però non ha alcun modo di sapere quando è sicuro smettere di tenere traccia di un certificato scaduto.

L'analisi all'interno di una CRL è simile al procedimento di ricerca in un tabulato. Inoltre, il metodo di elaborazione delle CRL può richiedere alle società la configurazione dei propri client, in modo che vengano elaborate più CRL provenienti da diverse CA. Ogni CRL è firmata dalla CA e al suo interno include, oltre alla liste delle revoche effettuate, anche il proprio nome, la data in cui la lista è stata creata e la data di pubblicazione delle prossima lista.

Un'ulteriore problema che può portare nuove falle nella sicurezza è il fatto che spesso sono necessari diversi giorni prima di ricevere una notifica riguardo ad un certificato revocato. Il punto critico riguardo all'utilizzo delle CRL è la decisione sull'intervallo di pubblicazione tra una lista e la successiva: un certificato che è stato compromesso insieme alla sua chiave, deve essere revocato il prima possibile, ovvero appena scoperta la compromissione. Se questo non avviene e la CRL viene pubblicata molti giorni dopo la revoca del certificato, gli utenti non sarebbero a conoscenza del problema e potrebbero accettare messaggi firmati con la chiave corrispondente al certificato, la quale potrebbe essere usata da un intruso e non dal reale proprietario. Tuttavia, anche se la pubblicazione della CRL avvenisse in tempo reale con la revoca del certificato, non ci sarebbe, per gli utenti, la garanzia assoluta di venirne a conoscenza in tempo.

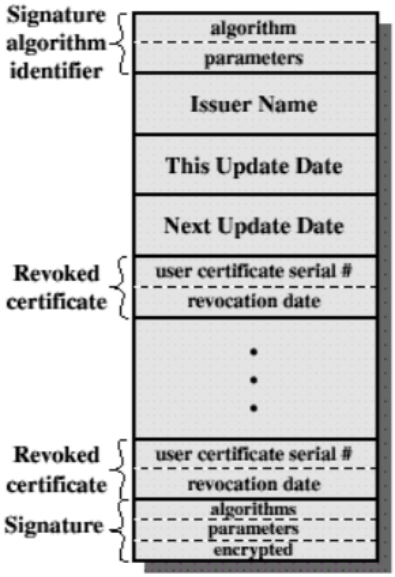

Figura 1.8. Formato di una Certificate Revocation List.

L'ultimo problema introdotto dalle CRL è la locazione di queste liste. Potrebbe sembrare ragionevole che, una volta stabilita dalla CA, questa non cambi nel tempo. In generale, l'autorità di certificazione pubblica la lista dei certificati revocati ad intervalli regolari nel suo repository pubblico. I certificati X.509 hanno un'estensione che permette di indicare dove deve essere scaricata la CRL. Tuttavia, questa estensione non consente alla CA di indicare la data in cui la locazione della CRL è cambiata. Perciò, se un client scarica due certificati firmati dalla stessa CA, contenenti due diversi URL per la lista dei certificati revocati, non avrebbe altre indicazioni per decidere quale delle due utilizzare.

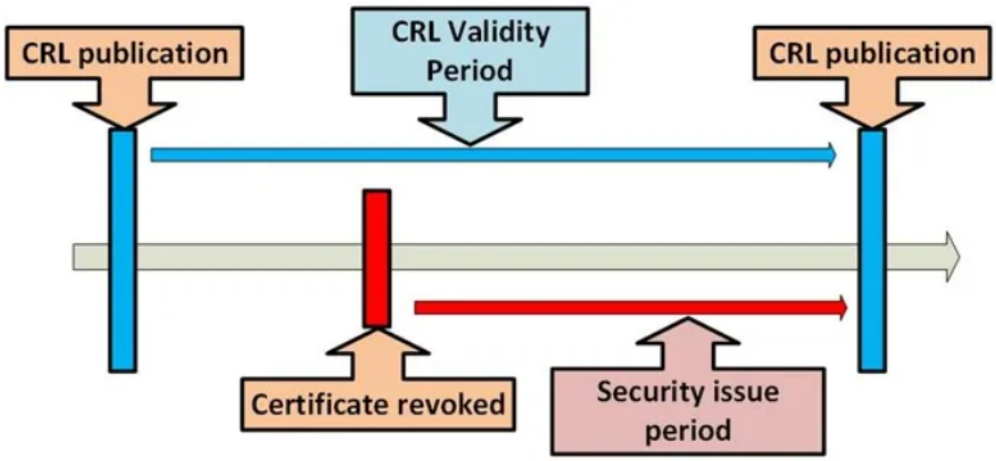

Figura 1.9. Diagramma temporale dell'emissione di un certificato.

Esistono perciò alcuni possibili problemi nell'utilizzo delle CRL come mezzo di notifica della revoca dei certificati, come i problemi di gestione e il problema dell'attualità delle informazioni di revoca. Infatti, se un chiave viene compromessa, un client vorrebbe saperlo nel più breve lasso di tempo nel caso riceva il certificato revocato. Con le liste di revoca dei certificati, il client dovrebbe scaricare l'intera CRL aggiornata, o almeno una CRL delta, ogni volta che riceve un nuovo certificato. Per questo motivo è stato sviluppato "Online Certificate Status Protocol" [6], in modo di consentire ai client di consultare lo stato di un certificato partendo semplicemente dal suo numero

di serie. OCSP è uno standard dell'Internet Engineering Task Force (IETF) e comunica online agli utenti la situazione in tempo reale dei certificati, permettendo così un maggiore velocità rispetto al sistema delle liste di revoca dei certificati e rimuovendo tutte le problematiche riguardanti la logistica, il carico di elaborazione e i tempi di notifica. Un'applicazione client di un'organizzazione permette di controllare immediatamente le revoche dei certificati semplicemente generando ed inviando una richiesta ad un OCSP responder, un server in rete su cui sono memorizzate tutte le informazioni aggiornate riguardanti le revoche. Nella richiesta che l'utente invia all'OCSP responder devono essere presenti il numero di serie del certificato e un hash dell'*Issuer Name*, questo perchè il server OCSP può gestire le revoche dei certificati di diverse CA. A questo punto il server risponderà con uno messaggio sulla validità del certificato, restituendo uno dei tre seguenti valori:

- GOOD
- REVOKED
  - Revocation Reason
  - Revocation Time
- UNKNOWN

Sebbene la richiesta OCSP sia indipendente dal protocollo utilizzato, l'approccio più comune è quello rappresentato dall'utilizzo di HTTP.

1.3 Classificazione degli attacchi

TLS è sicuramente il protocollo crittografico più diffuso nell'ambito delle telecomunicazioni e dell'informatica, poiché permette una comunicazione sicura tra due parti fornendo autenticazione, integrità dei dati e confidenzialità. Proprio a causa dell'enorme uso che ne facciamo, TLS è continuamente sotto la lente di ingrandimento di esperti di sicurezza e hacker, che studiano soluzioni per poter migliorarne la sicurezza oppure modi sempre diversi per aggirarla e scoprire nuovi attacchi contro questo protocollo. A rendere più complicato il lavoro dei ricercatori della sicurezza è il fatto che il protocollo si basa su diversi elementi e che ognuno di essi può portarsi dietro vulnerabilità differenti; in più gli stessi aggiornamenti volti ad aumentare la sicurezza di TLS possono apportare modifiche a questi elementi e mettere in mostra nuove vulnerabilità. Infatti, come già era successo con il suo predecessore SSL, anche il protocollo TLS è stato oggetto di continui attacchi sin dalla sua creazione. Come per la sicurezza, anche gli attacchi al protocollo evolvono e migliorano la propria efficacia sfruttando i punti di vulnerabilità del protocollo. Per raggruppare gli attacchi al protocollo, è possibile definire le seguenti categorie di attacchi [9]:

Attacchi alla crittografia: questa categoria comprende tutti gli attacchi che sfruttano le debolezze di uno degli algoritmi di crittografia, come RSA, MD5, DES, 3DES o RC4. La debolezza di un algoritmo è dipendente anche dall'ambito in cui l'algoritmo viene utilizzato: per la decifrazione oppure per lo scambio delle chiavi, o ancora per la firma digitale.

Attacchi alle ciphersuites: raccoglie tutti gli attacchi che traggono beneficio dalle modalità in cui viene usata la crittografia nelle ciphersuite e non direttamente dall'algoritmo utilizzato. Un esempio può essere la modalità Cipher Block Chaining o CBC che, se usata, rende la comunicazione vulnerabile ad alcune tipologie di attacco, indipendentemente dall'algoritmo di crittografia scelto.

Attacchi al protocollo TLS: questa tipologia di attacchi è quella più legata direttamente al protocollo TLS, non riguarda gli elementi che lo compongono ma proprio come si comporta e le funzionalità legate al protocollo stesso. Le funzionalità che sono colpite da attacchi sono molteplici: dal supporto per le vecchie versioni di TLS all'utilizzo delle chiavi pre-condivise (PSK), dalla compressione dei dati alla ripresa di una sessione, e molte altre ancora.

Attacchi alle implementazioni delle librerie: nel corso degli anni son state create diverse librerie per l'implementazione di SSL/TLS, tra cui ne ricordiamo alcune: OpenSSL, GnuTLS, NSS, MatrixSSL. Tuttavia, l'implementazione del protocollo all'interno di alcune librerie presentava degli errori o delle mancanze che hanno portato alla creazione di nuovi attacchi contro queste librerie. É perciò possibile che alcuni attacchi funzionino solamente contro una libreria e non contro altre, la cui implementazione di TLS è priva di un determinato errore.

Ognuna di queste categorie prevede attacchi diversi ma, nonostante ciò, ognuna può essere influenzata dall'altra. Le contromisure che possono essere prese per sfuggire a questi attacchi sono diverse per ciascuna categoria; se per gli attacchi alle librerie è sufficiente una patch per risolvere l'errore e far scomparire la vulnerabilità, per le altre categorie non è sufficiente, è necessario rivedere l'intera funzionalità o l'intero algoritmo arrivando persino a smettere di usarli, perdendo di conseguenza i vantaggi che portavano. Analizzerò ora alcuni attacchi per ogni precedente categoria, soffermandomi in particolare sul funzionamento dell'attacco e sulle possibili contromisure che possono essere prese per mitigarlo o prevenirlo del tutto.

1.4 Attacchi Man-In-The-Middle (MITM)

Un attacco Man-In-The-Middle o MITM, che letteralmente significa "uomo nel mezzo", dà la possibilità ad un attaccante di intercettare, modificare o eliminare i messaggi di una conversazione tra un client e un server, senza che nessuna delle due parti se renda ne conto. Nella sezione precedente non è stata definita una categoria in particolare per questo l'attacco MITM, poichè si tratta di una tipologia di attacco a sé stante e che, come vedremo successivamente, viene spesso utilizzato all'interno di altri attacchi per poterli eseguire correttamente.

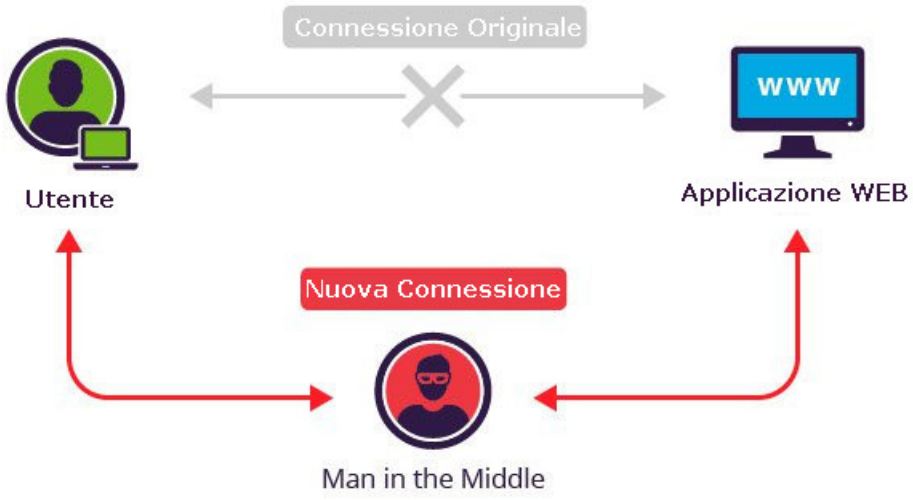

Figura 1.10. Rappresentazione di un attacco Man-In-The-Middle.

Questa tipologia di attacchi ha successo solo se sia il client sia il server sono effettivamente convinti di stare comunicando direttamente l'uno con l'altro. Sebbene il protocollo TLS preveda la possibilità di autenticare entrambe le parti mediante l'uso di certificati rilasciati da una Certification Authority (CA), una buona parte delle comunicazioni avviene tra un client anonimo e un server. Di conseguenza, per l'attaccante è sufficiente riuscire a falsificare l'identità del server, ingannando il client, per poter eseguire con successo l'attacco. Per creare una copia del certificato digitale del server sarebbe necessaria la chiave privata del server stesso, perciò l'attaccante può limitarsi a presentare al client un certificato autofirmato per il dominio per cui è stata fatta richiesta oppure un certificato effettivamente valido rilasciato da una CA, ma facente riferimento ad un altro dominio.

La maggior parte dei software applicativi rileva il tentativo dell'attaccante di presentare un certificato falsificato e mostra all'utente un messaggio di avviso, lasciandogli la libertà di scegliere se chiudere o continuare la comunicazione. Affinché l'attacco abbia un esito positivo, l'attaccante deve quindi fare affidamento o su difetti nei software applicativi, i quali non rilevano il tentativo di contraffazione del certificato, oppure sulla negligenza e l'inesperienza degli utenti che molto spesso ignorano i messaggi di avviso, rendendosi così vulnerabili ad un possibile attacco da parte di malintenzionati.

1.5 Attacchi alla crittografia

Per questa prima categoria ho scelto di mostrare il funzionamento di due attacchi e delle relative contromisure. Il primo è ai danni dell'algoritmo RSA e si tratta dell'attacco di Bleichenbacher, uno dei primi di questa categoria ad essere scoperto; il secondo è chiamato Invalid Curve è colpisce la crittografia a curve ellittiche.

1.5.1 Bleichenbacher attack

Conosciuto anche come "Adaptative Chosen Ciphertext Attack" o "Million Message Attack", questo attacco è stato scoperto da Daniel Bleichenbacher nel 1998. In un semplice attacco "chosen ciphertext", l'attaccante sceglie arbitrariamente il ciphertext e lo invia alla vittima ottenendo in risposta il testo in chiaro o parte di esso; invece, se l'attaccante effettua la scelta del ciphertext in base ai risultati dei tentativi precedenti, allora questa tipologia viene chiamata *adattativa*. Secondo quanto descritto da Bleichenbacher in "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1", è possibile effettuare questo attacco contro i sistemi che usano un algoritmo di crittografia RSA con il padding PKCS#1 v1.5 [12]. Il formato PKCS#1 v1.5 per la cifratura, ad esempio del PreMasterSecret, è strutturato come segue: due byte pari a 0x00 0x02, una sequenza di byte casuali, un byte di separazione pari a 0x00 ed infine il PreMasterSecret. Tale messaggio viene successivamente codificato ed inviato al server, il quale eseguirà la decifrazione, effettuerà un controllo sui primi due byte ed infine rimuoverà la parte di byte casuali, ottenendo così il PreMasterSecret. Nel caso in cui i primi due byte non corrispondano a 0x00 0x02, il server rileva l'errore ed invia un messaggio al client avvisandolo che il testo codificato non è conforme al formato PKCS#1 v1.5. Come nel caso dell'attacco Padding Oracle che vedremo più avanti, una volta intercettato il messaggio, un attaccante può sfruttare questo comportamento del server come un *oracolo* con lo scopo finale di decifrare il messaggio ed ottenere le informazioni che desidera.

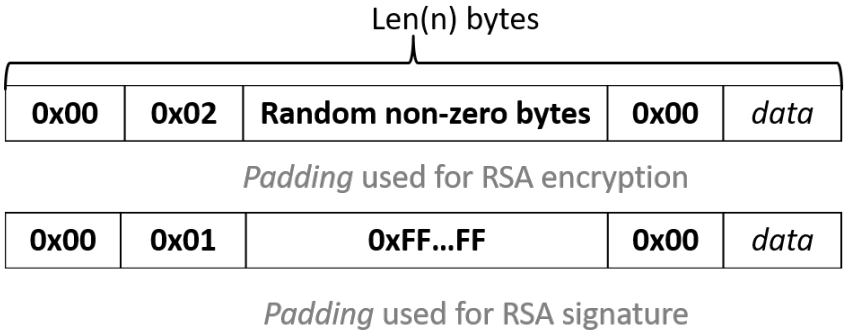

Figura 1.11. Padding utilizzato per la cifratura e la firma RSA.

Indichiamo ora con C il testo cifrato (ciphertext), con P il testo in chiaro (plaintext), con e l'esponente pubblico del server, con n il modulo e con s un valore intero scelto a piacere. L'attaccante intercetta il messaggio cifrato $C = P^e \ mod \ n$ e successivamente invia al server una serie di messaggi $C' = Cs^e \ mod \ n$, variando opportunamente il valore s fino a quando non

riceve più in risposta il messaggio del server che segnala che il formato PKCS#1 v1.5 non è stato rispettato. A questo punto l'attaccante è a conoscenza del fatto che i primi due byte del messaggio, ovvero le prime 16 cifre, sono uguali a 0x00 0x02 e di conseguenza $C' = Cs^e = (Ps)^e \ mod \ n \in [2n/2^{16}, 3n/2^{16})$. Con la stessa strategia è possibile trovare un valore s_1 tale che $s_1 \geq 2^{16}/3$ e riuscire a determinare un valore $(Pss_1)^e \ mod \ n \in [2n/2^{16}, 3n/2^{16})$; di conseguenza avremo che $(Ps)^e \ mod \ n \in [6n/2^{32}, 9n/2^{32})$. Ripetendo questi passi un numero finito di volte è possibile ridurre l'intervallo ad un unico valore, il quale moltiplicato per s^{-1} ci fornirà il plaintext P da cui potremo facilmente estrarre il PreMasterSecret. Secondo le stime di Daniel Bleichenbacher, è sufficiente esaminare circa 2^{20} messaggi affinché l'algoritmo termini e ci fornisca un risultato.

Questa tipologia di attacchi mette in luce come anche degli innocui messaggi di errore rendano vulnerabile un server, rivelando informazioni utili ad un attaccante per effettuare un attacco. In seguito alla scoperta dell'attacco, nel 1998, viene stabilito il nuovo formato PKCS#1 v2.0, il quale definisce un nuovo schema di cifratura per RSA denominato OAEP ("Optimal Asymmetric Encryption Padding"), il quale non risulta più vulnerabile all'attacco di Bleichenbacher.

1.5.2 Invalid curve

In crittografia, una "curva ellittica" è un insieme di coordinate (x, y) dove x e y sono compresi tra 0 e p, tale che p è un numero primo e $y^2 = x^3 + ax + b \bmod p$, più un punto bonus chiamato "punto all'infinito" che, controintuitivamente, si comporta in modo simile al numero zero: se lo sommiamo a un qualsiasi numero, otteniamo come risultato il numero stesso.

Le curve ellittiche possiedono alcune importanti proprietà:

- I punti sulla curva formano un gruppo abeliano, cioè un gruppo nel quale il risultato dell'applicazione di un'operazione non dipende dall'ordine in cui sono scritti gli elementi. Perciò i punti della curva ellittica si possono sommare tra loro come in una normale addizione;
- Un punto può essere moltiplicato per un numero naturale n semplicemente sommando sé stesso n volte;
- Per ogni punto (x, y) esiste un inverso $(x, -y)$ tale che qualsiasi punto sommato al suo inverso produce come risultato il punto all'infinito;
- Se scegliamo a, b e n nella funzione precedente, la moltiplicazione per un numero regolare è una funzione botola: ossia una funzione semplice da calcolare in una direzione ma difficile da calcolare nella direzione opposta; per esempio, dato un punto P e un numero n, calcolare un punto Q tale che $Q = n * P$ è semplice, ma dato il punto P e il punto Q, trovare un n tale che $Q = n * P$ è molto difficile.

Generalmente, gli algoritmi che si basano sulle curve ellittiche sono scritti attorno a una curva specifica; il client e il server si scambiano i punti sulla curva e gli scalari e successivamente li utilizzano per fare i calcoli. I problemi sorgono quando questi algoritmi vengono esposti a dei punti che non soddisfano l'equazione della curva.

Un attacco "Invalid Curve" avviene quando un client forza il server a calcolare il segreto comune usando un punto non appartenente alla curva definita dall'algoritmo; ciò può avvenire se il server non effettua un controllo sull'appartenenza del punto alla curva, rendendosi vulnerabile ad un attacco esterno. Se un attaccante ha la possibilità di mandare un punto non valido, moltiplicato per un numero n e vederne il risultato allora può riuscire a ricavare la chiave segreta. L'attaccante può inviare un punto appartenente ad una curva differente da quella specificata dall'algoritmo e che comprenda pochi punti al suo interno. Per esempio, potrebbe scegliere un punto non giacente sulla curva con la coordinata y pari a zero; poiché sappiamo che l'inverso di un punto (x,0) è sempre (x,0), allora questo punto sommato a sé stesso darebbe come risultato il punto all'infinito.

Se l'attaccante sceglie un punto P di questo tipo, osservando il risultato del segreto moltiplicato per il punto P, scoprirà se il segreto è pari o dispari. Sommando il punto P a sé stesso si ottiene il punto all'infinito, aggiungendo di nuovo il punto P si ottiene di nuovo il punto P stesso. Conoscendo ciò, se il risultato è il punto P iniziale allora il segreto è dispari, se invece il

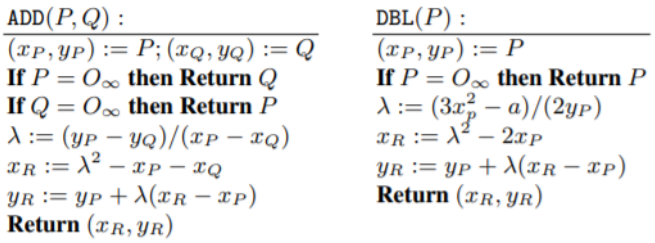

Figura 1.12. Operazioni possibili negli algoritmi a curve ellittiche.

risultato è il punto all'infinito è pari. A questo punto, l'attaccante può inviare altri punti uguali al punto all'infinito quando moltiplicati per 3, per conoscere *segreto* mod 3, punti uguali al punto all'infinito quando moltiplicati per 5, per conoscere *segreto* mod 5, e così via fino a quando è possibile usare il Teorema Cinese del Resto per calcolare il segreto effettivo.

In un certo senso, questo attacco può essere paragonato al bug Heartbleed, che vedremo successivamente, poiché anch'esso fa trapelare la chiave privata del server e permette ad un utente malintenzionato di averne accesso e di poter impersonare il server il futuro.

1.5.3 Contromisure

A differenza di altre categorie, per gli attacchi alla crittografia non esistono delle regole generali per prendere delle contromisure; è invece necessario studiare caso per caso e introdurre delle soluzioni "ad hoc".

Per l'attacco di Bleichenbacher, come abbiamo detto, una semplice soluzione è quella di non utilizzare più il formato PKCS#1 v1.5 ma aggiornarlo alla versione 2.0 che introduce il nuovo schema di cifratura OAEP. Tuttavia, se si ha la necessità o se si vuole continuare ad utilizzare la versione 1.5, è bene conoscere il rischio che si corre e le possibili contromisure che possono essere prese per annullare completamente la vulnerabilità all'attacco. La prima immediata contromisura che può essere presa è quella di evitare che il server comunichi il tipo di errore che è avvenuto (in questo caso che il padding non è conforme al formato PKCS# v1.5), ma invece venga segnalato un errore generico che non dia informazioni all'attaccante. Sebbene la vulnerabilità iniziale sia stata cancellata, in questo modo ne sorge una seconda: l'attacco di Bleichenbacher si è evoluto sfruttando le tempistiche di segnalazione degli errori; l'attaccante può calcolare quanto ci mette il server a segnalare un errore per capire che cosa l'abbia causato. In questo modo l'attacco risulta sicuramente essere più complicato, ma ancora fattibile. La seconda contromisura che può essere implementata e che risolve definitivamente il problema consiste nel far sì che il server non segnali nessun errore quando il padding non è conforme, bensì sostituisca il *PreMasterSecret* con una sequenza di byte casuali e prosegua nella fase di handshake. Al termine, i messaggi Finished che client e server si scambieranno saranno differenti da quelli attesi e la connessione verrà chiusa, in questo modo il server avrà inserito un ritardo nella segnalazione dell'errore impedendo all'attaccante di conoscere l' effettiva ragione per la quale la connessione è terminata.

Per quanto riguarda l'attacco Invalid Curve la soluzione è molto più semplice, anche se uno studio condotto dalla Ruhr University di Bochum ha dimostrato come molti di coloro che implementano la crittografia a curve ellittiche non ne siano a conoscenza. Per prevenire completamente l'attacco è sufficiente che il server effettui un controllo sul punto che gli viene passato, accertandosi che il punto appartenga effettivamente alla curve corretta. Il costo di questa operazione è decisamente trascurabile rispetto a una moltiplicazione scalare completa di un punto sulla curva ellittica. Effettuando questo semplice controllo, in caso di attacco, il server rileverebbe immediatamente il tentativo di utilizzare un punto esterno alla curva e terminerebbe la connessione, impedendone così l'esecuzione dell'attacco.

1.6 Attacchi alle ciphersuites

I due attacchi principali di questa categoria che andrò a mostrare sono l'attacco Padding Oracle e l'attacco POODLE. Per quanto riguarda il primo attacco, successivamente analizzerò brevemente anche le due varianti Lucky13 e la CVE-2016-2107; per il secondo invece verranno mostrate sia la versione contro SSL sia quella contro TLS.

1.6.1 Padding Oracle

Pubblicato nel 2002 da Serge Vaudenay, questo attacco consente di recuperare parti di testo cifrato, a patto che nella comunicazione venga utilizzata una suite in modalità CBC [23]. In questa modalità, il plaintext viene diviso in blocchi di lunghezza fissa e, nel caso in cui la lunghezza del plaintext non sia multipla della lunghezza del blocco, vengono aggiunti dei byte di padding per ottenere un numero intero di blocchi; se invece la lunghezza del plaintext è già un multiplo della lunghezza del blocco verrà aggiunto un intero blocco di padding. Inoltre, nella modalità CBC, il padding non è autenticato e può essere modificato a piacere nella sua forma crittografata senza violarne l'integrità. Questa tipologia di attacco chiamata "chosen ciphertext" è la stessa dell'attacco di Bleichenbacher e si basa sull'esistenza di un *oracolo*, ossia sulla possibilità da parte di chi lo esegue di avere la conferma se un dato tentativo sia corretto o meno.

Nelle operazioni di cifratura, per ottenere il primo blocco cifrato C1 a partire dal blocco di plaintext P1 è necessario eseguire l'operazione di XOR tra il blocco P1 e il vettore di inizializzazione IV, ovvero una sequenza casuale di byte di grandezza pari a un blocco, e successivamente cifrare il risultato con la chiave di cifratura. Una volta ottenuto il blocco C1, si esegue l'operazione di XOR tra questo blocco e il secondo blocco di plaintext P2, ed infine si cifra nuovamente il risultato con la chiave ottenendo così il secondo blocco cifrato C2. Si prosegue in questo modo fino alla cifratura di tutti di blocchi di plaintext.

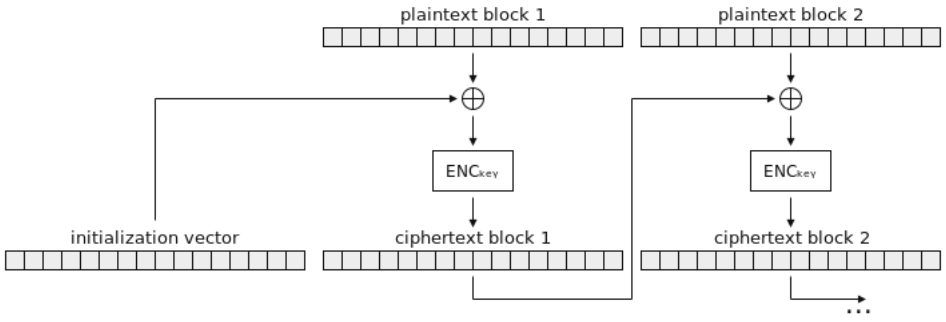

Figura 1.13. Operazione di cifratura dei blocchi in modalità CBC.

Per la decifrazione si eseguono le stesse operazioni ma in ordine inverso: si decifra con la chiave di decifrazione, ottenendo un blocco BLOCK, e poi si esegue l'operazione di XOR tra questo blocco e il blocco cifrato precedente (o il blocco IV nel caso si stia decifrando il primo blocco).

Il caso in cui sono presenti più blocchi cifrati si può tranquillamente ridurre al caso in cui sia presente un singolo blocco da decifrare e il vettore di inizializzazione IV.

Per prima cosa l'attaccante inizializza il blocco IV a 0, il quale sarà utilizzato nell'operazione di XOR e, mediante la sua modifica, gli permetterà di manipolare il plaintext a piacere. Successivamente modifica ripetutamente l'ultimo byte del blocco IV, fino ad ottenere come risultato dell'operazione di XOR un plaintext P'1 con l'ultimo byte uguale a 0x01, cioè un padding valido. Esegue poi un'operazione di XOR tra il blocco IV modificato e il plaintext P'1 ottenuto e inserisce l'ultimo byte del risultato in un vettore di azzeramento IV0. In questo modo, utilizzando questo

blocco IV0 nell'operazione di XOR con il blocco BLOCK, si otterrà come risultato un plaintext P'1 con l'ultimo byte uguale a 0x00. Secondo la proprietà per cui se A $\oplus$ B = 0 allora A = B, abbiamo che se IV0 $\oplus$ BLOCK = 0 allora IV0 = BLOCK.

A questo punto, l'attaccante ricomincia con le operazioni di modifica del blocco IV fino ad ottenere 0x02 0x02 negli ultimi due byte del plaintext P'2 e poi esegue nuovamente l'operazione di XOR tra il blocco IV modificato e il plaintext P'2 ricavato, ottenendo così il penultimo byte del blocco IV0. Ripete ciclicamente tutte le operazioni, in modo da ottenere tutti i possibili padding (0x03 0x03 0x03, 0x04 0x04 0x04 0x04, etc...) e ricavando un byte alla volta l'intero blocco BLOCK. Una volta completate le operazioni, basterà che esegua l'operazione di XOR tra il blocco BLOCK e il vettore di inizializzazione IV originale e avrà ottenuto il blocco di plaintext P1. Nel caso in cui siano presenti più blocchi cifrati, è possibile eseguire le stesse operazioni su di essi per ricavare i relativi plaintext, il tutto pur non essendo in possesso della chiave di decifrazione.

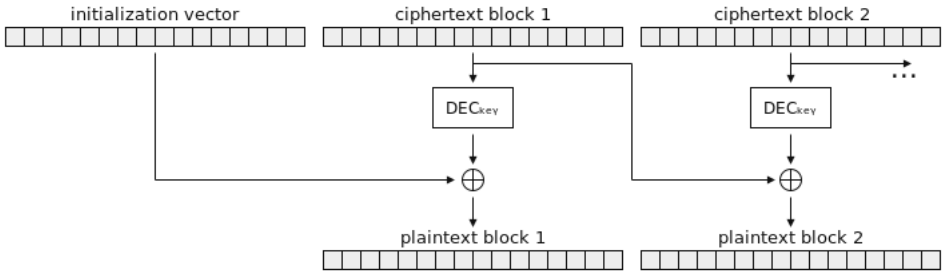

Figura 1.14. Operazione di decifrazione dei blocchi in modalità CBC.

Secondo le stime dell'autore di questo attacco, sono necessari circa $256/2 = 128$ tentativi per indovinare un singolo byte. Poichè un errore sul padding determina la chiusura della sessione tra client e server, un attaccante avrà bisogno di aprire all'incirca $128 * N$ sessioni, dove N è il numero di byte che vuole decifrare.

1.6.2 Lucky13

Lucky13 è una variante dell'attacco Padding Oracle, pubblicata per la prima volta nel 2013 da Nadhem AlFardan e Kenny Paterson, che sfrutta la differenza del tempo di processamento dei vari messaggi TLS con padding differente. I messaggi che hanno almeno due byte di padding corretto vengono processati più velocemente dal server rispetto a quelli che hanno un solo byte di padding corretto o che hanno il padding non corretto. Perciò, un attaccante può riconoscere questi messaggi mediante la temporizzazione dei pacchetti di risposta e, una volta individuati, è possibile ricavare parti di testo cifrato con le stesse modalità utilizzate nell'esecuzione dell'attacco Padding Oracle.

Lo svantaggio principale, rispetto all'attacco originale presentato prima, risiede nel fatto che l'attaccante ha bisogno di trovarsi vicino alla vittima quando agisce, in modo da evitare che i pacchetti che viaggiano in rete siano soggetti a ritardi non controllabili, che impedirebbero la corretta esecuzione dell'attacco.

1.6.3 CVE-2016-2107

La vulnerabilità CVE-2016-2107 è un perfetto esempio di come un attacco possa non appartenere a solo una delle categorie di attacchi che abbiamo elencato: gli attacchi alle ciphersuites e gli attacchi

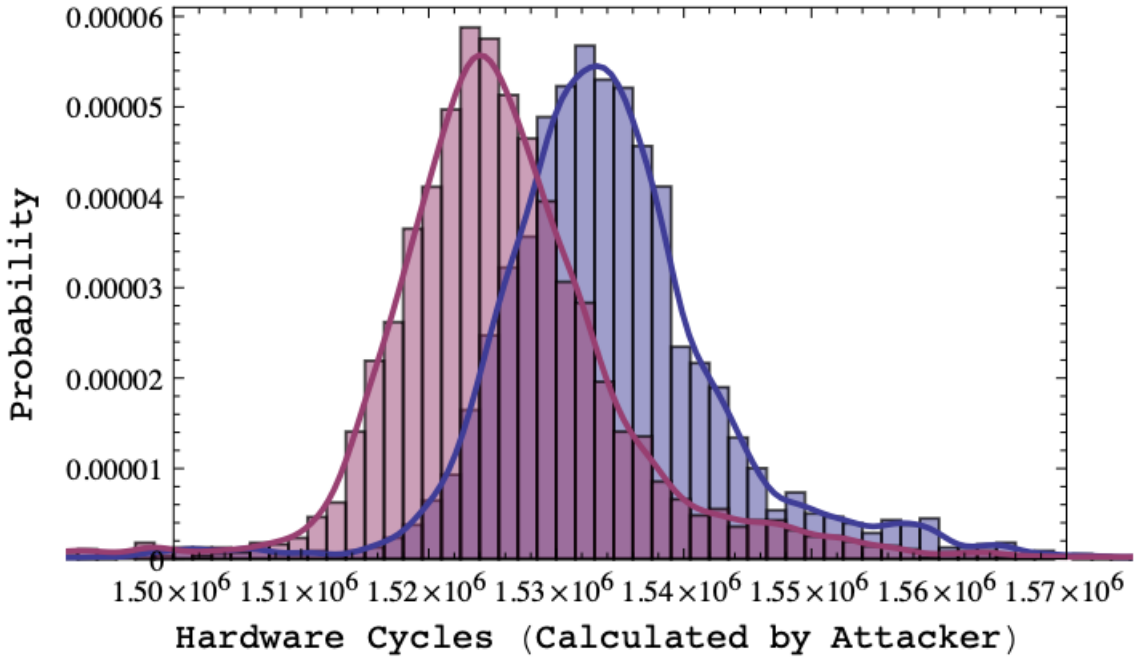

Figura 1.15. Risultato dell'analisi delle risposte con padding lungo (rosso) e corto (blu).

alle implementazioni delle librerie. Infatti questa vulnerabilità riguarda l'implementazione AES-NI ("Advanced Encryption Standard New Instructions"), ma solamente nelle versioni OpenSSL 1.0.1 precedenti a 1.0.1t e OpenSSL 1.0.2 precedenti a 1.0.2h. In queste particolari versioni non veniva tenuta in considerazione l'allocazione di memoria durante un certo controllo del padding, il che permetteva ad un attaccante di ottenere informazioni sensibili in chiaro mediante un attacco Padding Oracle contro la sessione AES CBC. Questa vulnerabilità è stata introdotta a causa di una errata correzione di un'altra (CVE-2013-0169).

1.6.4 POODLE

POODLE, sigla che sta per Padding Oracle On Downgraded Legacy Encryption, è stato scoperto nel 2014 dal Security Team di Google [26]. Si tratta di un attacco che sfrutta una caratteristica propria di molte implementazioni TLS: ovvero, per poter comunicare con i server cosiddetti "legacy", sono retrocompatibili e supportano ancora SSL 3.0, un protocollo obsoleto e insicuro.

Durante il primo tentativo di handshake, il client propone al server la più alta versione del protocollo supportata (es. TLS 1.2); se il server non supporta quella versione, l'handshake fallisce e il client ritenta con una versione del protocollo precedente, fino a raggiungere un accordo. In una corretta negoziazione della versione del protocollo, il downgrade può essere causato, oltre che su richiesta del server (il client propone TLS 1.2, il server può rispondere con TLS 1.0), anche da problemi di rete oppure da un attaccante che esegue un attacco Man-In-The-Middle. L'attaccante può interferire con i tentativi di handshake del client, modificando la sua proposta di utilizzare TLS 1.0 o una versione superiore, e costringendolo a proporre, e quindi usare, SSL 3.0. Questa specifica versione può utilizzare due tipologie di cifratura differenti: la cifratura a flusso RC4 o la cifratura a blocchi in modalità CBC, quest'ultima soggetta agli attacchi "chosen ciphertext" descritti precedentemente e perciò vulnerabile.

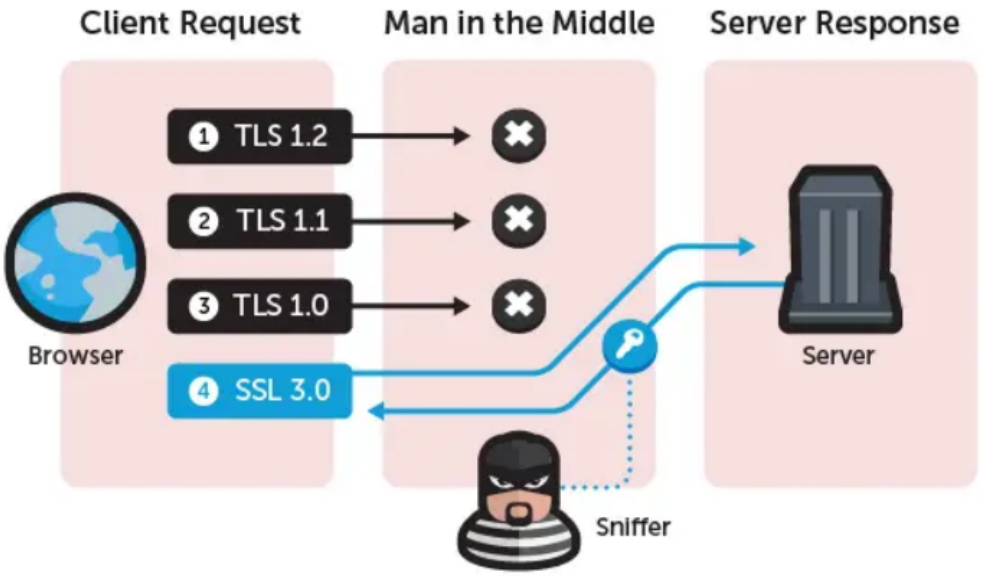

Figura 1.16. Funzionamento dell'attacco POODLE contro SSL 3.0.

1.6.5 POODLE contro TLS

Nel dicembre del 2014, due mesi dopo la scoperta dell'attacco POODLE contro SSL 3.0, il ricercatore della Qualys, Ivan Ristic, pubblica un articolo intitolato "POODLE bites TLS" ("POODLE morde TLS") [28], nel quale annuncia la scoperta di una nuova variante dell'attacco POODLE ai danni di alcune implementazioni di TLS.

SSL 3.0 non richiedeva un particolare formato per il padding, eccetto per l'ultimo byte che ne doveva indicare la lunghezza, e perciò risultava essere vulnerabile. Nonostante TLS avesse regole più severe per quanto riguardava il formato del padding, è risultato che circa il 10% dei server utilizzavano un'implementazione TLS che non ne eseguiva il controllo di validità dopo la decifrazione, rendendo quei server possibili bersagli di un attacco POODLE.

Dunque, non era più necessario costringere i server ad utilizzare SSL 3.0 ma anche la versione 1.2 di TLS andava bene. Gli obiettivi principali dell'attacco erano i browser, nei quali, per cominciare, era necessario iniettare del codice JavaScript; successivamente erano sufficienti circa 256 richieste per scoprire un unico carattere di un cookie o solamente 4096 per un cookie di 16 caratteri.

1.6.6 Contromisure

Per la prima tipologia di attacchi, ovvero Padding Oracle e le sue varianti, la soluzione più ragionevole è quella di autenticare il ciphertext. Per raggiungere lo scopo sono possibili due alternative: la prima consiste nell'utilizzare una modalità di cifratura autenticata come può essere GCM oppure OCB, mentre la seconda prevede di continuare ad utilizzare la modalità CBC ma usando la modalità per l'autenticazione dei messaggi HMAC. In entrambi i casi, il server sarà in grado di rifiutare qualunque messaggio modificato da un attaccante senza nemmeno eseguirne la decifrazione. Per la variante Lucky13 esiste una terza soluzione che prevede l'inserimento di ritardi casuali nel processo di decifrazione nella modalità CBC. Tuttavia questa soluzione non previene completamente l'attacco ma lo mitiga solamente, poiché rende il compito dell'attaccante più arduo ma, anche aumentando il numero di campioni utilizzati, è ancora possibile eseguire

l'attacco. Infine, per la vulnerabilità CVE-2016-2107 esiste una soluzione definitiva che consiste nell'aggiornare OpenSSL ad una versione che non abbia problemi riguardanti l'implementazione AES-NI, risolvendo così definitivamente i problemi.

Per quanto riguarda l'attacco POODLE, disabilitare completamente SSL 3.0 può non essere la soluzione più pratica, poiché è possibile che lo si voglia ancora utilizzare quando si comunica con sistemi più obsoleti. Perciò è stato introdotto il meccanismo `TLS_FALLBACK_SCSV`: un valore che viene incluso nel *ClientHello.cipher_suites* dell'handhsake e che permette ai server aggiornati di rifiutare una connessione in caso di tentativo di downgrade. Il server, ogni volta che una nuova connessione include questo valore, controlla sia la versione del protocollo proposta dal client sia la più alta versione supportata dal client stesso: se quest'ultima è più alta allora il server rifiuta la connessione, impedendo di fatto di utilizzare SSL 3.0. Invece, per risolvere il problema sfruttato dalla versione di POODLE contro TLS è stato sufficiente inserire un controllo di validità del padding nell'implementazione che non lo supportava, risolvendo in questo modo i problemi che derivavano dalla sua mancanza.

1.7 Attacchi al protocollo TLS

Riguardo le funzionalità del protocollo TLS ho scelto di mostrare inizialmente un trittico di attacchi chiamati CRIME, BREACH e HEIST riguardanti tutti e tre la compressione dei dati; successivamente verrà analizzato uno degli attacchi più recenti di questa categoria chiamato Selfie e riguardante l'uso delle PSK.

1.7.1 CRIME, BREACH, HEIST

Nel corso del 2012, Juliano Rizzo e Thai Duong, ovvero coloro che avevano già scoperto l'attacco BEAST contro TLS, presentano al mondo un nuovo attacco chiamato CRIME. Si tratta del primo di una serie di attacchi la cui caratteristica principale consiste nello sfruttare la compressione dei dati per estrarre informazioni riservate [33]. In particolare, CRIME sfrutta la compressione a livello TLS, da non confondere con la più diffusa compressione HTTP la quale sarà colpita l'anno dopo, nel 2013, dall'attacco BREACH. Infine, nel 2016, l'attacco HEIST permetterà di eseguire i due precedenti attacchi esclusivamente dal browser, sfruttando le sue debolezze, senza richiedere di dover effettuare prima un attacco MITM.

L'attacco CRIME ("Compression Ratio Info-leak Made Easy"), come detto, sfrutta la compressione a livello TLS, ovvero la compressione dei dati che avviene prima della crittografia. Questa compressione utilizza la ridondanza dei dati per diminuire la lunghezza della richiesta HTTP e sostituisce ogni istanza di una stringa duplicata con un puntatore alla prima occorrenza di quella stringa. Perciò, un attaccante in posizione di Man-In-The-Middle può osservare il traffico di rete della vittima e successivamente manipolarlo in modo da inviare ripetute richieste HTTP al sito con il quale sta comunicando, nelle quali inserirà dati predicibili mischiati a contenuti arbitrari per osservarne il risultato. Eseguendo questo attacco è possibile estrarre dei dati dalle richieste HTTP, perciò il miglior risultato che si può ottenere è ricavare il token di sessione della vittima. La condizione necessaria affinché l'attacco possa essere messo in atto è che, sia il server sia il client, supportino l'algoritmo di compressione dati *DEFLATE* oppure il protocollo *SPDY*, quest'ultimo però è decisamente meno diffuso. Se nel periodo in cui l'attacco è stato scoperto, per i client questa condizione era rispettata, lo stesso non si può dire per i server, poiché non tutti implementavano questo algoritmo. Tuttavia, se anche il server supportava l'algoritmo di compressione, era possibile eseguire l'attacco in questo modo:

1. Forzare la vittima a inviare ripetute richieste HTTP al sito con cui sta comunicando;
2. In ogni richiesta HTTP, provare ad aumentare i dati ridondanti;
3. Analizzare la variazione della lunghezza della richiesta HTTP dopo la compressione: se la lunghezza è diminuita allora la ridondanza è aumentata ed è stato indovinato il primo carattere;

4. Ripetere il processo per indovinare gli altri caratteri

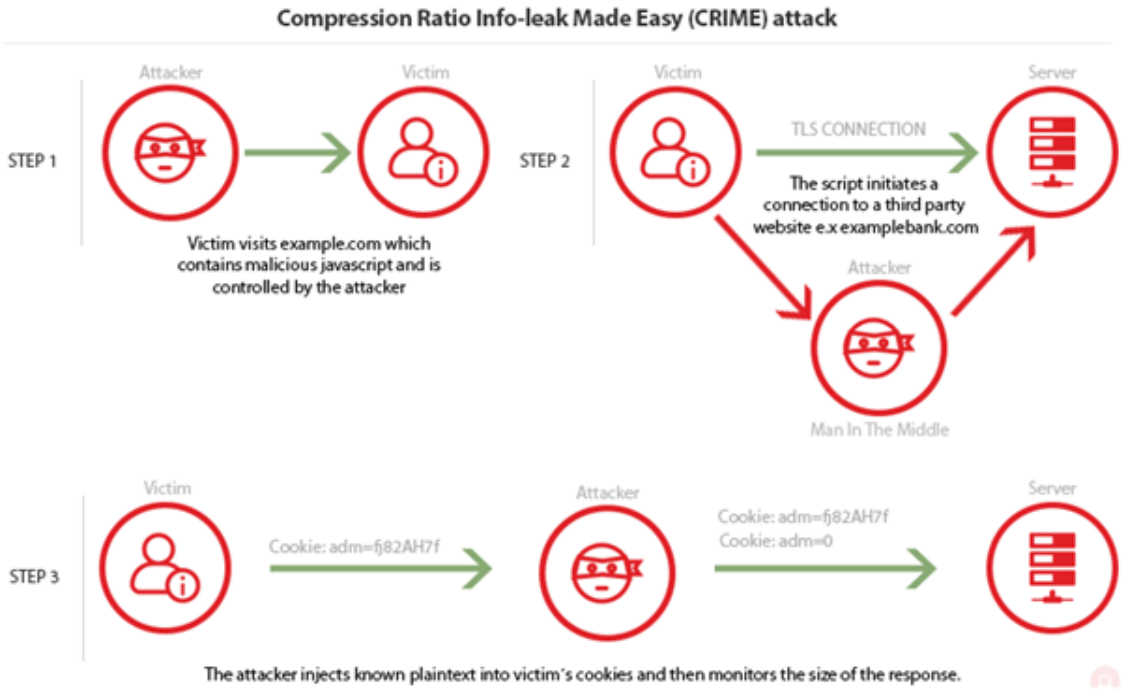

Figura 1.17. Esempio dell'esecuzione di un attacco CRIME.

Sebbene non sia obbligatorio, l'uso di JavaScript offre buone performance nell'esecuzione dell'attacco: sono sufficienti 6 richieste per indovinare un byte di dati. Nel 2012, il browser Chrome supportava sia l'algoritmo *DEFLATE* che il protocollo *SPDY*, mentre il browser Firefox solamente il primo; tuttavia questi due browser utilizzano gli aggiornamenti automatici perciò la maggior parte degli utenti avrà ottenuto in breve tempo una versione aggiornata dei due browser, ovvero senza il supporto per la compressione TLS. Secondo il progetto *SSL Pulse* che si occupata del monitoraggio di oltre 150.000 siti SSL/TLS, nel 2012 circa il 42% dei siti analizzati era vulnerabile a questo attacco ma, com'era prevedibile, questa percentuale è calata drasticamente negli anni fino ad arrivare al 0,1% nell'ultimo aggiornamento datato 17 ottobre 2021.

L'anno successivo venne scoperto l'attacco BREACH, una variante dell'attacco CRIME effettuata contro la compressione HTTP, molto più comune rispetto a quella a livello TLS. Al di là della differenza sul tipo di compressione attaccata, l'esecuzione dell'attacco BREACH ricalca essenzialmente gli stessi passi dell'attacco visto precedentemente; tuttavia questo attacco è indipendente dalla versione di SSL o TLS e funziona contro qualsiasi ciphersuite, sia stream, un po' più facilmente, sia a blocchi, richiedendo del lavoro aggiuntivo.

In ordine cronologico, di questo gruppo di attacchi l'ultimo ad essere scoperto si chiama HEIST ("HTTP Encrypted Information can be Stolen through TCP-window"), il quale introduce una serie di tecniche che permettono di effettuare attacchi contro SSL/TLS direttamente dal browser. Con HEIST diventa possibile sfruttare alcune vulnerabilità nei protocolli di rete senza che sia necessario intercettarne il traffico. In particolare, questo attacco side-channel utilizza il modo in cui le risposte sono inviate a livello TCP in combinazione con il fatto che SSL/TLS rilascia informazioni sulla lunghezza della risposta, per poter desumere la lunghezza esatta del plaintext. Concretamente, significa che i due precedenti attacchi, CRIME e BREACH, sono ora effettuabili direttamente dal browser, mediante un qualunque sito o script malevolo, senza la condizione necessaria di dover essere in posizione MITM per intercettare e modificare il traffico.

1.7.2 Selfie

Ad opera di Nir Drucker e Shay Gueron, "Selfie" è uno dei più recenti attacchi scoperti e risale al 2019 [36]. Questo attacco sfrutta l'uso delle chiavi pre-condivise (Pre-Shared Keys o PSK) in TLS e si basa sul fatto che TLS non impone l'autenticazione esplicita del server e del client in ogni messaggio. Sebbene le PSK possano essere utilizzate anche in TLS 1.2, non si tratta di una situazione molto comune; perciò la nostra attenzione è rivolta a TLS 1.3 in cui l'uso delle PSK è più diffuso. Quest'ultima versione di TLS permette a due parti di stabilire delle chiavi di sessione condivise a partire da delle PSK concordate precedentemente out-of-band. Queste PSK sono utilizzate per la mutua autenticazione delle due parti, assumendo che non siano state condivise con nessun altro, e permettendo così di saltare la verifica dei certificati e risparmiando banda.

Prendiamo in considerazione una rete di peer comunicanti, dove ognuno di essi può agire come un client TLS o come un server TLS, e assumiamo che le PSK siano già state pre-distribuite. L'attacco si basa su due proprietà di TLS:

- Un nodo può aprire connessioni indipendenti parallele
- Un nodo che agisce come client non controlla esplicitamente l'identità del server, ma verifica solamente che sia un legittimo proprietario della relativa PSK

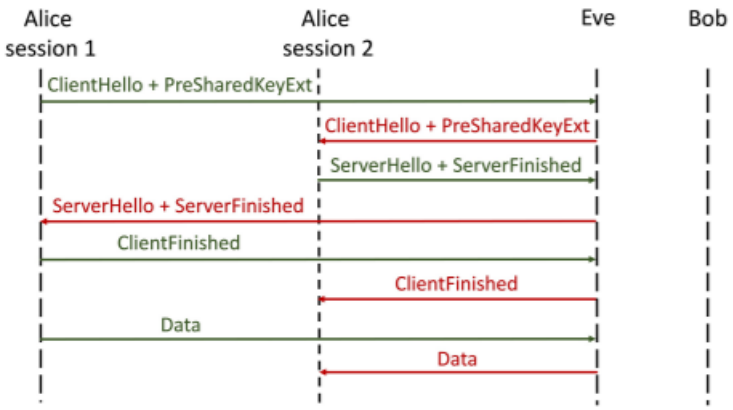

Figura 1.18. Rappresentazione dell'esecuzione dell'attacco Selfie.

Chiamiamo le due parti che vogliono instaurare una connessione Alice e Bob e l'attaccante che intercetta la conversazione Eve. Avviene ciò che è rappresentato in Fig. 1.18:

1. Alice invia a Bob il messaggio *ClientHello* con un'estensione *pre_shared_key*;
2. Eve cattura il messaggio e lo invia indietro ad Alice, fingendosi Bob;
3. Alice autentica il messaggio e viene indotta a credere che il messaggio sia stato mandato da Bob, l'unico a possedere la PSK corretta;
4. Alice risponde con i messaggi *ServerHello* e *ServerFinished*;
5. Eve cattura i messaggi e li invia indietro ad Alice;
6. A questo punto Alice ha aperto una sessione *Selfie* con sé stessa;
7. Quando Alice manda dei dati, riceve gli stessi dati indietro.

Per dimostrare la minaccia che porta questo attacco, immaginiamo ancora il seguente scenario. Alice apre una sessione convinta di stare comunicando con Bob, ma in realtà comunica con sé stessa. Successivamente invia un messaggio a Bob del tipo:

"Se possiedi il file data.txt puoi cancellarlo. Io ne ho una copia"

Il messaggio, invece che a Bob, viene recapitato ad Alice, la quale è indotta a pensare che il messaggio le sia stato inviato direttamente da Bob. A questo punto Alice controlla se possiede una copia del file *data.txt* (che ovviamente avrà) e cancellerà il file. Se non esistono altre copie del file, allora questo è andato perduto.

1.7.3 Contromisure

Se per l'attacco CRIME la soluzione è semplice e immediata, lo stesso non si può dire di BREACH e HEIST. Per prevenire completamente l'attacco CRIME è sufficiente disabilitare l'uso della compressione dati a livello TLS sul lato client. Infatti il client invia sempre una lista di algoritmi di compressione nel messaggio di *ClientHello* e il server ne sceglie uno di quelli che gli vengono proposti, ma se il client non propone alcun metodo di compressione allora i dati non verranno compressi.

Come spiegato precedentemente, l'attacco BREACH sfrutta la compressione a livello HTTP e quindi disabilitare quella a livello TLS è inutile. Sfortunatamente, non esiste un'unica soluzione pratica per prevenire l'attacco, ma bensì è solo possibile attivare delle contromisure volte a mitigarlo e a rendere il compito degli attaccanti più complicato. Ecco le mitigazioni che è possibile mettere in atto:

- disabilitare la compressione HTTP, ma riduce nettamente le performance;
- separare i segreti dall'input dell'utente;
- segreti di mascheramento (randomizzazione efficace mediante un'operazione di XOR con un segreto casuale per ogni richiesta);
- protezione delle pagine vulnerabili mediante CSRF (Cross-Site Request Forgery);
- nascondere la lunghezza aggiungendo un numero casuale di byte alla risposta.

Le contromisure sono in ordine di efficacia, ma non di praticità; ognuna di esse offre più o meno protezione dall'attacco ma la difficoltà di metterle in pratica è differente.

Gli scopritori dell'attacco HEIST, Mathy Vanhoef e Tom Van Goethem, oltre ad aver spiegato come eseguirlo, hanno anche proposto alcune soluzioni per prevenirlo. La maggior parte delle soluzioni che propongono risultano essere, sempre secondo loro, infattibili, incomplete o inadeguate. Tuttavia, ne esiste una che viene etichettata come completa e che sembrerebbe il miglior approccio per prevenire l'attacco: consiste nel disabilitare i cookies di terze parti. Quando l'attaccante fa in modo che la vittima avvii le richieste a sito web bersaglio, viene restituita una risposta specifica dell'utente. Questo perché i cookie della vittima sono inclusi nella richiesta, il che significa che dal punto di vista del sito web le richieste fanno parte della sessione di navigazione della vittima. Impedendo l'inclusione di questi cookie, la richiesta verrà autenticata e nessun contenuto specifico dell'utente verrà restituito e perciò non potrà essere rubato.

Come per l'attacco HEIST, anche per l'attacco Selfie sono gli stessi scopritori a proporre una soluzione per prevenire l'attacco. La prima soluzione proposta è la modifica del protocollo TLS affinché includa la seguente restrizione:

"Le PSK esterne DEVONO essere utilizzate insieme ai certificati dei server."

Nonostante questa restrizione prevenga l'attacco, il problema consiste nel fatto che le PSK erano state introdotte nel protocollo per evitare l'uso dei certificati e, perciò, la loro utilità verrebbe meno. La miglior soluzione che viene proposta è quella di limitare l'uso delle PSK aggiungendo una regola alla loro definizione, ovvero:

"Una PSK NON DEVE essere condivisa tra più di un client e un server."

Questo previene completamente l'attacco Selfie, senza la necessità di utilizzare i certificati.

1.8 Attacchi alle implementazioni delle librerie

Infine, per gli attacchi alle librerie ho scelto di analizzare un attacco a OpenSSL chiamato Heartbleed e una coppia di attacchi all'implementazione del protocollo di handshake di TLS: il primo chiamato Early CCS riguarda sempre OpenSSL, mentre il secondo, chiamato Early Finished, riguarda OpenJDK e CyaSSL.

1.8.1 Heartbleed

Una parte importante del protocollo TLS/SSL è l'estensione chiamata *heartbeat* ("battito cardiaco") [37], la quale viene utilizzata da due computer che stanno comunicando per far sapere all'altro che sono ancora connessi, ed è molto utile in quei casi in cui nessuno dei due esegua azioni di scaricamento o caricamento per un periodo di tempo considerevole. Dopo un determinato lasso di tempo, uno dei due computer invia all'altro alcuni dati cifrati all'interno di un messaggio chiamato *Heartbeat Request*, il secondo computer risponde con una copia esatta della sequenza di dati cifrati ricevuti, per dimostrare che la connessione è ancora attiva.

Il 3 aprile 2014, una squadra di ingegneri della sicurezza (composta da Riku, Antti e Matti) e Neel Mehta di Google Security portano alla luce un bug presente in questa estensione, il quale poteva portare al furto di dati più o meno importanti: semplici indirizzi di memoria, username o password degli utenti o addirittura le chiavi di crittografia stesse. Il bug era comparso per la prima volta nella versione 1.0.1 di OpenSSL, uscita due anni prima, e riguardava la presenza di informazioni sulla lunghezza della Heartbeat Request presenti nel messaggio stesso.

Nel momento in cui un computer riceve un Heartbeat Request, legge le informazioni riguardanti la lunghezza del messaggio e alloca un buffer di memoria pari a quella lunghezza in cui salva i dati criptati, infine rilegge i dati e li invia come risposta all'altro computer. La vulnerabilità sorge poiché, l'implementazione di OpenSSL in questione, non eseguiva un controllo sulla lunghezza del messaggio in modo da verificare che fosse effettivamente uguale a quella dichiarata al suo interno. Così facendo, se una richiesta diceva di avere una lunghezza pari a 40 KB ma era solamente di 20 KB, il computer che la riceveva allocava comunque un memory buffer di 40 KB, vi salvava i 20 KB di dati cifrati al suo interno ed infine rispondeva con i 20 KB di dati cifrati più altri 20 KB di dati presenti in memoria, qualunque tipo di dato fossero.

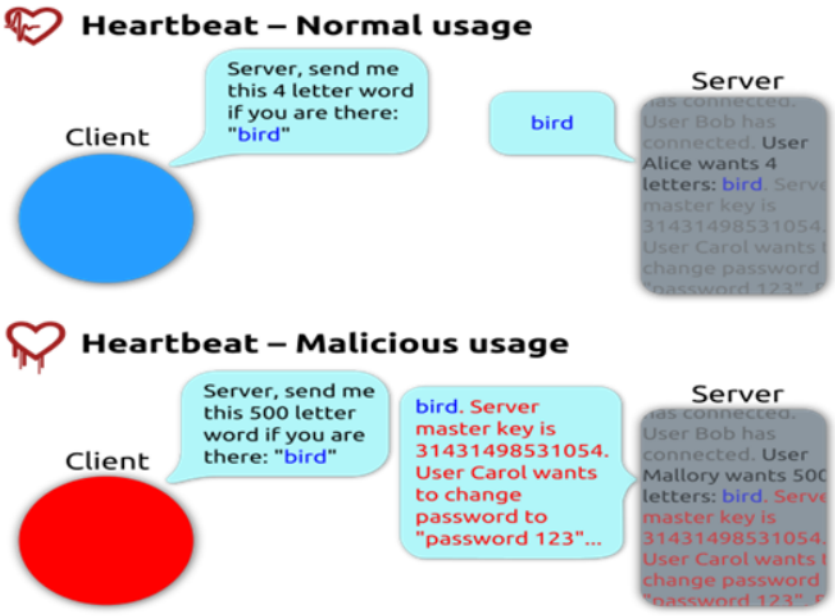

Figura 1.19. Rappresentazione dell'utilizzo dell'estensione *heartbeat*.

Una differenza particolare rispetto agli attacchi precedenti consiste nel fatto che questo non richiede di effettuare precedentemente un attacco MITM, ma gli attaccanti possono comunicare direttamente con il server o il client vulnerabile.

1.8.2 Early CCS

Questo attacco sfrutta una vulnerabilità chiamata "CCS Injection", scoperta da Masashi Kikuchi di Lepidum nel 2014 [41] e presente sin dal rilascio della prima versione di OpenSSL nel 1998. In un handshake corretto, il client e il server si scambiano dei messaggi in un ordine ben stabilito e i messaggi *ChangeCipherSpec* (CCS) devono essere inviati appena prima del messaggio Finished. OpenSSL rispetta questo ordine inviando i messaggi al momento giusto ma, in fase di ricezione, permette l'arrivo dei messaggi CCS in un momento qualunque dell'handshake e questo può essere sfruttato da un attaccante esterno al fine di decifrare e modificare i dati sul canale di comunicazione. Tuttavia, per poter eseguire l'attacco con successo, è necessario che sia il client sia il server operino su una versione vulnerabile di OpenSSL, se anche solo uno dei due risulta non essere vulnerabile allora l'attacco fallisce.

Sin dalle prime versioni di OpenSSL, se un messaggio CCS veniva inviato al client subito dopo il messaggio ServerHello, ma prima della generazione del master secret, allora le chiavi e l'hash Finished venivano generate utilizzando un master secret vuoto, basandosi dunque solo sulle informazioni pubbliche. La ricezione di un secondo messaggio CCS da parte del client non provocava il ricalcolo delle chiavi, ma solo dell'hash Finished, con l'utilizzo del master secret corretto e impedendo così all'attaccante di generare un hash Finished accettabile. Dalla versione 1.0.1 di OpenSSL, una patch modifica il punto in cui vengono calcolati i valori Finished, permettendo al server di utilizzare un hash Finished accettabile anche nel caso in cui abbia elaborato erroneamente un messaggio CCS. Tuttavia, il codice del server (a differenze di quello del client) gli impedisce di ricevere due messaggi CCS all'interno dello stesso handshake, perciò, se un attaccante inviava un CCS anticipato al server per fissare le chiavi a valori conosciuti, non era possibile per il server ricevere un secondo messaggio CCS che lo portasse a calcolare l'hash Finished corretto. Ma, come spiegato prima, con OpenSSL 1.0.1, il server calcolerà comunque l'hash Finished al momento opportuno, rispondendo al client con un hash accettabile. Se ne conclude che, in una comunicazione tra un qualunque client OpenSSL e un server OpenSSL 1.0.1, un attaccante in posizione di MITM ha la possibilità di inviare messaggi CCS a entrambe le parti per fissare le chiavi a valori a lui conosciuti, senza però provocare errori sugli hash Finished, i quali continueranno a rimanere allineati tra loro, permettendo in questo modo all'attaccante di decifrare e modificare la comunicazione tra client e server.

1.8.3 Early Finished

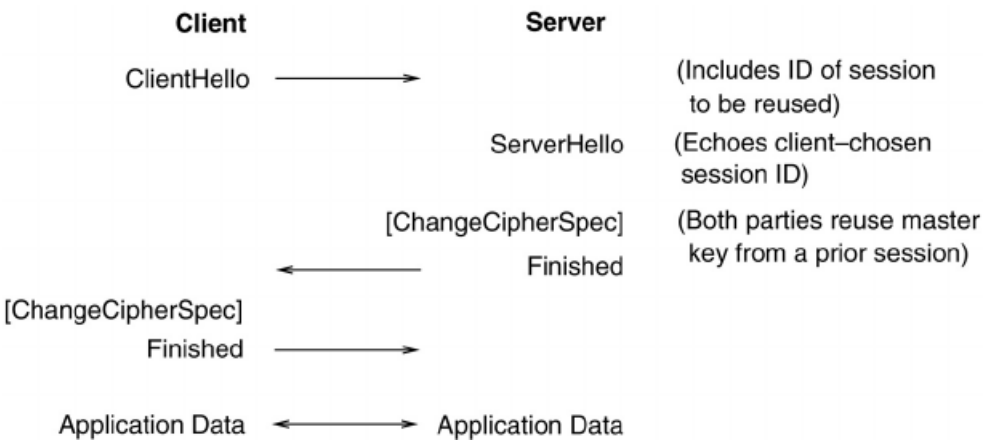

Figura 1.20. Flusso di messaggi nel protocollo di handshake abbreviato.

Nel gennaio del 2015, venne scoperto che alcune implementazioni TLS presenti nell'estensione JSSE di OpenJDK e in CyaSSL (l'attuale wolfSSL) non verificavano correttamente l'effettiva ricezione del messaggio *ChangeCipherSpec* durante il processo di handshake [43]. A causa di questo mancato controllo, un attaccante MITM poteva costringere un client a stabilire una connessione senza abilitarne la crittografia. All'invio del messaggio di *ClientHello* era sufficiente che l'attaccante rispondesse con il messaggio *ServerHello*, il messaggio *Certificate* contenente l'identità del server da impersonare ed infine il messaggio *Finished*, saltando il resto della negoziazione, incluso il messaggio CCS. Trovandosi di fronte ad un tipico caso di handshake abbreviato, queste implementazioni di TLS consideravano il server come autenticato e cominciavano ad inviare messaggi *ApplicationData* in chiaro.

1.8.4 Contromisure

Quest'ultima categoria di attacchi è quella che presenta le contromisure più semplici da applicare. Infatti, essendo le vulnerabilità presenti a causa di difetti nell'implementazione di alcune librerie, è sufficiente aspettare il rilascio di un patch che risolva il problema e aggiornare la libreria del nostro sistema alla patch più recente.

Il bug Heartbleed è presente in OpenSSL dal dicembre 2011, ed è stato diffuso con la versione 1.0.1 di marzo 2021 rimanendo presente sino alla versione 1.0.1f. Venne ufficialmente rimosso il 7 aprile 2014 insieme al rilascio della versione 1.0.1g. Questo significa che molti sistemi operativi che supportavano quelle versioni di OpenSSL risultavano vulnerabili all'attacco, tra questi ricordiamo: Debian Wheezy, Ubuntu 12.04.4 LTS, Fedora 18 e CentOS 6.5.

Per quanto riguarda la vulnerabilità "CCS Injection", come detto precedentemente, è risultato essere presente sin dalla prima versione di OpenSSL. Per tutte le versioni ancora in sviluppo al momento della scoperta dell'attacco è stata rilasciata una patch di aggiornamento che correggeva il bug in modo da non poter essere più soggetti ad attacchi del genere. La versione 0.9.8 di OpenSSL diventa non più vulnerabile con il rilascio del patch 0.9.8za, la versione 1.0.0 con il rilascio della patch 1.0.0m ed infine la versione 1.0.1 con il rilascio della patch 1.0.1h.

Infine, la vulnerabilità che era presente in alcune versioni sia in CyaSSL sia in OpenJDK 6 è stata completamente risolta con il rilascio delle due rispettive patch: la versione 3.3.0 di CyaSSl del maggio 2014 e la versione 6b34-1.13.6.1 di OpenJDK del febbraio 2015.

Capitolo 2

Strumenti di analisi del traffico

In questo capitolo verrà eseguita una breve introduzione di due strumenti molto utili per l'analisi del traffico: gli Intrusion Detection System, o IDS, e gli sniffer. Ci soffermeremo in particolare su tre di questi software e sulle motivazioni che ci hanno portato a scegliere di utilizzarli. Infine seguirà una spiegazione riguardo a come installare, configurare e rendere pronto per l'utilizzo uno degli IDS, ovvero quello che alla fine si è rivelato più adatto ad essere utilizzato nell'esecuzione di alcuni degli esperimenti.

2.1 Introduzione

Nell'ambito della sicurezza informatica, la categoria di strumenti per l'analisi del traffico di rete copre un ruolo di un certo rilievo. Un buon amministratore di rete ha il dovere di essere sempre a conoscenza della tipologia di traffico che viaggia sulla sua rete, in entrata o in uscita, e perciò deve essere in grado di determinare una serie di regole in modo da poter bloccare il traffico indesiderato. Inoltre dovrebbe poter prevenire qualsiasi tipologia di accesso non desiderato da parte di intrusi o, per lo meno, deve essere in grado di verificare successivamente se uno di questi accessi indesiderati è effettivamente avvenuto e le conseguenze che ha comportato. Se non vengono prese le dovute precauzioni attivando tempestivamente le contromisure, le conseguenze di questi accessi possono essere molto gravi, in particolare in ambito aziendale dove un accesso non autorizzato metterebbe a rischio l'interno sistema della azienda, portando potenzialmente alla perdita di informazioni sensibili o al furto di esse da parte di sconosciuti. Per poter porre un limite a questi danni, o addirittura per poterli prevenire, negli anni sono stati sviluppati molteplici strumenti che possono facilitare questo compito agli amministratori di rete. Ognuno di questi strumenti fornisce delle caratteristiche particolari, differenti da quelle che supportano gli altri, perciò è vivamente consigliato utilizzarli non singolarmente ma in contemporanea, facendoli collaborare, in modo da aumentare ancora di più la sicurezza della rete.

I principali strumenti di difesa sono i seguenti:

firewall: rappresentano la prima linea di difesa contro le minacce esterne. Si tratta di un dispositivo che permette il monitoraggio del traffico sia in entrata che in uscita, mediante l'utilizzo di una serie di regole di sicurezza che consentono o bloccano il passaggio di una certa tipologia di traffico.

sniffer: permettono di registrare tutto il traffico che attraversa la rete, il quale viene memorizzato in modo da permetterci di poterlo analizzare successivamente.

intrusion detection system o IDS: come gli sniffer, registrano tutto il traffico che attraversa la rete e lo memorizzano per consentirci di analizzarlo. Inoltre, però, permettono il rilevamento di attività inappropriate, errate o anomale che vengono opportunamente segnalate all'amministratore di rete, il quale valuterà l'evento che si è verificato. Sebbene il

rilevamento avvenga in tempo reale, in caso di attacco verrà inviato un avviso all'amministratore di rete, ma non verranno immediatamente attivate delle contromisure, sarà compito dell'amministratore di rete dover decidere sul momento in che modo agire.

intrusion prevention system o IPS: possiedono le stesse caratteristiche di un IDS, ma l'analisi del traffico "sospetto" secondo opportune regole avviene in tempo reale e con la possibilità di determinare il comportamento che la rete deve tenere in risposta ad un particolare evento. É dunque possibile rilevare un attacco in corso e agire automaticamente di conseguenza, in modo da poter bloccare l'attacco e impedire che qualcuno danneggi la nostra rete.

Di questi strumenti di analisi del traffico, ne verranno presentati tre: *Suricata*, *Zeek* e *Wireshark*. I primi due fanno parte della categoria degli IPS, sebbene a noi non interessi la parte di prevenzione degli attacchi ma solamente quella di rilevamento, mentre il terzo è uno sniffer. Verrà approfondito il ruolo che questi strumenti svolgono nell'ambito della sicurezza informatica e seguirà una spiegazione riguardo le motivazioni che mi hanno portato a decidere di usare questi strumenti in particolare.

2.2 Intrusion Detection System

Gli Intrusion Detection Systems (IDS) sono tra gli strumenti più utilizzati nell'ambito della sicurezza informatica e se usati insieme ad altre tipologie di strumenti, come i firewall, forniscono una protezione più completa. Tutti questi strumenti ci permettono di eseguire un monitoraggio continuo della rete, con l'obiettivo di identificare per tempo possibili attacchi alle macchine o alla rete informatica. Lo scopo dei firewall consiste nel filtrare i pacchetti di dati che transitano in entrata e in uscita dalla rete locale, secondo un insieme di regole predefinite; gli IDS, invece, analizzano i pacchetti di dati e i comportamenti che essi generano, anche all'interno della rete locale stessa. Tipicamente, gli IDS sono composti da diversi sensori posizionati in punti strategici della rete, i quali raccolgono dati e informazioni e le comunicano ad un server centrale che ne esegue un'analisi secondo delle regole in modo da poter individuare possibili "anomalie". Per definizione, un IDS è un sistema di rilevazione perciò, nel caso in cui si verifichi un attacco o un comportamento sospetto, questi strumenti non eseguono azioni correttive, bensì si limitano ad agire passivamente inviando una notifica secondo le modalità stabilite; successivamente sarà compito dell'amministratore di rete quello di stabilire in che modo procedere. Questa modalità di analisi è particolarmente soggetta al rischio di un numero elevato di "falsi positivi", perciò, al fine di ridurre questo numero, l'intervento umano è quasi sempre necessario per fissare delle regole che determinino quali comportamenti siano leciti e quali no. Col passare del tempo, queste regole dovranno essere aggiornate sia dall'amministratore sia dagli algoritmi di "apprendimento automatico" del sistema, in modo da poter tenere in considerazione anche nuove tipologie di attacchi. Ora andremo a vedere nel dettaglio due di questi strumenti di difesa che sono stati entrambi utili nell'analisi del traffico durante gli esperimenti effettuati: *Suricata* e *Zeek*.

2.2.1 Suricata

Suricata è software di Intrusion Detection and Prevention System, progettato e sviluppato da Victor Julien, Matt Jonkman e William Metcalf. La prima versione del software è stata rilasciata nel 2009, in accordo con la direzione della Open Information Security Foundation (OISF), una fondazione no-profit gestita dalla comunità e organizzata per costruire un motore IDS/IPS di nuova generazione. Inoltre, la fondazione OISF ha preso l'impegno di mantenere *Suricata* open source per sempre.

Suricata può essere considerato il successore di Snort, un altro software IDPS nato nel 1998 che ha posto le basi per la creazione di software di reportistica e gestione validi tutt'oggi nell'uso di *Suricata*. Proprio come il suo predecessore, anche *Suricata* è scritto il linguaggio C e possiede la caratteristica di essere snort-rule compliant, ovvero supporta l'uso di regole che rispettano le modalità di scrittura e di formattazione proprie di Snort; in questo modo la transizione da un sistema all'altro viene decisamente semplificata. Oltre al fatto di essere completamente open source, le

Figura 2.1. Logo di *Suricata*.

altre peculiarità che rendono *Suricata* un software di spicco sono il supporto al multithreading e la possibilità di effettuare l'analisi del traffico di rete anche offline, mediante pcap.

Una parte fondamentale nell'uso di questo IDPS è la gestione delle regole. Nella maggior parte delle occasioni, gli utenti usano il set di regole già esistente all'interno di *Suricata*, tuttavia è sempre possibile procedere alla creazione di regole personalizzate al fine di integrare o sostituire quelle già presenti. Il formato delle regole consiste in tre parti fondamentali:

- un'*azione* che determina ciò che succederà nel caso in cui si abbia un riscontro con la regola;
- un'*intestazione* che definisce il protocollo, gli indirizzi IP, le porte e la direzione della regola;
- le *opzioni* che definiscono le specifiche della regola.

Le principali azioni tra cui scegliere sono: generare un avviso (*alert*), fermare ulteriori controlli sul pacchetto (*pass*), scartare il pacchetto (*drop*)e inviare un messaggio di errore al mittente del pacchetto (*reject*). Mentre nell'intestazione è possibile scegliere tra quattro protocolli base: tcp, udp, icmp e ip, oppure anche tra altri protocolli di livello applicazione come, ad esempio, http, ftp, tls o dns. L'ultima parte della regola è costituita dalle opzioni, le quali vengono racchiuse tra le parentesi e separate tra loro dal punto e virgola. Alcune opzioni hanno a loro volta delle impostazioni (come *msg*), mentre altre sono semplicemente delle parole chiave (*nocase*) a cui non è necessario aggiungere nulla. Un esempio di regola può essere:

- `alert tls any any -> any any (msg:‘‘Match bytes in TLS cert"; tls.certs; content: ‘‘|06 09 2a 86|"; sid:2230032;)`

Questa regola produce un avviso ogni qualvolta, in una connessione TLS tra una qualsiasi sorgente e una qualsiasi destinazione, viene trasmesso un certificato TLS che contiene al suo interno i byte *06 09 2a 86*. La parte finale della regola indicata con *sid* è solamente un codice di riferimento da attribuire manualmente alla regola.

Per poter creare delle regole personalizzate è necessario prima creare un nuovo file al percorso */etc/suricata/rules/* e in seguito è possibile scrivere al suo interno le nostre regole. Una volta che abbiamo concluso la parte di creazione delle regole, è necessario aggiornare il file di configurazione di *Suricata* (*/etc/suricata/suricata.yaml*) affinché sia incluso il file delle regole personalizzate e abilitare la creazione del relativo file di log.

2.2.2 Zeek

Come alternativa a *Suricata*, abbiamo preso in considerazione un tool che si chiama *Zeek*. Si tratta di una piattaforma software open source per l'analisi del traffico di rete e il monitoraggio della sicurezza. Questo strumento è stato sviluppato agli inizi degli anni '90 da Vern Paxson sotto il nome di "Bro", in omaggio al "Big Brother" presente nella novella "1984" di George

Figura 2.2. Logo di *Zeek*.

Orwell, e solo successivamente, nel 2018, gli è stato cambiato il nome in *Zeek*. *Zeek* non ha le stesse caratteristiche di un dispositivo di sicurezza attivo come può essere un firewall o un Intrusion Prevention System (IPS), ma più che altro viene utilizzato come Network Security Monitor (NSM) in modo da poter condurre indagini su possibili attività sospette o dannose all'interno della rete.

Una delle caratteristiche principali che ci offre questo software è l'esteso insieme dei file di log, per mezzo dei quali ha la possibilità di descrivere le attività di rete. Questi file di log vengono prodotti non solo per ogni connessione presente sulla rete, ma per quasi ogni protocollo di rete, di trasporto e applicazione. Perciò, all'interno di questi file è possibile trovare le informazioni riguardo a tutte le sessioni HTTP, le richieste DNS con le relative risposte, i certificati SSL, il contenuto chiave delle sessioni SMTP e molto altro ancora. *Zeek* produce tutte queste informazioni in un formato human-readable, come ad esempio il formato JSON, il quale è adatto anche per effettuare una successiva analisi utilizzando strumenti esterni. Inoltre il tool è dotato di alcune funzionalità per una serie di analisi di rilevamento, tra cui l'estrazione di file da sessioni HTTP, il rilevamento di malware, la segnalazione di versioni vulnerabili di software, il rilevamento degli attacchi SSH a forza bruta oppure la convalida delle catene di certificati SSL.

In breve, *Zeek* è ottimizzato per l'interpretazione del traffico di rete e la generazione di file di log basati su quel traffico, tuttavia non è così adatto per gli utenti che cercano o hanno bisogno di un approccio basato sul rilevamento di regole, tipico di alcuni IDS come *Suricata*.

2.2.3 Confronto Suricata/Zeek

I due tool che abbiamo visto in questa sezione, *Suricata* e *Zeek*, forniscono due differenti tipi di protezione di rete, tuttavia entrambe possono essere utili per il rilevamento di minacce conosciute o sconosciute.

Suricata è uno dei migliori motori di rilevamento delle minacce basato su regole che si trovi in circolazione, consente di identificare rapidamente un gran numero di minacce e permette l'implementazione di regole aggiuntive, per quando ne vengono scoperte di nuove. Il tool è basato su un'architettura multi-thread, il che consente di effettuare l'ispezione del traffico ad alte prestazioni e una rapida elaborazione di grosse quantità di regole, rispetto al grande volume del traffico di rete. Inoltre continua ad essere compatibile con le regole basate su Snort e supporta il linguaggio di scripting LUA, permettendo quindi agli utenti di poter creare regole per la rilevazione di minacce molto più complesse.

Zeek, invece, effettua il monitoraggio dei flussi di traffico e produce in output dei file di log che registrano tutto ciò che lo strumento comprende sull'attività di rete e su altri metadati, molto utili per l'analisi e la comprensione del comportamento delle rete. Molti di questi metadati che vengono prodotti da *Zeek* erano già disponibili precedentemente dai dati di acquisizione dei pacchetti (PCAP), tuttavia *Zeek* fornisce la possibilità di cercarli, indicizzarli e interrogarli in modi che prima, con PCAP, non erano possibili. Il linguaggio di programmazione di *Zeek*, strutturato in modo simile al C++, può essere utilizzato per il calcolo di statistiche, per eseguire corrispondenze di espressioni regolari e per la personalizzazione dell'interpretazione dei metadati in base ad esigenze specifiche dell'utente.

Dunque, *Suricata* si dimostra molto più efficiente di *Zeek* nel monitoraggio del traffico per le minacce note e nella generazione di avvisi quando queste vengono rilevate. Inoltre, le informazioni sulle nuove e più recenti minacce sono spesso disponibili in un formato che risulta compatibile con *Suricata*, dando all'utente la possibilità di implementare immediatamente nuove regole prima che queste minacce possano causare dei danni. *Zeek* invece fornisce un volume di dati ad alta qualità

nettamente maggiore di *Suricata*, il che fornisce una visibilità e un contesto completi del traffico di rete, consentendo il rilevamento delle anomalie e la caccia alle minacce.

L'uso in combinazione di *Suricata* e *Zeek* è decisamente molto efficace per la ricerca di minacce. Si potrebbero, ad esempio, usare insieme suddividendosi i compiti: *Suricata* avrebbe la possibilità di generare un avviso nel momento in cui il sistema viene compromesso, mentre *Zeek*, registrando le connessioni prima e dopo il verificarsi dell'incidente, permetterebbe un'analisi al fine di determinare se altre comunicazioni rafforzano o aiutano a dare una spiegazione dell'incidente. Idealmente, è possibile utilizzare *Suricata* per un'identificazione più rapida degli attacchi, le cui regole sono prontamente disponibili per essere implementate, e utilizzare *Zeek* in modo da fornire i metadati utili per una corretta valutazione degli avvisi generati da *Suricata*.

2.3 Sniffer

Uno sniffer, o analizzatore di pacchetti, è uno strumento che viene utilizzato per la registrazione del traffico in entrata e in uscita da un computer connesso alla rete. L'obiettivo principale di questo strumento è il monitoraggio e l'analisi del traffico al fine di rilevare eventuali problemi e mantenere il sistema efficiente. É possibile registrare tutti i dati che transitano sull'intera rete o solamente su di un segmento di nostro interesse, così da poter osservare qualsiasi dettaglio dell'interazione tra due o più nodi, qualunque sia il protocollo di comunicazione utilizzato. Possiamo considerare due branchie differenti di sniffer:

- prodotti commerciali rivolti agli amministratori di rete per la manutenzione interna delle reti;
- prodotti dell'underground informatico che possiedono più funzioni rispetto ai tool commerciali.

Questi ultimi, paragonabili ad un analizzatore di rete, offrono all'utente più operazioni rispetto al semplice ascolto e archiviazione del traffico: eseguono un'analisi statistica sul traffico e sulla composizione dei pacchetti.

Le tipiche funzioni di uno sniffer sono le seguenti:

- filtraggio e conversione dei dati e dei pacchetti in un formato human-readable;
- analisi dei difetti, della qualità e della portata della rete;
- creazione di log riguardanti il traffico di rete;
- scoperta delle intrusioni mediante l'analisi dei log;
- ricerca di username, password o token inviati in chiaro.

L'ultima funzione tra quelle elencate viene spesso utilizzata dagli hacker che vogliono rubare informazioni dai dati non crittografati scambiati da due parti, in modo da poter eseguire in seguito alcuni attacchi.

Sebbene la maggior parte degli sniffer attuali siano software, gli sniffer hardware svolgono ancora un ruolo importante nella risoluzione dei problemi di rete, permettendo di registrare e analizzare anche i possibili errori hardware come quelli di controllo di ridondanza ciclico (Cyclic Redundancy Check o CRC), problematiche relative alla tensione o al cavo e molti altri ancora. Tuttavia, entrambe le tipologie di sniffer sono costituite da due elementi principali molto importanti:

capture driver: il componente che cattura il traffico della rete, lo filtra e lo memorizza nel buffer;

buffer: il componente dove vengono memorizzati i dati, secondo due possibili modalità:

- fino al momento in cui viene completamente riempito;

- "round robin", ovvero sostituendo i dati più vecchi con quelli più recenti.

L'origine della parola "sniffer" è dovuta al primo programma di questo tipo che è stato creato: "The Sniffer Network Analyzer" sviluppato dalla Network Associates Inc. Ora invece questa parola è di uso comune nel mondo dell'informatica e si riferisce a tutti i programmi che implementano le funzioni tipiche che sono state descritte precedentemente.

Se guardiamo alle offerte presenti al momento non è possibile non citare *Wireshark*, precedentemente noto come Ethereal: uno sniffer completamente gratuito e open source. Proprio questo software è quello su cui abbiamo fatto affidamento per alcuni dei miei esperimenti e che sarà presentato più nel dettaglio nella prossima sezione.

2.3.1 Wireshark

Questo progetto nasce nel 1998 da Gerald Combs con il nome di Ethereal e prende il nome di *Wireshark* solo nel 2006, in seguito ad un mancato raggiungimento di un accordo con il detentore dei diritti del nome "Ethereal". Si tratta di un software completamente open source, il quale viene supportato economicamente grazie ai contributi volontari di tantissimi esperti di networking di tutto il mondo. Al momento, *Wireshark* risulta essere l'analizzatore di protocolli di rete più diffuso e più utilizzato al mondo.

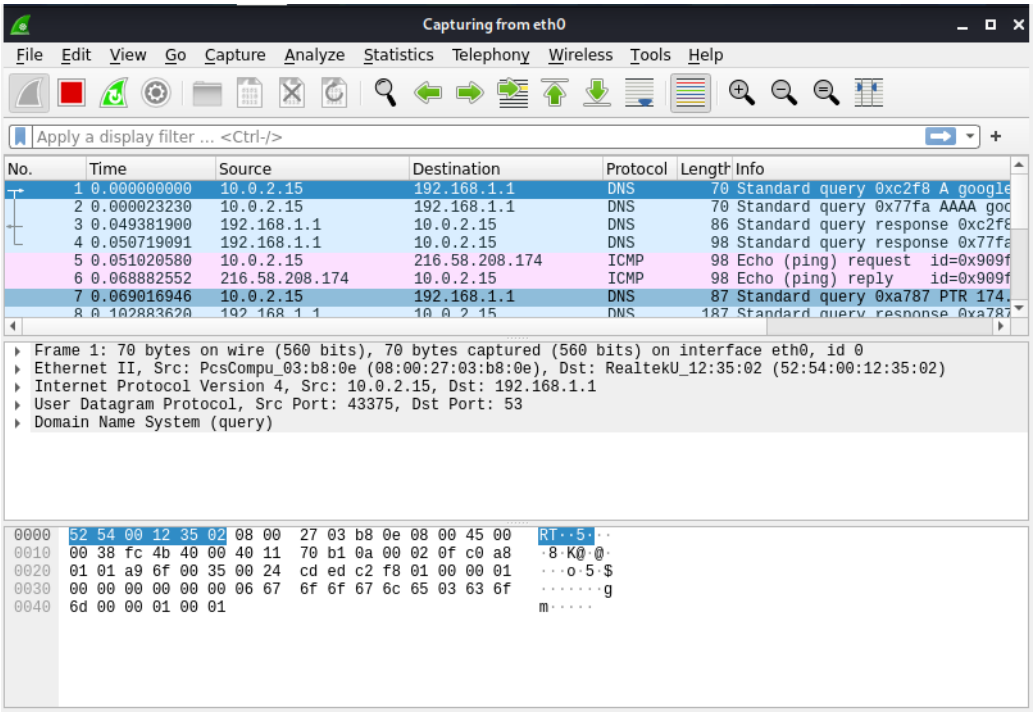

Figura 2.3. Interfaccia a tre riquadri di *Wireshark*.

Questo software permette di effettuare un'ispezione approfondita di centinaia di protocolli, la cui lista viene aggiornata continuamente, e ci consente di vedere cosa sta succedendo sulla nostra rete a livello microscopico. Viene effettuata la cattura del traffico in tempo reale con la possibilità di memorizzare i dati ed eseguire un'analisi più approfondita offline, il tutto sfruttando la tipica interfaccia a tre riquadri ("three-pane interface"). Questi sono disposti in relazione gerarchica: dal meno dettagliato, che ci mostra informazioni come indirizzi IP sorgente e destinazione, protocollo, tipologia di pacchetto, al più dettagliato, il quale ci mostra direttamente i bit che compongono il pacchetto. Inoltre, è possibile utilizzare il software sulla maggior parte dei sistemi Unix (inclusi Linux, Solaris, FreeBSD e NetBSD), su macOS e sui sistemi Microsoft Windows. In confronto ad uno sniffer comune, oltre alle tipiche funzioni, *Wireshark* possiede anche le seguenti:

- i dati acquisiti possono essere sfogliati mediante interfaccia grafica (GUI) oppure da riga di comando con l'utility TShark;
- permette di eseguire una ricca analisi di VoIP (Voice over IP);
- legge e scrive molti formati di acquisizione diversi: tcpdump, Pcap NG, Cisco Secure IDS iplog, Microsoft Network Monitor e altri reperibili sul sito ufficiale;
- cattura dei file compressi con *gzip* e decompressione al volo;
- i dati sul traffico possono essere letti da: Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token ring, Frame relay e FDDI;
- viene supportata la decifrazione per molti protocolli tra cui IPsec, ISAKMP, Kerberos, SSL/TLS, WEP e WPA/WPA2;
- è possibile utilizzare filtri di visualizzazione per colorare o evidenziare selettivamente le informazioni per un'analisi intuitiva e veloce;
- gli output possono essere esportati in XML, CSV o plaintext.

Dunque, anche se durante gli esperimenti sono state utilizzate solo un sottoinsieme delle funzioni a disposizione, *Wireshark* si è rivelato uno strumento molto utile alla realizzazione del lavoro, permettendomi di comprendere al meglio cosa stava succedendo sulla rete durante i tentativi di eseguire un attacco.

2.4 Configurazione di Zeek

2.4.1 Prerequisiti

Prima di poter effettuare l'installazione di *Zeek*, è necessario verificare che le dipendenze richieste dal tool siano installate sulla nostra macchina. Per poter essere installato ed eseguito correttamente, *Zeek* necessita della seguente lista di tool e librerie:

- Libpcap;
- OpenSSL;
- BIND8;
- Libz;
- Bash;
- Python 3.5 o superiore;
- CMake 3.5 o superiore;
- Make;
- Compilatore C/C++ con supporto per C++17;
- SWIG;
- Bison 2.5 o superiore;
- Flex.

Per verificare la presenza delle dipendeze ed, eventualmente, procedere all'installazione di quelle mancanti, è possibile utilizzare il comando appropriato, in base al sistema operativo presente sulla nostra macchina:

RPM/RedHat-based Linux :

```
$ sudo yum install cmake make gcc gcc-c++ flex bison libpcap-devel openssl-devel
python3 python3-devel swig zlib-devel
```

DEB/Debian-based Linux :

```
$ sudo apt-get install cmake make gcc g++ flex bison libpcap-dev libssl-dev python3
python3-dev swig zlib1g-dev
```

macOS :

```
$ xcode-select --install
```

Questo comando non installa direttamente le dipendenze, ma serve per installare *Xcode* o *Command Line Tools*. In seguito sarà possibile scaricare i package cmake, swig, swig-python, openssl e bison da *Homebrew* o *MacPorts*.

Una volta terminata l'installazione delle dipendenze richieste sarà possibile procedere con l'installazione vera e propria di *Zeek*.

2.4.2 Installazione

Per poter installare *Zeek*, prima è necessario scaricare manualmente i pacchetti sorgente del tool dal sito ufficiale. Nella sezione textitGet Zeek/Downloads troviamo i pacchetti per le ultime versioni di *Zeek*, perciò è sufficiente scaricarli nel formato *.tar.gz* per il sistema scelto. Una volta completato il download è necessario decomprimere l'archivio, dunque ci spostiamo nella cartella in cui è stato scaricato ed eseguiamo da linea di comando:

- `tar -xvzf zeek-4.1.1.tar.gz`

Eventualmente sostituendo il numero di versione con quello che si è scelto di scaricare. Terminata la decompressione, entriamo nella cartella *zeek-4.1.1* da cui è possibile eseguire gli ultimi comandi per configurare ed installare *Zeek*:

- `./configure`
- `make`
- `make install`

Il programma verrà installato al percorso di default */usr/local/zeek*, tuttavia è possibile scegliere un percorso differente a cui installare il tool: è sufficiente indicarlo al momento dell'esecuzione del primo comando `./configure` mediante l'utilizzo dell'opzione `--prefix` seguito dal percorso che abbiamo scelto.

Dopo qualche momento di attesa, il programma dovrebbe essere stato installato correttamente. Prima di poter avviare *Zeek* in modo che inizi a monitorare la nostra rete, è necessario effettuare una configurazione manuale in modo da determinare quali script debba caricare prima della sua esecuzione. Ogni script contiene le regole di comportamento del programma: gli indicano in modo specifico cosa controllare all'interno del traffico e a quali evento è necessario che reagisca. Dunque, ora vedremo come eseguire la configurazione degli script aggiungendo a quelli di default anche un nostro script personalizzato.

2.4.3 Configurazione degli script

Per permettere a *Zeek* di rendere al meglio delle sue possibilità, è necessario conoscere e saper utilizzare tutte le funzionalità che ci mette a disposizione. Una delle più importanti è la possibilità di scrivere script personalizzati per determinare nuove regole, mediante cui intercettare il traffico. Inoltre, oltre a specificare al software quali informazioni è necessario recuperare dal traffico di rete, dobbiamo anche determinare come queste informazioni debbano essere salvate negli appositi file di log. Una volta che avremo personalizzato questi aspetti sarà possibile cominciare ad utilizzare *Zeek* e procedere ad una successiva analisi dei file di log che saranno creati.

La creazione di uno script si basa su tre concetti fondamentali:

Stream: un *log stream* corrisponde ad un singolo file di log. Definisce l'insieme dei campi di cui è composto un log con i loro nomi e tipi. Ad esempio, esistono lo stream *http*, per registrare le attività HTTP, e lo stream *ssl*, per registrare il traffico SSL/TLS.

Filtri: per ogni stream esiste un insieme di filtri relativo ad esso, il quale specifica quali informazioni è necessario salvare e in che modo. C'è la possibilità di aggiungere filtri addizionali per salvare solo un sottoinsieme delle informazioni dei log, salvare le informazioni in un luogo diverso (di default sono salvate sul disco) o impostare un intervallo personalizzato di rotazione. Nel caso vengano rimossi tutti i filtri da uno stream, allora viene disabilitato l'output per quello stream.

Writer: per ogni filtro esiste un *writer* che definisce il formato di output per le informazioni che devono essere salvate. Il *writer* di default è quello ASCII, che permette di produrre file ASCII separati mediante *tab*. Sono disponibili altri *writer*, come per l'output binario o l'accesso diretto ad un database.

Per la creazione di uno stream, è necessario seguire alcune semplici regole:

- deve essere definito un tipo *record* che comprende tutti i campi che dovranno essere salvati;
- deve essere definito un ID del *log stream* in modo che lo identifichi in modo univoco;
- un *log stream* deve essere creato utilizzando la funzione `Log::create_stream`;
- quando i dati da memorizzare sono disponibili, deve essere chiamata la funzione `Log::write`.

```
1 module Test;
2
3 export {
4     redef enum Log::ID += { LOG };
5     type Info: record {
6         ts: time        &log;
7         id: conn_id     &log;
8         service: string &log &optional;
9         missed_bytes: count &log &default=0;
10    };
11 }
12
13 redef record connection += {
14    test: Info &optional;
15 };
16
17 event zeek_init() &priority=5
18    {
19    Log::create_stream(Test::LOG, [$columns=Info, $path="test"]);
20    }
21
22 event connection_established(c: connection)
23    {
24    local rec: Test::Info = [$ts=network_time(), $id=c$id];
25
26    c$test = rec;
27
28    Log::write(Test::LOG, rec);
29    }
```

Figura 2.4. Script di esempio per il software *Zeek*.

Nella Fig. 2.4 è presente un esempio di uno script base per la creazione di uno nuovo file di log. Come è possibile osservare, per prima cosa viene creato un nuovo modulo, che noi abbiamo chiamato *Test* che ci permette di creare un nuovo *log stream*. All'interno della sezione *export*, per prima cosa, viene creato un nuovo ID per il nuovo stream che, per convenzione, è chiamato *LOG* e, subito dopo, viene definito il tipo di record che conterrà i dati da memorizzare. É possibile notare come ogni campo abbia l'attributo `&log`, senza il quale il campo non comparirebbe all'interno del log di output. Inoltre, sono presenti altri due attributi chiamati `&optional`, il quale

indica che a quel campo potrebbe non essere assegnato alcun valore all'interno del file di log, e `&default`, il quale definisce un valore di default da attribuire a quel determinato campo. Alla riga 13 è presente una parte opzionale che ci permette di aggiungere un nuovo campo al record *connection*, in questo modo i dati che stiamo memorizzando (i record `Info`) saranno facilmente accessibili in molteplici gestori di eventi, di cui abbiamo due esempi nella parte finale. L'evento chiamato *zeek_init()* si riferisce a quando noi avviamo effettivamente *Zeek* mediante l'apposito comando, a questo punto viene creato lo stream utilizzando la funzione `Log::create_stream` e viene automaticamente aggiunto il filtro di default che impone di salvare i file di log su disco. L'evento chiamato *connection_established()* invece si attiva quando viene catturato il pacchetto SYN-ACK, proveniente dal risponditore, all'interno di un handshake TCP. Al suo interno, alla riga 26, viene memorizzata una copia dei dati presenti nel record *connection* in modo che altri gestori di eventi possano accedervi, e successivamente viene chiamata la funzione `Log::write`, la quale porta alla memorizzazione dei dati nel file di log specificato.

I filtri comandano due aspetti nella produzione dei file di log: eseguono un controllo su quali voci del log di stream devono essere scritte e definiscono le modalità con cui implementare la scrittura di questi file di log. Quest'ultimo aspetto viene gestito specificando un *writer* di log che implementa le operazioni di scrittura, come può essere il *writer* ASCII. Nel momento in cui uno stream viene creato gli viene automaticamente assegnato un filtro di default, il quale può essere rimosso, rimpiazzato oppure è possibile aggiungere altri filtri allo stream. Tutto questo è possibile farlo mediante l'utilizzo delle due funzioni `Log::add_filter` o `Log::remove_filter`. Ogni filtro possiede un nome univoco, limitato allo stream a cui appartiene, in modo che tutti i filtri assegnati ad uno stream possiedano nomi differenti. Utilizzare la funzione per aggiungere un nuovo filtro con un nome che già esiste all'interno dello stream, porterà alla sostituzione dello stream esistente.

```
event zeek_init()
{
        local f = Log::get_filter(Conn::LOG, "default");
        f$path = "myconn";
        Log::add_filter(Conn::LOG, f);
}
```

Figura 2.5. Esempio dell'uso dei filtri in uno script *Zeek*.

Normalmente, il nome del file di log per un dato *log stream* viene determinato quando è creato lo stream, a meno che non ne venga specificato uno differente mediante un filtro. Nella Fig. 2.5 è possibile vedere un piccolo esempio di codice permette di sostituire il filtro di default con uno nuovo, il quale specifica un nuovo valore per il campo *path* permettendo di cambiare il nome del file di log. In questo caso specifico il nome *conn.log* viene sostituito dal nome *myconn.log*.

Ogni filtro possiede un *writer* che, se non specificato altrimenti nel momento in cui viene aggiunto un filtro allo stream, corrisponde al writer di default ASCII. Esistono due modalità per specificare un writer differente da quello di default: una consiste nel cambiare il writer di default per tutti i filtri, mentre l'altra permette di specificare un writer specifico per un unico filtro. Nel primo caso è sufficiente ridefinire l'opzione `Log::default_writer` impostando un nuovo writer che sarà utilizzato come default, nel secondo caso invece è necessario modificare il valore del campo *writer* del filtro prescelto.

SQLite è un sistema di database SQL semplice e ampiamente utilizzato, il cui uso permette a *Zeek* di accedere e scrivere i dati in un formato facile da utilizzare in interscambio con altre applicazioni. Nell'esempio della Fig. 2.6 viene mostrato come creare ed aggiungere un nuovo filtro SQLite per il log relativo alla connessione *conn*. Se non esiste ancora, *Zeek* creerà il file database */var/db/conn.sqlite*, al cui interno creerà una tabella chiamata *conn* alla quale incomincerà ad aggiungere informazioni sulle connessioni. Questa modifica aggiunge un nuovo filtro a fianco di quello di default ASCII, ma non lo elimina, perciò il file di log *conn.log* verrà comunque creato

```
event zeek_init()
{
        local filter: Log::filter =
                [
                $name="sqlite",
                $path="/var/db/conn",
                $config=table(["tablename"] = "conn"),
                $writer=Log::WRITER_SQLITE
                ];

        Log::add_filter(Conn::LOG, filter);
}
```

Figura 2.6. Esempio dell'uso dei writer in uno script *Zeek*.

e aggiornato. Se non vogliamo che ciò accada possiamo rimuovere il filtro di default in questo modo:

- `Log::remove_filter(Conn::LOG, ‘‘default’’);`

Una volta che il nostro script per la creazione dei file di log è stato creato, è necessario effettuare le ultime modifiche. Innanzitutto dobbiamo spostare il file *.zeek*, contenente il nostro script, al percorso */usr/local/zeek/share/zeek/site*: si tratta di una directory contenente un file di configurazione e adibita al compito di contenere tutti gli script personalizzati dell'utente, senza il pericolo che questi vadano persi con i futuri aggiornamenti del software *Zeek*. Una volta effettuato lo spostamento è necessario modificare il file *local.zeek*, presente all'interno della directory, indicando l'opzione per caricare ed utilizzare anche il nostro script, perciò apriamo il file e al fondo inseriamo una linea contenente il seguente codice:

- `@load site/filename.zeek`

Dove è necessario sostituire *filename.zeek* con il nome che abbiamo dato al nostro script.

É necessario ripetere l'operazione per ogni nuovo script che viene inserito nella directory e che vogliamo che *Zeek* utilizzi. Una volta eseguite tutte queste operazioni è possibile far partire l'esecuzione del software, il quale caricherà tutti gli script indicati nel file di configurazione e comincerà ad analizzare il traffico seguendo le regole e salvando le informazioni nelle modalità indicate sia dagli script presenti di default sia dagli script personalizzati che abbiamo inserito noi stessi.

Capitolo 3

Strumenti di attacco

All'interno di questo capitolo verranno presentati tutti i tool di attacco utilizzati all'interno degli esperimenti che vedremo alla fine. Di questi strumenti verranno messe in mostra le loro funzionalità e cosa è possibile fare con ognuno di essi. La parte relativa all'installazione e alla configurazione dei tool sarà invece approfondita nel capitolo successivo.

3.1 Ettercap

Ettercap è un software open source che fornisce una serie di funzionalità utili per lo sniffing di connessioni in tempo reale, il filtraggio di contenuti e, più in generale, per l'analisi di rete e degli host. Inoltre, risulta essere in grado di intercettare tutto il traffico di rete all'interno del dominio di collisione, permettendo l'esecuzione di attacchi di tipo Man in The Middle (MITM). Il software funziona su diversi sistemi operativi come Linux, macOS, Solaris e Microsoft Windows, sebbene tutte le funzionalità siano disponibili unicamente sulle piattaforme Unix. Per quanto riguarda il funzionamento del software, ci viene data la possibilità di scegliere tra 3 interfacce utente (text mode, curses e GTK) e 4 differenti modalità di funzionamento:

IP-based: il filtraggio dei pacchetti avviene mediante il controllo degli indirizzi IP sorgente e destinazione;

MAC-based: il filtraggio dei pacchetti avviene mediante il controllo degli indirizzi MAC;

ARP-based: si procede all'avvelenamento delle tabelle ARP per effettuare lo sniffing sulla LAN commutata tra due host;

PublicARP-based (o SmartARP-based): si procede all'avvelenamento delle tabelle ARP per effettuare lo sniffing sulla LAN commutata da un host vittima a tutti gli altri host.

Tra le funzionalità più rilevanti che *Ettercap* ci offre e che possiamo sfruttare possiamo citare: la modifica dei pacchetti in transito sulla rete, la possibilità di aggiungere o rimuovere pacchetti all'interno di una sessione di rete, l'analisi delle connessioni HTTPS, la decifrazione di username e password per diversi protocolli di rete (Telnet, FTP, SSH1, POP, IMAP e altri) ed infine la scansione passiva della rete (ovvero senza l'invio di pacchetti) con la possibilità di distruggere le connessioni. Il software non richiede una configurazione iniziale, ma è bensì possibile impostare tutte le opzioni di cui abbiamo bisogno direttamente da riga di comando oppure collegandovi un file di configurazione esterno.

Sebbene offra buone funzionalità riguardo gli attacchi Man-In-The-Middle, il prossimo tool che andremo a vedere risulta essere più avanzato, in particolare per quanto riguarda le connessioni HTTPS. Perciò, anche se inizialmente i miei esperimenti prevedevano l'uso di *Ettercap*, alla fine si è rivelato più performante utilizzare un altro programma.

3.2 mitmproxy

La prima versione di *mitmproxy* viene pubblicata nel 2010 da Aldo Cortesi e Maximilian Hils, si tratta di un insieme di tool che fornisce un proxy interattivo che può essere usato per intercettare, ispezionare, modificare e replicare il traffico web come HTTP/1, HTTP/2, WebSocket o qualsiasi altro protocollo protetto da SSL/TLS. Con questi strumenti è possibile decodificare diversi tipi di messaggi (da HTML a Protobuf), intercettare messaggi specifici, modificarli prima che raggiungano la destinazione prevista oppure replicarli in seguito verso un client o un server. Quando parliamo di *mitmproxy* ci riferiamo a uno dei seguenti tre strumenti:

mitmproxy: si tratta un proxy di intercettazione interattivo, compatibile con SSL/TLS e provvisto di un'interfaccia a console;

mitmweb: si tratta di una interfaccia basata su web per *mitmproxy*;

mitmdump: si tratta della versione di *mitmproxy* basata su riga di comando.

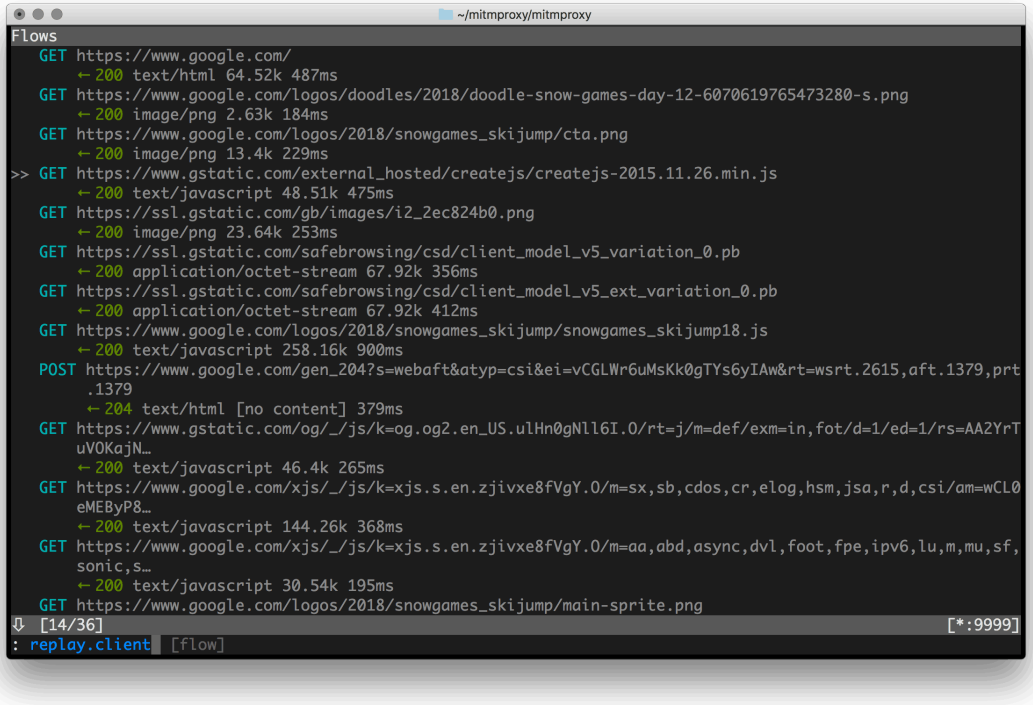

Figura 3.1. Interfaccia a console di *mitmproxy*.

mitmproxy è uno strumento a console che permette all'utente di esaminare e modificare il traffico HTTP in modo interattivo. Differisce da *mitmdump* in quanto tutti i flussi sono mantenuti in memoria, il che lo porta a poter prelevare e manipolare campioni di piccole dimensioni. Con *mitmdump*, invece, l'utente ha a disposizione funzionalità molto simili a *tcpdump*, le quali gli consentono di visualizzare, registrare e trasformare il traffico HTTP in modo programmatico. Molto più simile a *mitmproxy* è *mitmweb*, l'interfaccia utente basata su web, che permette anch'essa l'analisi e la modifica del traffico HTTP in modo interattivo e mantenimento in memoria di tutti i flussi.

Come è possibile dedurre dal nome dell'insieme di tool, *mitmproxy* ci fornisce anche tutta una serie di funzionalità molto utili per l'esecuzione di attacchi MITM, in particolare: la possibilità di salvare intere conversazioni HTTP per poterle analizzare o replicare successivamente, replicare una conversazione HTTP lato client oppure le risposte HTTP dei server ed infine permette la generazione al volo di certificati SSL/TLS, l'intercettazione e la modifica al volo di richieste e risposte HTTPS.

3.3 TLS-Attacker

Ad aprile 2016 la Ruhr University Bochum, la Paderborn University e *Hackmanit GmbH* sviluppano e rilasciano la prima versione di *TLS-Attacker*, un framework basato su Java per l'analisi delle librerie TLS. *TLS-Attacker* fornisce la possibilità di creare flussi di messaggi TLS personalizzati e modificare arbitrariamente il contenuto dei messaggi, al fine di analizzare e testare il comportamento delle implementazioni TLS. All'interno della struttura del framework sono presenti diversi moduli Maven tra cui citiamo: *TLS-Client*, un'applicazione client utile per collegarsi ad un server, *TLS-Server*, un'applicazione server che può essere usata per far partire un server, *TLS-Mitm*, un prototipo di un'applicazione per effettuare attacchi MITM, e *Attacks*, un'applicazione che contiene l'implementazione di alcuni test delle vulnerabilità e di alcuni attacchi conosciuti. Quest'ultima applicazione citata è quella che abbiamo sfruttato per compiere i nostri esperimenti, al fine di poter scovare delle vulnerabilità in alcuni server e successivamente poter eseguire alcuni attacchi. I vari attacchi per cui è possibile eseguire dei test con questo framework fanno parte di tre tipologie differenti:

- Attacchi crittografici: attacchi come Padding Oracle e le sue varianti Lucky13 e la CVE-2016-2107, l'attacco di Bleichenbacher, gli attacchi Invalid Curve e tutte le varianti dell'attacco POODLE;
- Attacchi state machine: gli attacchi Early ChangeCihperSpec e Early Finished;
- Attacchi overflow: l'attacco Heartbleed.

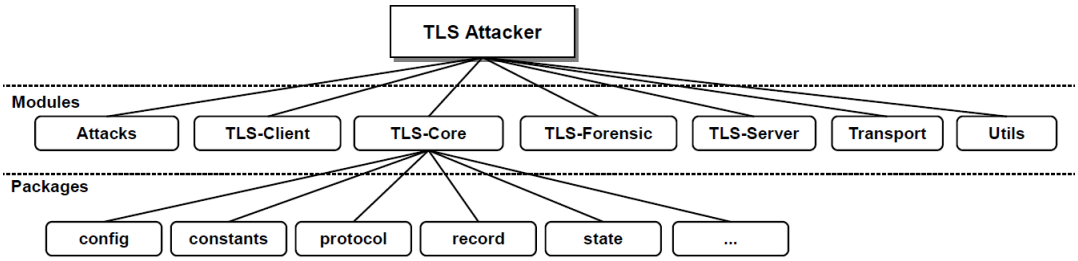

Figura 3.2. Struttura dei moduli all'interno di *TLS-Attacker*.

Mediante il lancio di codice opportuno da riga di comando, è possibile collegarsi ad un server e verificare se presenta delle vulnerabilità per l'attacco scelto. In seguito, se la vulnerabilità viene confermata, è possibile cominciare l'esecuzione dell'attacco vero e proprio tramite l'opzione *-executeAttack*. Purtroppo, ad oggi, dicembre 2021, tale opzione è presente per quasi tutti gli attacchi, tuttavia per la maggior parte di essi si tratta di una funzione wrapper dentro cui manca l'implementazione vera e propria. Il codice per eseguire effettivamente l'attacco è presente unicamente per l'attacco di Bleichenbacher e l'attacco Invalid Curve.

3.4 Metasploit framework

Nel 2003 H.D. Moore e Matt Miller presentarono il loro progetto denominato *Metasploit*, uno strumento di rete portatile basato sul linguaggio Perl e successivamente convertito completamente in linguaggio Ruby quattro anni dopo. Questo progetto forniva informazioni riguardo le vulnerabilità, semplificando le operazioni di penetration testing e lo sviluppo di sistemi per il rilevamento delle intrusioni. Nel 2009, la licenza del progetto viene acquistata da *Rapid7*, un'azienda fornitrice di servizi per la sicurezza informatica avente sede a Boston. Il progetto diventa parte della sezione che si occupa dello sviluppo degli IDS e degli expolit ai danni di macchine remote. Di questi strumenti, una parte risiede nel framework *Metasploit*, un sottoprogetto open source implementato direttamente all'interno del sistema operativo Kali Linux. Questo framework è costituito da diversi moduli, tra cui una serie di strumenti per sfruttare le vulnerabilità di un sistema, un

insieme di codici malevoli, dei shellcode (ovvero codici che sono programmati per attivarsi una volta all'interno della vittima), dei codificatori e dei software malevoli programmati a nascondersi al fine ottenere l'accesso ad una macchina. I passaggi principali per poter sfruttare le vulnerabilità di un sistema ed accedervi consistono nel:

- Scegliere e configurare l'exploit, ovvero il codice che sfrutta le vulnerabilità per penetrare in un sistema;
- Verificare che il sistema attaccato sia vulnerabile all'exploit;
- Scegliere e configurare il payload, ovvero il codice da eseguire una volta penetrati in un sistema;
- Scegliere la tecnica di codifica in modo da non essere rilevati dagli IPS;
- Eseguire l'exploit.

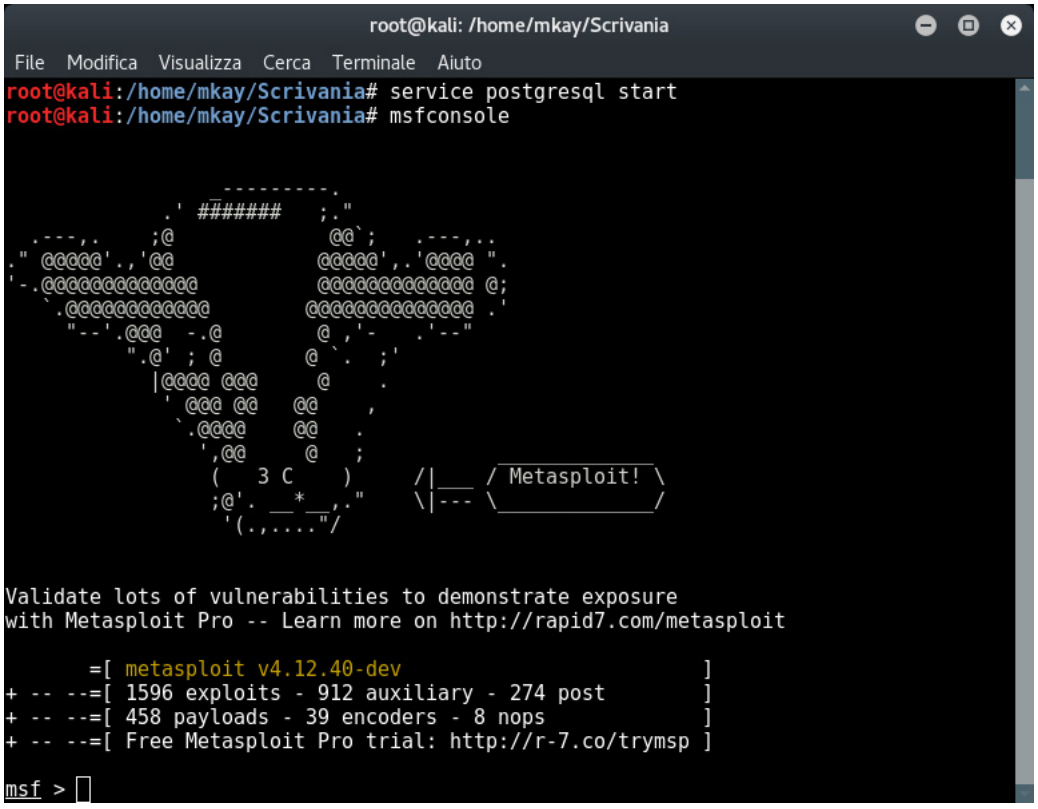

Figura 3.3. Interfaccia del framework *Metasploit*.

Le informazioni necessarie per la scelta dell'exploit e del payload, come la versione del sistema operativo o i servizi di rete installati sulla macchina, possono essere ottenuti mediante gli strumenti di OS fingerprinting e port scanning, forniti da alcuni software come Nmap. La raccolta degli exploit presente all'interno di *Metasploit* framework ne comprende più di 1700 e tutti fanno riferimento a molte piattaforme, tra cui le più famose sono: Android, Cisco, Java, JavaScript, Linux, Node.js, Unix e Windows.

Il continuo sviluppo di questo framework l'ha portato ad essere uno degli strumenti più utilizzati dagli analisti della sicurezza e dai pen-tester, permettendo di scovare vulnerabilità nascoste e di usare gli stessi metodi usati dagli hacker per infiltrarsi nelle reti e nei server.

3.5 padding-oracle-attacker e PadBuster

Il tool *padding-oracle-attacker*, scritto in TypeScript, è stato creato da Kishan Bagaria, mentre l'autore dello script Perl *PadBuster* è Brian Holyfield. Entrambi sono stati messi a disposizione su GitHub dai proprietari ed entrambi ci permettono di eseguire le stesse azioni: decrittare un ciphertext arbitrario, crittografare un plaintext arbitrario e determinare se un obiettivo risulta vulnerabile o meno, indicandoci inoltre in che modo differiscono le risposte del server nel momento in cui avviene un errore di decrittazione.

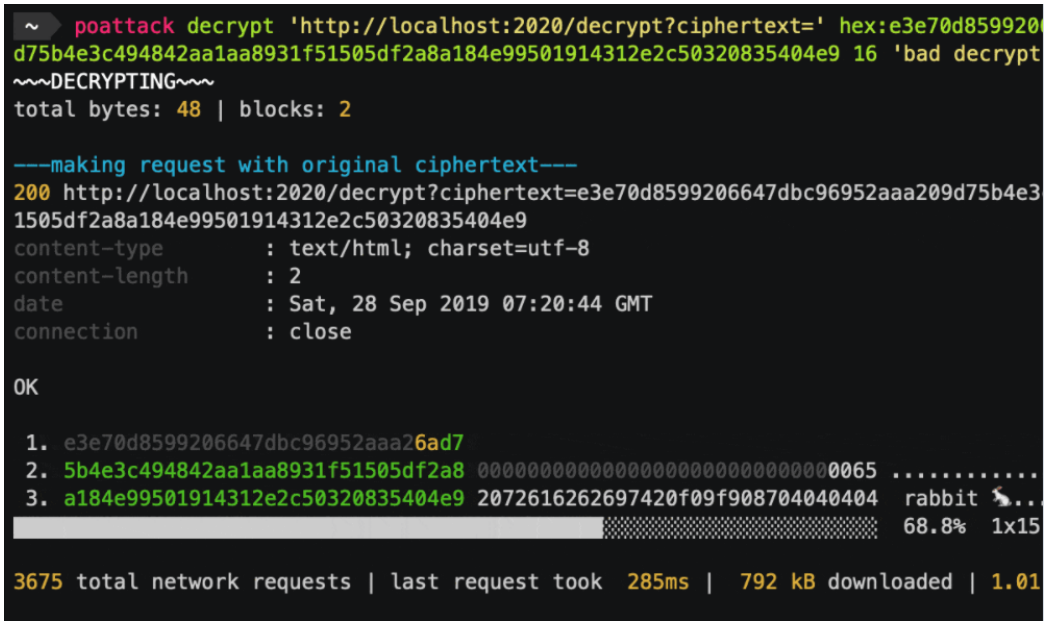

Figura 3.4. Interfaccia del tool *padding-oracle-attacker*.

Questi due strumenti è possibile eseguirli direttamente da riga di comando, è solamente necessario inserire i seguenti argomenti: l'URL della vittima, il ciphertext o il plaintext su cui vogliamo eseguire il test e la dimensione dei blocchi usati dall'algoritmo. Il tool *padding-oracle-attacker* richiede l'inserimento di un ulteriore argomento: il codice di errore che ci viene restituito quando la decrittazione sul server fallisce. Questo codice è possibile ottenerlo mediante una precedente analisi del serve vittima e la sua possibile vulnerabilità, che, come detto, ci fornirà informazioni su come varia la risposta del server in caso di errori. Lo script *PadBuster* è utilizzabile immediatamente appena scaricato e non necessita di ulteriori configurazioni, invece lo strumento *padding-oracle-attacker* ha bisogno del supporto di Node.js per funzionare correttamente, perciò è necessario verificarne la presenza prima di poter lanciare l'attacco, come descritto nell'Appendice A.4.

Gli esperimenti son stati eseguiti con entrambi i programmi, tuttavia il tool *padding-oracle-attacker* fornisce una interfaccia grafica migliore al fine di comprendere cosa sta succedendo durante l'attacco; perciò duranta la presentazione degli esperimenti verrà utilizzato questo tool.

3.6 Google Gruyere

Google Gruyere è un sito web, sviluppato proprio da Google, che simula i principi di un qualunque social network. Su di esso è possibile creare un profilo con nome, foto e descrizione, gestirlo e pubblicare anche alcuni brevi messaggi. La particolarità di questo sito web è la presenza di innumerevoli bug e vulnerabilità, questo perché lo scopo principale del sito non è fornire l'ennesimo servizio di social networking, ma bensì è quello di guidare l'utente alla ricerca di queste vulnerabilità ed instruirlo sia su come sfruttarle sia su come risolverle.

All'interno di questo sito le vulnerabilità sono divise per tipologia, in ogni sezione è possibile trovare una breve descrizione della vulnerabilità presente e un sfida da completare, come se fosse

Figura 3.5. Interfaccia del sito web *Google Gruyere*.

un gioco, al fine di trovare un istanza di quella vulnerabilità all'interno di *Google Gruyere*. Dunque, l'utente svolge il ruolo di hacker che trova e sfrutta questi bug di sicurezza e, lungo il percorso, impara a riconoscere in che modo può essere attaccata un'applicazione web, il tutto utilizzando delle comuni vulnerbilità, come quelle cross-site scripting (XSS) o le cross-site request forgery (XSRF). Impara inoltre come risolvere queste vulnerabilità e altri bug che possono avere impatto sulla sicurezza dell'applicazione, come gli attacchi DDOS o l'esecuzione di codice remoto.

Al di là dell'aspetto educativo in ambito di sicurezza informatica che offre *Google Gruyere*, nel mio lavoro di tesi questo sito si è rivelato utile al fine di testare i vari tool di attacco, in particolare per il tool *padding-oracle-attacker* presentato precedentemente. Il tool in questione presenta un problema: non funziona correttamente quando la comunicazione con un server che possiede un certificato SSL autofirmato, come risulta essere il server utilizzato nelle mie macchine virtuali. Per ovviare a questo problema, *Google Gruyere* si è dimostrato estremamente utile, permettendomi di poter simulare l'attacco contro un server reale che avesse un certificato SSL firmato da una CA.

Capitolo 4

Esecuzione degli attacchi

In quest'ultimo capitolo proveremo a mettere in pratica alcuni possibili attacchi al protocollo TLS visti precedentemente. Verrà inizialmente descritta la configurazione degli strumenti necessari per eseguire l'attacco e, successivamente, mostreremo l'esecuzione dell'attacco passo dopo passo e i risultati ottenuti da ciascuno di essi. Gli attacchi che che proveremo ad eseguire sono i seguenti:

- un attacco Man-In-The-Middle;
- un attacco Heartbleed;
- un attacco Padding Oracle.

4.1 Attacco MITM

L'attacco che abbiamo provato ad eseguire per primo è un semplice attacco Man-In-The-Middle, volto ad intercettare il traffico HTTPS di una comunicazione tra due parti. Normalmente HTTPS cifra tutto il traffico HTTP impedendo in questo modo, ad un attaccante che intercetta il traffico, di poterlo vedere in chiaro. L'obiettivo del nostro attacco consiste, perciò, nel modificare la configurazione proxy della macchina della vittima, in modo da costringerla ad utilizzare la nostra macchina come server proxy. Successivamente, mediante l'uso di un certificato autofirmato, cominceremo ad intercettare il traffico HTTPS della comunicazione tra la vittima e il server, in modo da poterlo vedere in chiaro e, di conseguenza, poter rubare le credenziali di accesso ad un sito web, che nel nostro caso sarà il noto social network Facebook.

Al fine di poter eseguire correttamente questo attacco sfrutteremo gli strumenti resi disponibili da *mitmproxy*, un insieme di tool che forniscono un proxy interattivo e che possono essere utilizzati da un utente per ispezionare ed agire sul traffico web di una comunicazione. Tra i vari strumenti che troviamo all'interno di *mitmproxy* ci viene già fornito un certificato autofirmato da poter utilizzare con estrema semplicità; in alternativa, se preferiamo, è possibile creare un certificato da zero utilizzando il tool *Ettercap* e le sue funzioni.

La modifica della configurazione proxy della macchina della vittima è un'operazione necessaria al fine di poter intercettare il traffico. In un attacco reale, per poterlo fare, è possibile fare affidamento su qualche virus o malware adibiti a questo compito. Poiché non si tratta del punto focale di questo attacco, per semplicità, noi agiremo manualmente sulla macchina della vittima impostando la nostra macchina come server proxy da utilizzare; simulando in questo modo che il virus o il malware abbiano già agito. Questo ci permetterà di poterci concentrare sull'intercettazione del traffico e di porre l'attenzione sugli aspetti strettamente collegati al protocollo TLS.

4.1.1 Configurazione di mitmproxy

Generalmente, questo tool non si trova già installato sui sistemi operativi Linux, a parte nella sua versione Kali Linux, dove è possibile trovarlo già presente. Perciò, nel caso sia necessario

installarlo, la prima cosa da fare è andare sul sito ufficiale di *mitmproxy* (link nell'Appendice B.1) dove è possibile scaricare il file compresso *.tar.gz* dell'omonimo tool. Una volta che il download è stato completato, decomprimiamo il file utilizzando il comando:

- `tar -xvzf mitmproxy-7.0.4-linux.tar.gz`

In seguito alla compressione, saranno comparse tre icone, ognuna riferita a un diverso file eseguibile, chiamate *mitdump*, *mitmproxy* e *mitmweb*. Il funzionamento di questi tre programmi e le loro caratteristiche sono state illustrate nel capitolo 3. Al fine di eseguire l'attacco MITM di nostro interesse, per noi sarà sufficiente l'utilizzo di *mitmweb*.

Come già indicato nella sezione precedente, simuleremo che un malware o un virus abbia già intaccato la configurazione proxy della nostra vittima, in modo da permetterci di intercettare il suo traffico. Dunque, sulla macchina della vittima apriamo una nuova istanza del browser che sarà utilizzato per navigare in rete, ci spostiamo nelle impostazioni relative al proxy ed indichiamo come HTTP proxy l'indirizzo IP della nostra macchina attaccante e come numero di porta *8080*. Infine spuntiamo la casella "Usa questo proxy anche per FTP e HTTPS". La Fig. 4.1 fa riferimento al browser Firefox, quello che utilizzerà la nostra vittima; mentre gli altri browser potrebbero avere schermate leggermente differenti.

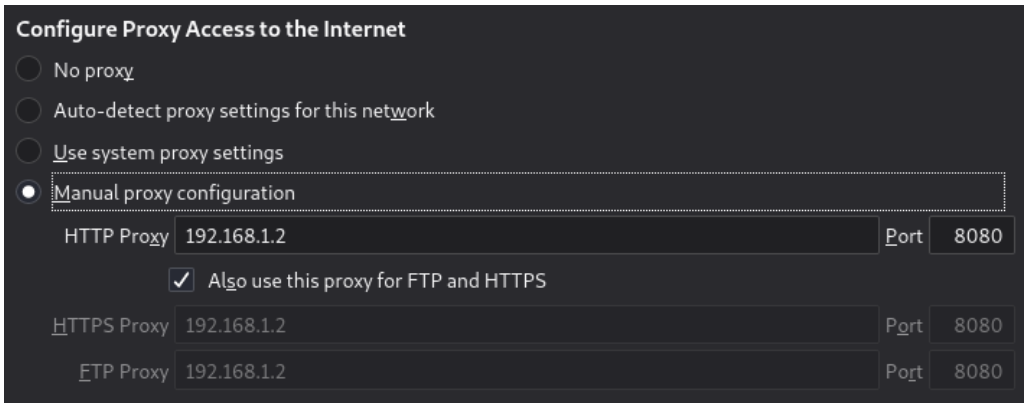

Figura 4.1. Finestra per la configurazione di un server proxy.

Una volta effettuate tutte le modifiche possiamo salvare le nuove impostazioni e proseguire con l'esecuzione dell'attacco.

4.1.2 Esecuzione

Dopo aver effettuato tutte le configurazioni e le modifiche necessarie, la situazione attuale è la seguente: la macchina attaccante Kali Linux ha installata una versione di *mitmproxy*, mentre la macchina della vittima è configurata per passare tutto il traffico HTTP, HTTPS e FTP attraverso un server proxy, che non è altro la nostra macchina attaccante e che quindi potrà osservare tutto il traffico.

Iniziamo mandando immediatamente in esecuzione *mitmweb*, precedentemente scaricato, semplicemente digitando il nome dell'eseguibile sulla riga di comando. Dopo qualche istante, si dovrebbe aprire il browser alla pagina corrispondente al seguente indirizzo:

- `127.0.0.1:8081/#/flows`

La pagina in questione, nella parte superiore, presenta tre schede chiamate rispettivamente *start*, *options* e *flow*, mentre la parte inferiore, inizialmente vuota, è utilizzata alla visualizzazione delle richieste effettuate dalla vittima. Quando la vittima comincia a navigare su Internet, la macchina dell'attaccante intercetterà tutto il traffico che verrà mostrato a schermo. A questo punto, l'attaccante può agire su di esso bloccando, cancellando e duplicando le richieste effettuate agendo

semplicemente sulla finestra di *mitmweb*. Nel nostro caso non è necessario agire direttamente sul traffico, perciò ci limiteremo ad osservarlo e a rubare informazioni contenute in esso.

Quando la vittima decide di effettuare il login al social network Facebook, apre il browser e si collega al sito opportuno. Durante queste azioni, sulla macchina dell'attaccante è possibile osservare che la sezione riservata al traffico HTTP e HTTPS della vittima, inizialmente vuota, ha cominciato a popolarsi. Mentre tenta di collegarsi al sito di Facebook, alla vittima verrà mostrato dal browser un avviso molto simile a quello della Fig. 4.2, in modo da indicargli un possibile problema di sicurezza causato da *mitmproxy* e chiedendogli di scegliere se tornare indietro alla pagina precedente o proseguire sul sito indicato. Molto spesso, le scarse conoscenze informatiche e di sicurezza dell'utente medio sono tali da spingerlo ad accettare il rischio e proseguire nella sua navigazione, non realizzando quanto questo metta in pericolo le sue comunicazioni. Proprio per questo motivo, l'utente medio è il candidato ideale a ricoprire il ruolo della vittima in queste tipologie di attacco.

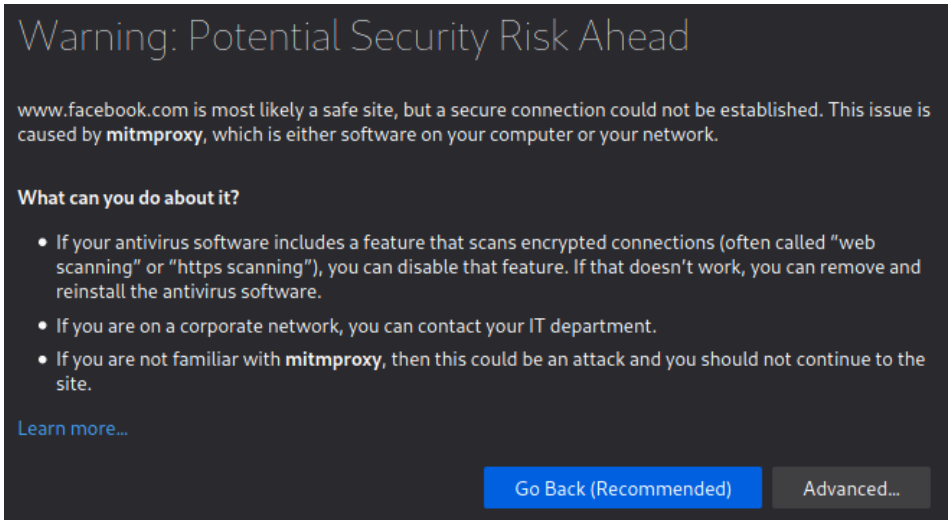

Figura 4.2. Avviso di un potenziale attacco MITM.

Una volta che l'utente prende la decisione di proseguire nella navigazione, il browser registra il sito a cui si sta collegando la vittima come un'eccezione e, in futuro, non mostrerà più il messaggio di allerta. Così facendo, viene permesso all'attaccante di poter continuare ad intercettare traffico ed informazioni anche durante le comunicazioni successive.

Ignara delle conseguenze della sua decisione, la vittima inserisce le proprie credenziali di accesso a Facebook ed effettua il login. All'attaccante è sufficiente analizzare la pagina delle richieste effettuate dalla vittima, filtrando le richieste in base al sito di interesse, in questo caso Facebook, e cercando quelle effettuate mediante il metodo HTTP *POST*. Una volta trovata è necessario selezionarla e, all'interno della finestra laterale che si aprirà, spostarsi nella scheda indicata con il nome *Request*, dove saranno presenti diverse informazioni in chiaro tra cui l'email e la password utilizzate dalla vittima per effettuare il login, proprio come nella Fig. 4.3.

Questa tipologia di attacco è una delle più semplici da mettere in pratica, ma spesso ha bisogno di un aiuto sia esterno sia da parte della vittima. A meno che non sia possibile intervenire manualmente sulla macchina della vittima, prendendone possesso fisicamente e modificando la configurazione proxy, è necessario avvalersi dell'aiuto di un malware o virus che eseguano il compito autonomamente. L'aiuto da parte della vittima non sarà un aiuto intenzionale: è necessario che accetti il rischio di proseguire su una connessione non considerata sicura dal browser in modo che l'attaccante abbia pieno accesso alle informazioni riservate che la vittima inserirà sul web. Il nostro esempio è stato presentato usando come sito a cui effettuare un login un social network e, nonostante abbia avuto successo e siamo riusciti a recuperare le credenziali in chiaro della vittima, i danni che si possono compiere rubando l'account Facebook di una persona sono relativamente limitati, sebbene possano comunque avere conseguenze molto spiacevoli per chi lo subisce. Tuttavia potrebbe capitare che, mentre eseguiamo l'attacco, la vittima decida di

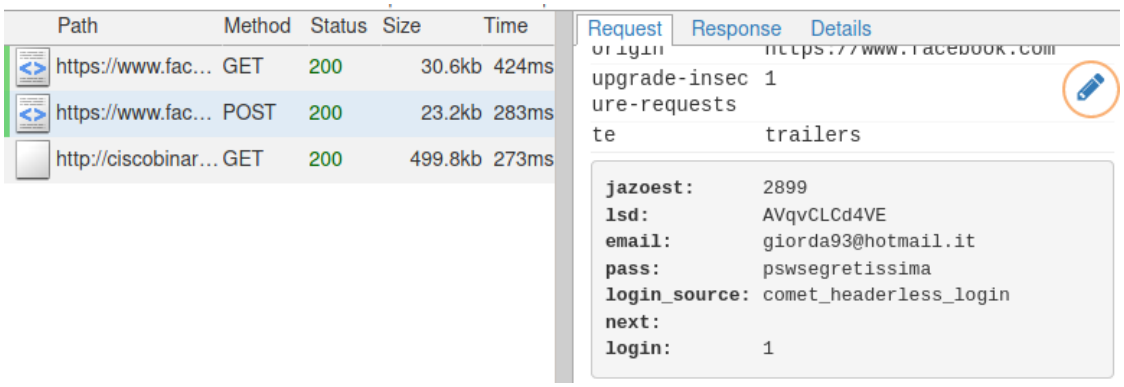

Figura 4.3. Credenziali in chiaro intercettate con *mitmproxy*.

collegarsi a qualche sito di acquisti online oppure al sito della banca, permettendo all'attaccante di ottenere informazioni ancora più sensibili. Inoltre, una consuetudine molto diffusa da parte dell'utente medio è quella di utilizzare, oltre ad un unico indirizzo email, la stessa password per siti differenti, in modo da evitare di dimenticarsela. Questo significa che l'email e la password intercettate eseguendo questa tipologia di attacco, nonostante vengano rubate durante un login ad un semplice social network, è possibile che vengano utilizzate dalla vittima anche per altri siti molto più importanti. Perciò, è possibile utilizzare delle tecniche di "social engineering" per ottenere ulteriori informazioni dalla vittima (es. se ha un conto online e presso che banca) che, unite alle credenziali ottenute precedentemente, possono portare a conseguenze decisamente spiacevoli.

4.2 Attacco Heartbleed

Il secondo attacco che abbiamo provato ad eseguire è l'attacco Heartbleed. Questa tipologia di attacco sfrutta una vulnerabilità presente nelle implementazioni TLS di OpenSSL, dalla versione 1.0.1 alla versione 1.0.1f. Questa vulnerabilità risiede nell'estensione cosiddetta "heartbeat", utilizzata da due host per comunicare l'un l'altro che sono ancora collegati e che la connessione non è stata chiusa o interrotta. In queste versioni vulnerabili di OpenSSL, un mancato controllo tra la lunghezza dichiarata nell'Heartbeat Request e l'effettiva lunghezza del messaggio rende gli host, che utilizzano questa implementazione, vulnerabili a questo attacco.

Per la realizzazione di questo attacco è stato necessario trovare un sistema operativo che utilizzasse una di queste versioni vulnerabile di OpenSSL; la scelta è ricaduta sulla versione 12.04.4 del sistema operativo Ubuntu, la quale utilizza OpenSSL 1.0.1. In seguito OpenSSL è stato aggiornato alla versione 1.0.1f ed eseguendo nuovamente l'attacco sono stati ottenuti, come prevedibile, i medesimi risultati.

Al fine di verificare la vulnerabilità del server all'attacco è possibile usare sia *TLS-Attacker*, un framework per l'analisi delle librerie TLS, sia *Nmap*, un software open source per effettuare port scanning volto all'individuazione dei servizi di rete attivi. Nel nostro caso utilizzeremo il secondo software poichè già presente sulla macchina attaccante, senza dover installare un tool esterno per la sola verifica della vulnerabilità.

Infine, per l'esecuzione dell'attacco è stato usato il framework *Metasploit*, contenente molti strumenti per sfruttare altrettante vulnerabilità. Uno di questi strumenti in particolare è utilizzato per inviare Heartbeat Request con impostazioni personalizzabili, in modo da poter eseguire l'attacco e ricavare informazioni riservate dal server.

4.2.1 Configurazione di Ubuntu 12.04.4

La configurazione della macchina dell'host vittima, al fine di poter eseguire l'attacco Heartbleed correttamente, prevede l'utilizzo della versione 12.04.4 del sistema operativo Ubuntu. Questa

versione del sistema operativo in particolare è una delle prime ad essere stata rilasciate avente già installata al suo interno una versione di OpenSSL vulnerabile ad Heartbleed, ovvero la versione 1.0.1.

Al fine di poter far partire l'esecuzione di un server con OpenSSL sono necessarie sia le chiavi e sia un certificato; all'interno del tool *TLS-Attacker* ci viene fornito uno script per la generazione di diverse coppie chiave-certificato: RSA, DSA e ECC. Come abbiamo descritto nella sezione dedicata ad Heartbleed, la vulnerabilità da sfruttare per questo attacco risiede nell'implementazione della libreria OpenSSL e non nel protocollo TLS vero e proprio, perciò la scelta delle chiavi risulta ininfluente ai fini del risultato, tutte risulterebbero vulnerabili. Perciò, sebbene l'attacco sia stato testo utilizzando tutte le possibili combinazioni di chiavi, successivamente verrà presentato l'attacco contro un server che utilizza una chiave RSA da 1024 bit e il relativo certificato.

```
ubuntu@ubuntu:~/TLS-Attacker/resources$ openssl s_server -key rsa1024key.pem -ce
rt rsa1024cert.pem
Using default temp DH parameters
ACCEPT
```

Figura 4.4. Avvio di un server con chiave RSA a 1024 bit e relativo certificato.

Per poter utilizzare lo script per la creazione delle chiavi e dei certificati non è necessario procedere con l'installazione completa di *TLS-Attacker*, ma è possibile scaricare unicamente lo script ed eseguirlo come spiegato nell'Appendice A.2.1.

4.2.2 Configurazione di Metasploit

Sulla macchina virtuale dell'host attaccante è stata utilizzata la versione 2021.1 del sistema operativo Kali Linux, rilasciata a febbraio 2021. Si tratta di una distribuzione Linux pensata per la sicurezza informatica e l'ethical hacking ("hacking etico": hacking volto a rilevare i punti deboli di un sistema e risolverli). Sebbene su questa particolare versione di Kali Linux il tool *Metasploit* sia già presente, non è detto che sia lo stesso per altre versioni e perciò potrebbere rivelarsi necessario installarlo manualmente, in modo da poter eseguire l'attacco. Esistono due versioni installabili di *Metasploit*: *Metasploit* Framework e *Metasploit* Pro; la prima gratuita e la seconda a pagamento dopo un primo periodo di prova gratuito. Gli strumenti che ci sono utili per l'esecuzione dell'attacco sono implementati anche nella versione gratuita, di conseguenza abbiamo deciso di affidarci a *Metasploit* Framework.

Nel caso fosse necessario procedere all'installazione è possibile seguire la guida presente nell'Appendice A.3.

Una volta conclusa la procedura di installazione, è possibile avviare il framework da riga di comando, digitando ciò che segue:

- `msfconsole`

e successivamente sarà possibile usufruire di tutte le funzionalità messa a disposizione da *Metasploit* Framework.

4.2.3 Esecuzione

La prima cosa che è necessario fare, consiste nell'avviare il server con OpenSSL utilizzando le chiavi e i certificati che abbiamo creato. Perciò apriamo una finestra del terminale all'interno della cartella *TLS-Attacker/resources* e lanciamo il comando:

- `openssl s_server -key rsa1024key.pem -cert rsa1024cert.pem`

Dove *rsa1024key.pem* e *rsa1024cert.pem* sono rispettivamente la chiave e il certificato che abbiamo scelto.

Una volta avviato il server sulla macchina vittima, la prima cosa da fare è spostarci sulla macchina attaccante in modo da eseguire un test per la vulnerabilità all'attacco Heartbleed contro il server stesso. Come spiegato precedentemente, è possibile effettuare questa verifica sia con *TLS-Attacker* sia con *Nmap*; ma poiché l'utilizzo del tool *TLS-Attacker* avrebbe richiesto l'installazione di un secondo programma sulla macchina attaccante, è stato più pratico utilizzare *Nmap*, già installato sul sistema operativo presente.

É dunque sufficiente eseguire il seguente codice da riga di comando, come in Fig. 4.5:

- `nmap -p [nport] -script ssl-heartbleed [IPaddress]`

Dove:

nport: è il numero di porta su cui comunicare con il server, 4433 nel nostro caso;

ssl-heartbleed: è lo script per la verifica della vulnerabilità;

IPaddress: è l'indirizzo IP del server, 192.168.1.5 nel nostro caso.

```
┌──(kali㉿kali)-[~]
└─$ nmap -p 4433 -script ssl-heartbleed 192.168.1.5
Starting Nmap 7.91 ( https://nmap.org ) at 2021-10-28 14:09 UTC
Nmap scan report for ubuntu.station (192.168.1.5)
Host is up (0.00044s latency).

PORT     STATE SERVICE
4433/tcp open  vop
| ssl-heartbleed:
|   VULNERABLE:
|   The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryp
tographic software library. It allows for stealing information intended to be
 protected by SSL/TLS encryption.
|     State: VULNERABLE
|     Risk factor: High
|       OpenSSL versions 1.0.1 and 1.0.2-beta releases (including 1.0.1f and
1.0.2-beta1) of OpenSSL are affected by the Heartbleed bug. The bug allows fo
r reading memory of systems protected by the vulnerable OpenSSL versions and
could allow for disclosure of otherwise encrypted confidential information as
 well as the encryption keys themselves.
|
|     References:
|       http://www.openssl.org/news/secadv_20140407.txt
|       http://cvedetails.com/cve/2014-0160/
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
```

Figura 4.5. Risultato della verifica della vulnerabilità.

La risposta all'esecuzione di questo script è un messaggio che ci fornisce informazioni riguardo il sistema operativo su cui gira il server, lo stato della porta su il server è stato contattato, lo stato della vulnerabilità all'attacco Heartbleed ("VULNERABLE/NOT VULNERABLE") e il fattore di rischio; oltre ad un breve messaggio in cui vengono indicate le possibili conseguenze dell'essere vulnerabili all'attacco. Una volta che è stata verificata l'effettiva vulnerabilità possiamo mettere in atto l'attacco.

Possiamo dunque aprire la console di *Metasploit* Framework con il comando:

- `msfconsole`

ed eseguire la ricerca dei moduli riguardanti Heartbleed:

- `search heartbleed`

Il modulo di cui abbiamo bisogno è quello indicato con il nome *openssl_heartbleed*, perciò lo carichiamo:

- `use auxiliary/scanner/ssl/openssl_heartbleed`

Una volta terminato il caricamento del modulo è necessario configurare alcune opzioni riguardanti l'attacco. Lo stato corrente di queste opzioni è visionabile utilizzando il comando *info*. Di queste, le principali sono:

DUMPFILTER: pattern per filtrare le informazioni rubate prima di memorizzarle;

LEAK_COUNT: numero di volte da rubare le informazioni per ogni SCAN o DUMP (default: 1);

RESPONSE_TIMEOUT: numero di secondi da attendere per ricevere una risposta dal server (default: 10);

RHOSTS: indirizzo IP dell'host vittima;

RPORT: numero di porta su cui contattare il server (default: 443);

THREADS: numero di thread concorrenti (al massimo uno per ogni host. Default: 1);

TLS_CALLBACK: protocollo da usare, "None" per usare i socket raw TLS (Accettati: SMTP, IMAP, JABBER, POP3, FTP, POSTGRES);

TLS_VERSION: versione di SSL/TLS da usare (Accettati: SSLv3, 1.0, 1.1, 1.2. Default: 1.0).

Inoltre, nonostante non venga mostrata a video, esiste un'ulteriore opzione molto importante:

HEARTBEAT_LENGTH: indica la lunghezza da indicare per l'Heartbeat Request (default: 65535).

Nel nostro caso, le uniche opzioni da modificare sono *RHOSTS* e *RPORT*, rispettivamente con l'indirizzo IP e la porta del server. Utilizziamo le impostazioni di default sia per la versione di TLS da utilizzare sia per la lunghezza della Heartbeat Request, quest'ultima già imposta al limite massimo. Perciò eseguiamo le modifiche descritte digitando:

- `set RHOSTS 192.168.1.5`
- `set RPORT 4433`
- `set VERBOSE true`

Quest'ultimo comando ci tornerà utile per osservare a video alcune informazioni in più riguardo a cosa sta avvenendo durante l'attacco.

Una volta terminata la parte di configurazione, non ci resta che mandare in esecuzione l'attacco con il seguente comando:

- `exploit`

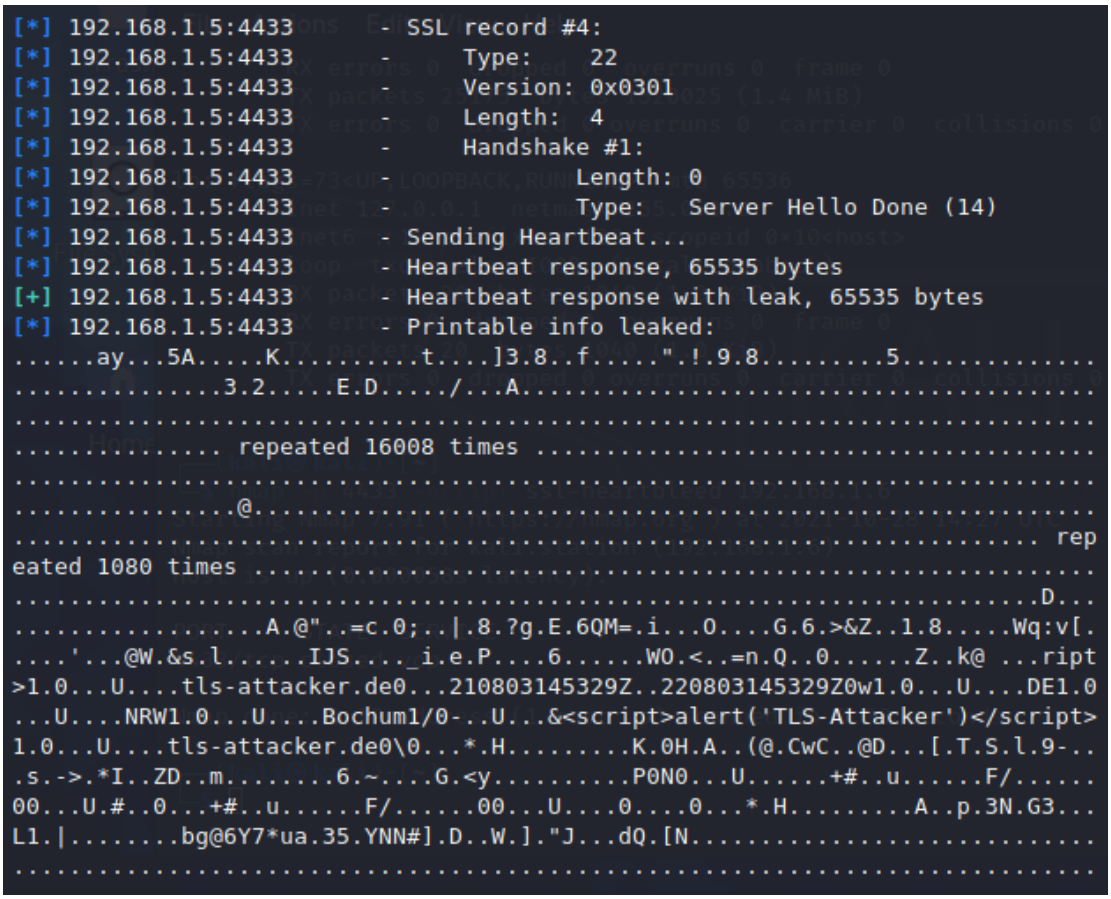

Figura 4.6. Risultato dell'esecuzione dell'attacco Heartbleed.

Per prima cosa inizia la fase di handshake, mediante l'invio del messaggio di *Client Hello* a cui il server risponderà con il messaggio di *Server Hello*. Come è possibile osservare nella Fig. 4.6, una volta concluso il protocollo di handshake, verrà inviata l'Heartbeat Request in cui sarà indicata una lunghezza pari a 65535 byte e verrà ricevuta in risposta un Heartbeat Response lunga effettivamente 65535 byte e perciò piena di informazioni provenienti dalla memoria del server. A questo punto, il framework *Metasploit* ci stampa a video tutte le informazioni casuali ricevute in risposta dal server attaccato, permettendoci così di analizzarle. Durante l'esecuzione dell'attacco non si ha il controllo riguardo le informazioni che vengono rubate e poiché lo scopo principale è quello di ottenere qualche informazione utile, è necessario ripetere questa procedura più e più volte. In questo modo, per ogni tentativo di esecuzione dell'attacco, si otterranno parti di memoria differenti, aumentando sensibilmente la probabilità di ricavarne informazioni utili.

Le statistiche più recenti riguardo a questa vulnerabilità risalgono a luglio del 2019, ovvero 5 anni dopo la pubblicazione delle modalità dell'attacco Heartbleed, e, secondo uno studio condotto da Shodan, 91,063 dispositivi risultavano ancora affetti da questa vulnerabilità. 21,258 di questi dispositivi, circa il 23%, erano situati negli Stati Uniti d'America; mentre solo il 3% (2,858) risultava situato sul suolo italiano.

4.3 Attacco Padding Oracle

Nella categoria degli attacchi alle ciphesuites, uno molto conosciuto è l'attacco Padding Oracle, il quale permette all'attaccante di decifrare parti di ciphertext, a patto che la comunicazione utilizzi una suite in modalità Cipher Block Chaining (CBC). Da SSLv3, una versione di SSL ormai deprecata, a TLS 1.2, l'uso di risposte differenti da parte del server in caso di errori permetteva

ad un attaccante di poter sfruttare il server stesso come un "oracolo" e di avere la possibilità di decifrare i byte di questo blocchi, il tutto senza la necessità di essere in possesso della chiave di decifrazione (per una descrizione più accurata sul funzionamento dell'attacco si faccia riferimento alla Sezione 1.6.1).

Prima di poter verificare se un server è vulnerabile a questa tipologia di attacco, è senz'altro d'aiuto ricercare quelle connessioni che rispettano le due caratteristiche principali: la possibilità di usare TLS 1.0, TLS 1.1 e TLS 1.2 e il supporto per le ciphersuites di tipo CBC. Per fare ciò, un possibile metodo è quello di utilizzare un IDS e le informazioni che ci fornisce sulle connessioni mediante file di log. Abbiamo deciso di utilizzare il software *Zeek*, il quale ci fornisce informazioni riguardo sia la versione di TLS utilizzata nella comunicazione sia, a differenza di *Suricata*, informazioni riguardanti alle ciphersuites supportate dal server e in uso nella comunicazione.

Una volta che le informazioni riguardo alle varie connessioni sono state memorizzate all'interno dei file log, è necessario filtrarle in modo che ci vengano mostrate solo quelle di nostro interesse. Questo ci ha spinto alla creazione di un semplice script in linguaggio awk, il cui funzionamento verrà approfondito in una sezione successiva, ma che in breve ci permetterà di estrarre dai file di log solamente le connessioni per cui ha un senso valutarne la vulnerabilità.

Al termine dell'esecuzione dello script in questione verrà prodotto un file di output, e grazie al tool *TLS-Attacker* sarà possibile testare se i server delle connessioni indicate nel file sono vulnerabili o meno all'attacco Padding Oracle. Il comando per la verifica della vulnerabilità potrà restituirci tre risultati distinti: `true / false / uncertain`; nel caso in cui il risultato ottenuto fosse quello di incertezza, è comunque possibile provare ad eseguire l'attacco.

Infine, quando *TLS-Attacker* ci avrà confermato l'effettiva vulnerabilità del server all'attacco, è possibile passare all'esecuzione vera e propria. A questo fine, ci siamo procurati di un tool ed uno script, entrambi messi a disposizione dai proprietari su GitHub: il tool si chiama *padding-oracle-attacker* e lo script *PadBuster*. Entrambi gli strumenti vanno eseguiti da riga di comando indicando pressoché le stesse opzioni, tuttavia il primo strumento ci da la possibilità di osservare meglio l'evolversi dell'attacco e cosa sta succedendo durante l'esecuzione, per questo motivo il tool che abbiamo deciso di utilizzare è *padding-oracle-attacker*.

4.3.1 Configurazione di Zeek

Dopo che è stata effettuata l'installazione come indicato nella Sezione 2.4.2, è possibile iniziare ad utilizzare l'IDS *Zeek* gestendo i comandi con ZeekControl, una shell interattiva messa a disposizione degli utenti. Per avviare ZeekControl è necessario spostarsi nella cartella */usr/local/zeek/bin* e, utilizzando i privilegi da amministratore, digitare da riga di comando:

- `./zeekctl`

La prima volta che viene usata la shell è consigliabile eseguire una breve configurazione automatica utilizzando i comandi:

- `install`
- `start`

O, in alternativa, mediante l'utilizzo di un unico comando che combina le due azioni precedenti e che andrà eseguito in seguito a qualsiasi cambiamento nelle politiche di *Zeek* o nella configurazione di ZeekControl:

- `deploy`

Infine, per interrompere l'esecuzione di *Zeek* e cominciare la compressione e il salvataggio dei file di log, è sufficiente digitare:

- `stop`

Nel momento in cui *Zeek* viene installato su un sistema, vengono create diverse cartelle sotto il percorso di installazione di default */usr/local/zeek/*, che noi chiameremo *$PREFIX*. Di tutte queste cartelle che sono state create, è molto utile conoscere il ruolo che alcune di esse ricoprono e quali informazioni possiamo trovare al loro interno. Nel percorso *$PREFIX/etc/* è possibile trovare tre diversi file di configurazione, che è necessario modificare:

networks.cfg :
nel quale possiamo definire la nostra rete locale al fine di una corretta analisi del traffico;

node.cfg :
nel quale possiamo definire un singolo nodo o diversi tipi di nodi e le loro impostazioni corrispondenti;

zeekctl.cfg :
s tratta del file di configurazione di ZeekControl, al cui interno sono presenti le impostazioni per le cartelle dei file di log, l'intervallo di rotazione di questi file e la configurazione email.

Se ci spostiamo nella directory *$PREFIX/logs/* è possibile vedere che è presente una cartella chiamata *current*. Questa cartella, indicata nel file *zeekctl.cfg*, contiene tutti i file di log che vengono prodotti da *Zeek* durante l'esecuzione del programma. Al momento dell'interruzione di *Zeek*, sarà creata una nuova cartella, il cui nome corrisponde alla data di quel giorno, dove tutti i file di log presenti nella cartella *current* verranno compressi e salvati al suo interno, permettendo così di poterli decomprimere ed analizzare in seguito.

Dalla directory *$PREFIX/share/* è possibile raggiungere altre tre cartelle differenti, che riguardano tutte gli script utilizzati dal programma per l'analisi del traffico. Le cartelle in questione sono:

zeek/base/: contiene gli script base che saranno sempre caricati;

zeek/policy/: contiene alcuni script più situazionali, i quali devono essere caricati esplicitamente dall'utente mediante i file di configurazione;

zeek/site/: contiene il file *local.zeek*, il quale può essere usato per caricare script dalla cartella *zeek/policy/* o altri script personalizzati.

Gli script contenuti all'interno della prima e della seconda cartella non andrebbero mai modificati, poiché eventuali modifiche effettuate dall'utente andrebbero perse con i successive aggiornamenti alle nuove versioni di *Zeek*. Nella terza cartella invece è possibile anche inserire gli script personalizzati, senza il timore che vadano persi nel momento in cui si effettua un aggiornamento del software.

Perciò, una volta effettuata la configurazione degli script è sufficiente utilizzare i comandi visti precedentemente per salvarla e cominciare ad analizzare il traffico sulla rete prescelta.

4.3.2 Configurazione script

Per poter effettuare la ricerca delle connessioni tra client e server che potrebbero risultare vulnerabili all'attacco Padding Oracle, abbiamo la necessità di eseguire un'analisi dei file di log che ci vengono forniti dall'IDS *Zeek*. All'interno di questi file dobbiamo cercare quelle connessioni che rispecchiano le seguenti caratteristiche:

- Utilizzano una delle seguenti versioni di TLS: 1.0, 1.1 e 1.2;
- Utilizzano una ciphersuite in modalità Cipher Block Chaining.

Durante l'esecuzione di *Zeek*, vengono appunto prodotti diversi file di log, quello che ci interessa maggiormente si chiama *ssl.log* e ci fornisce entrambe le informazioni che stiamo cercando. Questo file è il risultato dell'analisi del traffico SSL/TLS e, al suo interno, vengono registrate tutta una serie di informazioni, più o meno utili. Noi dovremo analizzare successivamente questo file, selezionando le informazioni di nostro interesse:

- Indirizzo IP e porta del client;
- Indirizzo IP e porta del server;
- Versione di SSL/TLS scelta dal server;
- Ciphersuite SSL/TLS scelta dal server.

Come detto prima, la verifica manuale di ogni connessione all'interno del file comporterebbe un impiego di tempo considerevole. Abbiamo quindi deciso di sviluppare uno script in linguaggio Awk per poter analizzare ed estrarre in poco tempo le informazioni rilevanti. Inoltre, il file che andremo ad analizzare memorizza tutte le informazioni suddividendole per colonne e attribuendo ad ogni colonna un nome identificativo, il che ha semplificato lo sviluppo dello script in questione.

É possibile vedere il codice dello script nella Fig. 4.7. Per crearne un copia è sufficiente copiare il codice dello script in un nuovo file di testo e salvarlo.

```
#!/usr/bin/awk -f
BEGIN {

        FS = "\t"
}

NR > 8 {

        if(($7 == "TLSv12" || $7 == "TLSv11" || $7 == "TLSv10") && $8 ~ /CBC/){
                print($5, "\t", $6, "\t", $7, "\t", $8 ) > "/home/kali/Desktop/poa_vuln.txt";
        }
}
```

Figura 4.7. Codice dello script awk *analysis.awk*.

Lo script Awk può essere eseguito tranquillamente da riga di comando, inserendo come unico argomento il percorso al file di log *ssl.log*:

- `./nome_script $PATH/ssl.log`

Dove `nome_script` deve essere sostituito con il nome che avete dato voi allo script.

Una volta avviata l'esecuzione, lo script aprirà il file di log indicato come argomento, in questo caso `$PATH/ssl.log`, ed eseguirà un'analisi su tutti i dati presenti nella colonna del campo *version* e del campo *cipher*. Sul primo campo controlla se il dato indicato è uguale a uno dei seguenti valori: *TLSv12*, *TLSv11* o *TLSv10*. Mentre sul secondo campo controlla se all'interno della stringa indicante la ciphersuite è presente la sigla *CBC*.

Se entrambe le condizioni qui sopra vengono rispettate, allora lo script stampa sul file predefinito che abbiamo chiamato `poa_vuln.txt` (indicato all'interno dello script e perciò modificabile) le informazioni che ci saranno utili per testare l'effettiva vulnerabilità del server, in un formato ordinato e tabulato. Le informazioni che vengono memorizzate sono:

- l'indirizzo IP del server;
- la porta del server;
- la versione TLS della connessione;
- la ciphersuite utilizzata nella connessione.

A questo punto non ci resta che verificare se queste connessioni sono vulnerabili affidandoci al tool *TLS-Attacker*, il quale ci segnalerà i server contro i quali è possibile eseguire l'attacco Padding Oracle.

4.3.3 Configurazione di TLS-Attacker

Infine, per avere modo di verificare l'effettiva vulnerabilità del server ad uno specifico attacco mi sono affidato a *TLS-Attacker*, un tool che ci permette di eseguire tale verifica mediante dei semplici input da riga di comando. Per poter compilare e funzionare correttamente, è richiesto che sulla macchina siano installati alcuni prerequisti. É possibile seguire l'Appendice A.2 per verificarne la presenza e procedere all'installazione di *TLS-Attacker*.

Terminata l'installazione di tutti i componenti precedenti, tutti i file *.jar* si troveranno all'interno della directory *apps*, da cui è possibile eseguire diversi comandi: verifica delle vulnerabilità, esecuzione di un attacco, collegamento ad un server, esecuzione di un server e altro ancora.

Nella directory *resources* è presente uno script per la creazione di chiavi e certificati di diverso tipo. Una volta che questo script è stato eseguito, abbiamo la possibilità di far partire l'esecuzione di un server utilizzando OpenSSL e indicando come parametri per i file *.pem* delle chiavi e dei certificati quelli appena creati con lo script in questione, come indicato nell'Appendice A.2.1.

Come detto, una volta all'interno della directory *apps* è possibile lanciare il comando per verificare se un server risulta essere vulnerabile ad un certo tipo di attacco, è sufficiente personalizzare il seguente comando:

- `java -jar Attacks.jar [Attack] -connect [host:port]`

Dove:

Attack : è il nome dell'attacco per cui verificare la vulnerabilità. I possibili attacchi sono:

- bleichenbacher
- cve20162107
- early_ccs
- early_finished
- generalDrown
- heartbleed
- invalid_curve
- lucky13
- padding_oracle
- poodle
- pskbruteforcerclient
- pskbruteforcerserver
- simple_mitm_proxy
- specialDrown
- tls_poodle

host: l'indirizzo IP della vittima;

port: la porta del server.

Se invece abbiamo la necessità di eseguire l'attacco vero e proprio, è sufficiente aggiungere l'opzione *-executeAttack* al comando per la verifica della vulnerabilità. Questa opzione è presente per quasi tutti gli attacchi elencati precedentemente, tuttavia per molti di loro non è stato implementato il codice di esecuzione dell'attacco e di conseguenza l'opzione risulta inutile. Ad oggi, il codice è stato implementato solamente per gli attacchi Bleichenbacher e Invalid Curve. Nel nostro caso utilizzeremo un altro tool per l'esecuzione dell'attacco, mentre *TLS-Attacker* lo sfrutteremo solo per la verifica delle vulnerabilità.

4.3.4 Esecuzione

Come già spiegato, per poter effettuare un attacco di tipo Padding Oracle, è necessario che la connessione con il server abbia determinate caratteristiche: la prima prevede che all'interno della connessione venga utilizzata una versione di TLS non superiore a 1.2, mentre la seconda impone che venga utilizzata una ciphersuite in modalità Cipher Block Chaining.

É perciò necessario utilizzare gli strumenti a nostra disposizione per individuare quelle connessioni che rispettino queste caratteristiche. Dopo aver configurato *Zeek* come illustrato in una sezione precedente, lo avviamo in modo che cominci ad intercettare il traffico su tutta la nostra rete locale ed produca i file di log che dovremo analizzare successivamente. É necessario attendere che l'IDS abbiamo intercettato più connessioni possibili, in modo da avere più probabilità di trovare un server che possa risultare vulnerabile.

La nostra macchina vittima si appoggia sul sistema operativo Ubuntu 15.10 e con la versione 1.0.2d di OpenSSL, che ancora supporta l'utilizzo di ciphersuite in modalità CBC. La nostra macchina attaccante basata su Kali Linux invece possiede la versione 1.1.1i di OpenSSL che, non supportando più le ciphersuite in modalità CBC in comune con il server, permette di collegarsi ad esso solamente utilizzando ciphersuite differenti e perciò non vulnerabili. Tuttavia, il tool *TLS-Attacker* ci fornisce la possibilità di collegarci a un server senza dover utilizzare OpenSSL ed utilizzando alcune ciphersuite in modalità CBC.

Dunque, ci spostiamo nella cartella *apps* di *TLS-Attacker* e avviamo la connessione con il server eseguendo il seguente comando:

- `java -jar TLS-Client.jar -connect [host]:[port]`

Questo comando ci permetterà di collegarci al server utilizzando la ciphersuite *DES_CBC3_SHA*.

```
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMDBAIACgQgDJ6LwEgVDaJjeJI8aia67xYkvhq0ARhkVUZlPxzn8H8E
MHTQB/og4DY2qCvxXAYItrqeKdfzlXq89xj9gkb0vU1kwCa7lCMzffIttZV/0pZt
Q6EGAgRhenF9ogQCAgEspAYEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
CIPHER is DES-CBC3-SHA
Secure Renegotiation IS supported
```

Figura 4.8. Connessione al server mediante ciphersuite vulnerabile all'attacco.

A questo punto, quando il file *ssl.log* presente nella cartella dove *Zeek* salva i file di log sarà pronto, è sufficiente avviare lo script *analysis.awk* che si trova sul desktop, inserendo come primo e unico argomento il percorso al file di log *ssl.log*:

- `./analysis.awk $PATH/ssl.log`

Una volta che l'esecuzione dello script sarà terminata, sul desktop comparirà un file di testo chiamato *poa_vuln.txt*, il quale conterrà al suo interno le informazioni di tutte le connessioni che erano presenti nel file *ssl.log* che rispecchiano le caratteristiche per essere vulnerabili all'attacco Padding Oracle.

Ora possiamo tornare all'interno della cartella *apps* del tool *TLS-Attacker*, da dove eseguiremo il seguente comando:

- `java -jar Attacks.jar padding_oracle -connect [host]:[port]`

sostituendo *host* e *port* rispettivamente con l'indirizzo IP e la porta del server che vogliamo testare. Ripetiamo questa operazione per tutte le coppie *indirizzo IP - porta* presenti nel file di output dello script, per ognuna ci verrà indicato se il server risulta essere vulnerabile o meno all'attacco Padding Oracle.

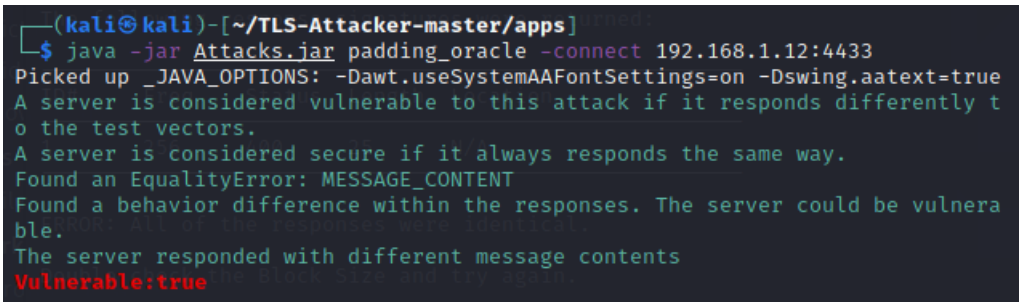

Figura 4.9. Risultato della verifica della vulnerabilità all'attacco Padding Oracle.

Come previsto, nel nostro caso il server risulta essere vulnerabile ed è perciò possibile procedere procedere all'esecuzione dell'attacco vero e proprio. Per fare ciò, utilizzeremo il tool *padding-oracle-attacker*, per il quale sono descritte le procedure di installazione nell'Appendice A.4.

L'attacco Padding Oracle ci da la possibilità di decifrare uno o più blocchi di dati, effettuando ripetute richieste al server ed analizzandone le risposte. I blocchi di dati che vogliamo decifrare è possibile analizzarli e recuperarli con uno sniffer di pacchetti, per cui noi utilizzeremo *Wireshark*. Avviando il software ed analizzando la comunicazione d'interesse con il server, è possibile osservare tutti i pacchetti in transito, a quali messaggi corrispondono e la sequenza di byte che li compone. É inoltre possibile copiare ed incollare queste sequenze di byte, ed è ciò che faremo noi.

Purtroppo non è possibile utilizzare il tool *padding-oracle-attacker* contro un server che possiede un certificato autofirmato, proprio come il nostro. Al momento dell'esecuzione dell'attacco viene lanciata un'eccezione che segnala la presenza del certificato autofirmato e l'esecuzione si interrompe. Per ovviare a questo problema e poter mostrare il funzionamento del tool, ho deciso di testare l'attacco contro un sito web messo a disposizione da Google chiamato *Google Gruyere*. Si tratta di un sito web appositamente per effettuare test di attacchi informatici, infatti sono presenti un buon numero di bug e vulnerabilità che, completando le sfide che il sito propone come se fosse un gioco, è possibile scoprire, conoscere ed imparare ad evitare.

Perciò, a questo punto apriamo il terminale in una directory qualunque e digitiamo il seguente comando:

- `poattack decrypt https://google-gruyere.appspot.com/decrypt?ciphertext=hex:[ciphertext_hex] 16 400`

Dove:

poattack: un alias per poter lanciare il tool da riga di comando;

decrypt: l'azione che vogliamo eseguire [*decrypt, encrypt, analyze*];

https://google-gruyere.appspot.com/decrypt?ciphertext= : l'url del server contro cui vogliamo eseguire l'attacco;

ciphertext_hex: il testo cifrato in esadecimale da dover decifrare, nonché multiplo della grandezza del blocco;

16: la grandezza del blocco;

400: il codice di errore che viene restituito dal server in caso di decifrazione fallita e che si può recuperare mediante l'analisi del server.

Mandiamo in esecuzione il comando e attendiamo che ci venga fornito il risultato. Il tool *padding-oracle-attacker* è stato scelto anche perché è possibile osservare sull'interfaccia cosa avviene durante l'esecuzione dell'attacco. Una volta terminata l'esecuzione, ci troveremo di fronte ad una schermata come quella della Fig. 4.10, da cui è possibile ricavare alcune informazioni su come si è svolto l'attacco:

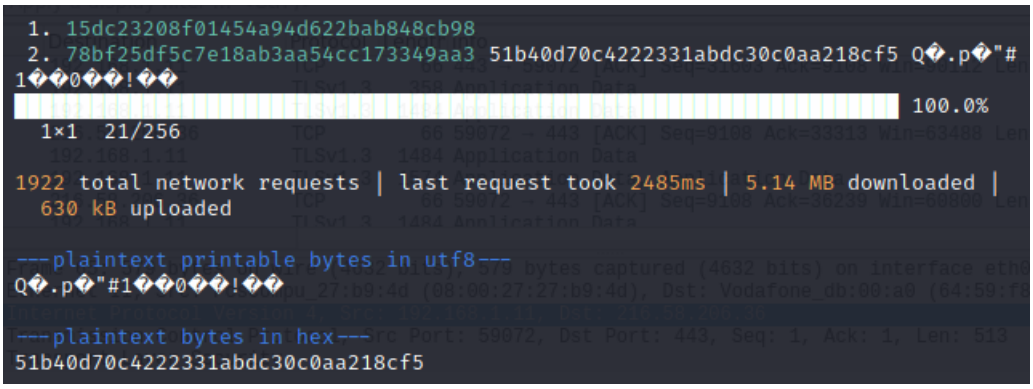

Figura 4.10. Risultato dell'esecuzione dell'attacco Padding Oracle.

- Il numero di richieste effettuate al server;
- La quantità di dati caricati e scaricati;
- Il testo in chiaro in due codifiche: UTF-8 e esadecimale.

Nel nostro caso, il plaintext risultante dall'esecuzione del mio attacco è una sequenza di caratteri casuali senza alcun significato. Tuttavia, se conduciamo l'attacco decifrando i blocchi opportuni, è possibile ottenere informazioni molto importanti che possono mettere a repentaglio la sicurezza del server e delle sue comunicazioni.

Capitolo 5

Conclusioni

In questo lavoro di tesi è stato effettuato uno studio analitico del protocollo TLS, delle sue funzionalità e dei possibili attacchi che possono essere messi in atto contro il protocollo, mediante lo sfruttamento di diverse vulnerabilità: quelle ancora presenti nelle versioni passate e quelle introdotte con gli aggiornamenti più recenti.

L'obiettivo principale era quello di effettuare un'analisi del traffico TLS che veniva scambiato dalla vittima con un altro nodo, utilizzando degli strumenti adatti al compito, come gli IDS *Zeek* e *Suricata*. L'analisi era volta a ricercare quelle connessioni che presentassero delle vulnerabilità conosciute, rendendole così delle potenziali vittime di un attacco. La fase successiva si è concentrata sulla ricerca di possibili strumenti per verificare l'effettiva presenza di qualche vulnerabilità e per lanciare qualche attacco. In rete esistono tantissimi tool o script per svolgere il compito di verifica delle vulnerabilità, noi ci siamo concentrati in particolare su *TLS-Attacker*, ancora in via di sviluppo ma con molte funzionalità e semplice da utilizzare. Meno diffusi, ma comunque presenti, sono gli strumenti veramente efficaci per lanciare un attacco, siano essi commerciali (come *Metasploit*) o sviluppati da privati e messi open source online (come *padding-oracle-attacker*).

Una volta trovati gli strumenti adatti, siamo riusciti a mettere in atto tre attacchi differenti a TLS. Il primo ad essere stato sviluppato è l'attacco Man-In-The-Middle che, sebbene sia un pò più semplice rispetto agli altri, ha permesso di intercettare il traffico HTTPS grazie al tool *mitmproxy*. Il secondo è stato l'attacco Heartbleed: una volta trovata una macchina vulnerabile a questo attacco mediante *TLS-Attacker* o *Nmap* abbiamo potuto recuperare i dati dalla memoria del server sfruttando *Metasploit*. Ed infine, un attacco Padding Oracle con l'ausilio dell'IDS *Zeek*, del tool *padding-oracle-attacker* e di uno script awk scritto da me medesimo. Nonostante questo attacco abbia riscontrato delle difficoltà nella sua esecuzione, una su tutte il fatto che il tool per l'attacco non riesca a lanciare l'attacco contro un server che utilizza un certificato autofirmato, è stato dimostrato come sia possibile da mettere in atto. Infine, sono state create anche delle guide per l'installazione dei vari strumenti in modo che, seguendo passo dopo passo le istruzioni, sia possibile creare il proprio ambiente di lavoro e replicare gli attacchi.

In conclusione, è stato mostrato come, sebbene le conoscenze e le risorse fossero limitate, effettuando un lavoro di ricerca e applicazione è possibile recuperare degli strumenti per lanciare degli attacchi informatici. Se gli strumenti offensivi sono stati creati proprio per questo compito, gli strumenti difensivi possono essere comunque utili e rivelarci informazioni che saranno necessarie per individuare le vittime degli attacchi.

Il protocollo TLS è costantemente in evoluzione, l'ultimo aggiornamento risale solo a qualche anno fa ed ha migliorato decisamente la sicurezza e l'affidabilità delle comunicazioni. Tuttavia, ogni aggiornamento porta con sé nuovi possibili punti di attacco e nuovi possibili attacchi. Il mondo della sicurezza informatica e quello degli attacchi informatici continuano a rincorrersi l'un l'altro: ogni nuova evoluzione dei protocolli e dei sistemi informatici può portare una nuova minaccia da parte dei malintenzionati e ogni nuovo attacco informatico che viene scoperto è una nuova sfida da risolvere.

Appendice A

Manuali di installazione

A.1 Installazione di Suricata

L'installazione dell'IDPS *Suricata* è possibile effettuarla in due modi differenti: dalla sorgente e attraverso i binary packages. Verranno illustrati entrambe le modalità così che possiate scegliere liberamente quale preferite.

Sorgente

Innanzitutto *Suricata* richiede delle dipendenze che vanno installate, perciò inizieremo da quelle. Le dipendenze comprendono il seguente insieme di librerie e tool:

- `libjansson`
- `libpcap`
- `libpcre2`
- `libmagic`
- `zlib`
- `libyaml`
- `make`
- `gcc`
- `pkg-config`
- `libgeoip` [OPZIONALE ma CONSIGLIATO]
- `liblua5.1` [OPZIONALE ma CONSIGLIATO]
- `libhiredis` [OPZIONALE ma CONSIGLIATO]
- `libevent` [OPZIONALE ma CONSIGLIATO]

Perciò possiamo procedere nell'installazione delle dipendenze con il seguente comando:

- `$ sudo apt-get install build-essential libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev pkg-config zlib1g zlib1g-dev libpcap-ng-dev libcap-ng0 make libmagic-dev libgeoip-dev liblua5.1-dev libhiredis-dev libevent-dev python-yaml rustc cargo libpcre2-dev`

Nel caso di eventuali errore, seguire i messaggi che verranno mostrati a schermo e modificare il comando in base ai suggerimenti che vi verranno dati.

A questo punto scarichiamo il file *tar.gz* dal sito ufficiale (`https://suricata.io/download/`) e decomprimiamolo con il comando:

- `$ tar xzvf suricata-6.0.4.tar.gz`

In questo momento, l'ultima versione rilasciata è la 6.0.4, ma basta sostituire quel valore con la versione che avete scaricato voi e che sarà indicata nel nome del file.

Entriamo all'interno della cartella:

- `$ cd suricata-6.0.4`

Eseguiamo lo script di configurazione:

- `$ sudo ./configure`

Invochiamo il seguente comando per effettuare la build del software:

- `$ sudo make`

Ed infine, installiamo *Suricata*:

- `$ sudo make install`

La procedura potrebbe richiedere qualche minuto. Una volta terminata sarà possibile iniziare ad utilizzare l'IDPS sfruttando tutte le sue funzionalità.

Binary Packages

Questa modalità è la più semplice, poiché non richiede di installare manualmente le dipendenze e permette di installare *Suricata* utilizzando un unico comando:

- `$ sudo apt-get install suricata`

Come prima, una volta terminata l'installazione, sarà possibile cominciare ad utilizzare immediatamente l'IDPS e le sue funzionalità.

A.2 Installazione di TLS-Attacker

Per poter essere eseguito correttamente e senza problemi, il tool *TLS-Attacker* richiede due prerequisiti installati sulla propria macchina:

- la versione 8 o superiore di Java JDK;
- il software *Maven*.

Andiamo con ordine e procediamo alla verifica della versione di Java JDK installata sulla nostra macchina. Su riga di comando digitiamo:

- `$ java -version`

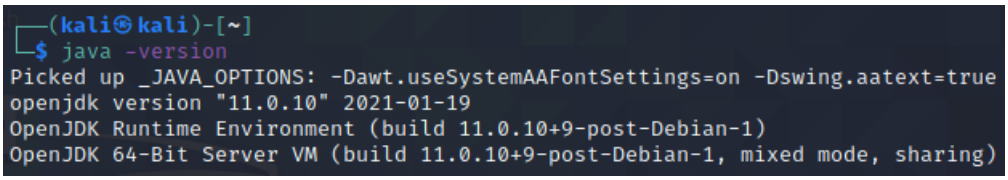

Figura A.1. Verifica della versione Java JDK.

Il risultato del comando ci indica la versione corrente di Java JDK, che in questo caso è la versione 11.0.10. Nel caso la versione installata sulla vostra macchina non fosse sufficiente è possibile procedere con l'installazione di OpenJDK 8 procedendo come segue. Per prima cosa aggiorniamo i repository con il comando:

- `$ sudo apt-get update`

E successivamente procediamo con l'installazione:

- `$ sudo apt-get install openjdk-8-jdk`

Terminata l'installazione, possiamo eseguire nuovamente il comando di verifica della versione per avere la conferma di aver installato correttamente OpenJDK 8.

Per l'installazione del software *Maven* possiamo procedere in modo simile, lanciando direttamente il seguente comando:

- `$ sudo apt-get install maven`

É possibile che l'installazione occupi un pò più di tempo rispetto a quella precedente.

Infine, una volta terminata, possiamo procedere all'installazione vera e propria del tool *TLS-Attacker*.

Iniziamo clonando il repository di *TLS-Attacker* sulla nostra macchina:

- `$ sudo git clone https://github.com/tls-attacker/TLS-Attacker.git`

Entriamo nella nuova cartella che è stata appena creata:

- `$ cd TLS-Attacker`

Ed infine, tramite *Maven*, lanciamo l'installazione:

- `$ mvn clean install`

A questo punto l'installazione del tool è terminata ed è possibile procedere nel suo utilizzo. Nella sezione successiva vedremo come creare delle chiavi e dei certificati grazie ad uno script presente all'interno di *TLS-Attacker*. Si tratta di un'operazione facoltativa, che non pregiudica l'utilizzo del tool, ma che permette di eseguire, nel caso fosse necessario, altre operazioni che prevedono l'utilizzo di chiavi e certificati (come, ad esempio, l'esecuzione di un server).

A.2.1 Creazione di chiavi e certificati

Nel caso sia necessario creare delle chiavi e dei certificati da utilizzare, ad esempio, per l'esecuzione dei server; è possibile sfruttare lo script messo a disposizione da *TLS-Attacker* che ci permette di creare sia delle chiavi sia dei certificati di diverso tipo.

Le chiavi e i certificati che possono essere creati sfruttando tre algoritmi di crittografia differenti:

- DSA a 512, 1024 e 2048 bit;
- RSA a 512, 1024, 2048 e 4096 bit;
- EC a 192, 256, 384 e 521 bit.

Dove le sigle si riferiscono a:

DSA: Digital Signature Algorithm;

RSA: dalle iniziali di Rivest, Shamir e Adleman, gli inventori dell'algoritmo;

EC: Elliptic Curve.

Per prima cosa, se non è già stato fatto seguendo l'Appendice A.2, è necessario clonare il repository di *TLS-Attacker* sulla propria macchina:

- `$ sudo git clone https://github.com/tls-attacker/TLS-Attacker.git`

Entriamo nella cartella *resources*:

- `$ cd TLS-Attacker/resources`

Modifichiamo i permessi di accesso al file eseguibile *keygen.sh*:

- `$ sudo chmod 777 keygen.sh`

Ed infine eseguiamo lo script per la creazione delle chiavi e dei certificati:

- `$ sudo ./keygen`

Una volta conclusa l'esecuzione dello script, all'interno della cartella potremo trovare undici coppie di file chiave-certificato:

- *dsa512key.pem* e *dsa512cert.pem*
- *dsa1024key.pem* e *dsa1024cert.pem*
- *dsa2048key.pem* e *dsa2048cert.pem*
- *ec192key.pem* e *ec192cert.pem*
- *ec256key.pem* e *ec256cert.pem*
- *ec384key.pem* e *ec384cert.pem*
- *ec521key.pem* e *ec521cert.pem*
- *rsa512key.pem* e *rsa512cert.pem*
- *rsa1024key.pem* e *rsa1024cert.pem*
- *rsa2048key.pem* e *rsa2048cert.pem*
- *rsa4096key.pem* e *rsa4096cert.pem*

A questo punto è sufficiente scegliere le chiavi ed i certificati che preferiamo e cominciare ad utilizzarli.

A.3 Installazione di Metasploit

Per l'installazione di *Metasploit* framework, è possibile procedere secondo due modalità ed entrambe verranno presentate in questa sezione.

La prima modalità consiste nel seguire le istruzioni presenti nella sezione *Download* del sito `https://www.metasploit.com/`. Le quali consigliano di invocare il seguente script che configurerà i pacchetti per i sistemi Linux e macOS:

- `$ sudo curl https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/templates/metasploit-framework-wrappers/msfupdate.erb > msfinstall && chmod 755 msfinstall && ./msfinstall`

Una volta conclusa l'installazione sarà possibile lanciare *msfconsole* da un finestra del terminale:

- `$ sudo /opt/metasploit-framework/bin/msfconsole`

Oppure, potrebbe essere già presente nel nostro *PATH* e basterà lanciarlo direttamente.

É possibile, inoltre, aggiornare i pacchetti con il comando:

- `$ sudo msfupdate`

La seconda modalità, a mio parere più semplice, non prevede l'invocazione di alcuno script ma l'utilizzo di *apt* ("Advanced Packaging Tool"), il gestore standard di pacchetti software della distribuzione GNU/Linux Debian. Sarà sufficiente lanciare il seguente comando:

- `$ sudo apt install metasploit-framework`

e attendere qualche minuto che l'installazione sia conclusa. Non è necessario fare altro, il programma sarà eseguibile ed utilizzabile fin da subito.

A.4 Installazione di padding-oracle-attacker

Al fine di poter eseguire il tool *padding-oracle-attacker* senza problemi, è necessario che sulla macchina sia presente *Node.js*, un runtime Javascript.

A sua volta per poter installare *Node.js* è molto utile avere *nvm*, uno script bash compatibile attraverso il cui possiamo gestire più versioni del runtime Javascript. Perciò, per prima cosa eseguiamo il comando per installare lo script:

- `$ sudo curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash`

Al momento, la versione `v0.39.0` è la più recente di *nvm*. É tuttavia possibile che vengano rilasciate nuove versioni col tempo, sarà dunque sufficiente sostituire questo valore con il valore della versione più recente.

Una volta completato l'esecuzione, possiamo proseguire con l'installazione di *Node.js*. Utilizzeremo lo script *nvm* con il comando:

- `$ nvm install node`

Questo comando procederà all'installazione dell'ultima versione del runtime Javascript. Se invece si vuole installare una versione specifica di *Node.js* basterà sostituire la parola `node` con il relativo numero di versione.

Finalmente è possibile procedere con l'installazione del tool *padding-oracle-attacker*, perciò lanciamo il comando:

- `$ npm install --global padding-oracle-attacker`

Infine, completata l'installazione, sarà necessario chiudere e riaprire il terminale per poter utilizzare il tool. Per lanciare il tool da linea di comando è possibile utilizzare uno dei seguenti alias:

- `padding-oracle-attacker`
- `padding-oracle-attack`
- `poattack`

Appendice B

Link utili

B.1 Link ai programmi

In questa sezione sono raccolti, in ordine alfabetico, i link a tutti i software, script, file *.iso* e ogni altro genere di programma utilizzato in questo lavoro.

Strumenti di analisi del traffico

Suricata: `https://suricata.io/download/`

Wireshark: `https://www.wireshark.org/download.html`

Zeek: `https://zeek.org/get-zeek/`

Strumenti di attacco

Ettercap: `https://www.ettercap-project.org/downloads.html`

Google Gruyere: `https://google-gruyere.appspot.com/`

Metasploit: `https://www.metasploit.com/download`

mitmproxy: `https://mitmproxy.org/`

PadBuster: `https://github.com/AonCyberLabs/PadBuster`

padding-oracle-attacker: `https://github.com/KishanBagaria/padding-oracle-attacker`

TLS-Attacker: `https://github.com/tls-attacker/TLS-Attacker`

Macchine virtuali

Kali Linux 2021.1: `https://cdimage.kali.org/kali-2021.1/`, scaricare il file chiamato `kali-linux-2021.1-live-amd64.iso`

Ubuntu 12.04.4: `http://old-releases.ubuntu.com/releases/12.04.4/`, scaricare il file chiamato `ubuntu-12.04.4-desktop-amd64.iso`

Ubuntu 15.10: `http://old-releases.ubuntu.com/releases/15.10/`, scaricare il file chiamato `ubuntu-15.10-desktop-amd64.iso`

Bibliografia

[1] C.Allen, T.Dierks, "The TLS Protocol Version 1.0", RFC-2246, January 1999, DOI 10.17487/RFC2246

[2] T.Dierks, E.Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC-4346, April 2006, DOI 10.17487/RFC4346

[3] E.Rescorla, T.Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC-5246, August 2008, DOI 10.17487/RFC5246

[4] E.Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", RFC-8446, August 2018, DOI 10.17487/RFC8446

[5] D.Cooper, S.Santesson, S.Farrell, S.Boeyen, R.Housley, W.Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC-5280, May 2008, DOI 10.17487/RFC8446

[6] S.Santesson, P.Hallam-Baker, "Online Certificate Status Protocol Algorithm Agility", RFC-6277, June 2011, DOI 10.17487/RFC6277

[7] R.Rivest, "The MD5 Message-Digest Algorithm", RFC-1321, April 1992, DOI 10.17487/RFC1321

[8] S.Kille, M.Wahl, A.Grimstad, R.Huber, S.Sataluri "Using Domains in LDAP/X.500 Distinguished Names", RFC-2247, January 1998, DOI 10.17487/RFC2247

[9] C.Boyd, A.Mathuria, D.Stebila, "Transport Layer Security Protocol" nel libro "Protocols for Authentication and Key Establishment", a cura di C.Boyd, A.Mathuria, D.Stebila, Springer, 2019, DOI 10.1007/978-3-662-58146-9

[10] D.Berbecaru, A.Lioy, "On the Robustness of Applications Based on the SSL and TLS Security Protocols" nel libro "Public Key Infrastructure", a cura di J.Lopez, P.Samarati, J.L.Ferrer, Springer, 2007, DOI 10.1007/978-3-540-73408-6_18

[11] G.Boyle, K.Paterson, "20 years of Bleichenbacher attacks", Technical Reports RHUL-ISG-2019-1, Information Security Group, Royal Holloway University of London (2019)

[12] D.Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1" nel libro "Advances in Cryptology - CRYPTO '98" a cura di H.Krawczyk, Springer, 1998, pp. 1-12, DOI 10.1007/BFb0055716

[13] "Bleichenbacher Attack Explained", https://medium.com/@c0D3M/bleichenbacher-attack-explained-bc630f88ff25

[14] T.Jager, J.Schwenk, J.Somorovsky, "Practical Invalid Curve Attacks on TLS-ECDH" nel libro "Computer Security – ESORICS 2015" a cura di G.Pernul, P.Y A Ryan, E.Weippl, Springer, 2015, pp. 407-425, DOI 10.1007/978-3-319-24174-6_21

[15] B.Kaliski, "PKCS #1: RSA Encryption Version 1.5", RFC-2313, March 1998, DOI 10.17487/RFC2313

[16] "Invalid Curve Attack affecting openpgp package", https://security.snyk.io/vuln/SNYK-JS-OPENPGP-460225

[17] "You could have invented that Bluetooth attack", https://blog.trailofbits.com/2018/08/01/bluetooth-invalid-curve-points/

[18] C,Meyer, J.Somorovsky, E.Weiss, J,Schwenk, S.Schinzel, E.Tews, "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks", 23rd USENIX Security Symposium (USENIX Security 14) San Diego, CA, 2014, pp. 733-748, ISBN: 978-1-931971-15-7

[19] G.Bertoletti, "Side Channel Attack contro il protocollo SSL", Webbit Padova, Italia, May 2013, pp. 1-47,

[20] E.Sohl, "Cryptopals: Exploiting CBC Padding Oracles", February 2021, https://research.nccgroup.com/2021/02/17/cryptopals-exploiting-cbc-padding-oracles/

[21] N.Sullivan, "Padding oracles and the decline of CBC-mode cipher suites", February 2016, https://blog.cloudflare.com/padding-oracles-and-the-decline-of-cbc-mode-ciphersuites/

[22] "Padding oracle attacks: in depth", https://blog.skullsecurity.org/2013/padding-oracle-attacks-in-depth

[23] B.Canvel, A.Hiltgen, S.Vaudenay, M.Vuagnoux, "Password Interception in a SSL/TLS Channel", nel libro "Advances in Cryptology - CRYPTO 2003" a cura di B.Boneh, Springer, 2003, Vol. 2729, DOI 10.1007/978-3-540-45146-4_34

[24] B.Kiprin, "What is SSL Lucky13 attack and how to prevent it from happening", April 2021, https://crashtest-security.com/prevent-ssl-lucky13/

[25] CVE-2016-2107, 29 gennaio 2016, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2107

[26] B.Moller, T.Duong, K.Kotowicz, "This POODLE Bites: Exploiting The SSL 3.0 Fallback", Google, September 2014

[27] "POODLE Vulnerability Puts Online Transactions At Risk", Trend Micro, October 2014, https://www.trendmicro.com/en_us/research/14/j/poodle-vulnerability-puts-online-transactions-at-risk.html

[28] I.Ristic, "Poodle Bites TLS", December 2014, https://blog.qualys.com/product-tech/2014/12/08/poodle-bites-tls

[29] R.duToit, "Technical Brief - TLS Vulnerabilities SSLV 4.x Mitigation and Protection", 2018, https://docs.broadcom.com/doc/tls-vulnerabilities-en

[30] I.Ristic, "CRIME: Information Leakage Attack against SSL/TLS", September 2012, https://blog.qualys.com/product-tech/2012/09/14/crime-information-leakage-attack-against-ssltls

[31] "CRIME SSL/TLS attack", Acutinex https://www.acunetix.com/vulnerabilities/web/crime-ssl-tls-attack/

[32] A.Balasinor, "SSL/TLS attacks: Part 2 - CRIME Attack", December 2013, https://niiconsulting.com/checkmate/2013/12/ssltls-attacks-part-2-crime-attack/

[33] A.Prodromou, "TLS Security 6: Examples of TLS Vulnerabilities and Attacks", Acutinex, March 2019, urlhttps://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/

[34] Y.Gluck, N.Harris, A.Prado, "BREACH: REVIVING THE CRIME ATTACK", Black Hat conference, July 2013

[35] M.Vanhoef, T.Van Goetham, "HEIST: HTTP Encrypted Information can be Stolen through TCP-windows", South seas, ABE, August 2016

[36] N.Drucker, S.Gueron, "Selfie: reflections on TLS 1.3 with PSK", J Cryptol, Vol. 34, No. 27, May 2021, DOI 10.1007/s00145-021-09387-y

[37] The Heartbleed bug, https://heartbleed.com/

[38] J.Fruhlinger, "What is the Heartbleed bug, how does it work and how was it fixed?", September 2017, https://www.csoonline.com/article/3223203/what-is-the-heartbleed-bug-how-does-it-work-and-how-was-it-fixed.html

[39] Qualys SSL Labs, https://www.ssllabs.com/ssl-pulse/

[40] nccgroup, "EARLY CCS ATTACK ANALYSIS", June 2014, https://research.nccgroup.com/wp-content/uploads/2020/07/earlyccs-advisory.pdf

[41] M.Kikuchi, "How I discovered CCS Injection Vulnerability (CVE-2014-0224)", Lepidum, June 2014, http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html

[42] A.Langley, "Early ChangeCipherSpec Attack", June 2014, https://www.imperialviolet.org/2014/06/05/earlyccs.html

[43] N.Cuppens, F.Cuppens, J.L.Lanet, A.Legay, J.Garcia-Alfaro, "CVE-2014-6593: Early Finished (Server Impersonation)" nel libro "Risks and Security of Internet and Systems" a cura di N.Cuppens, F.Cuppens, J.L.Lanet, A.Legay, J.Garcia-Alfaro, Springer, 2020, pp. 98, DOI 10.1007/978-3-319-76687-4

[44] Red Hat Customer Portal, CVE-2014-6593, January 2015, https://access.redhat.com/security/cve/cve-2014-6593

[45] Intrusion Detection System, https://en.wikipedia.org/wiki/Intrusion_detection_system
[46] Zeek, https://zeek.org/
[47] Zeek documentation, https://docs.zeek.org/en/master/
[48] Suricata user guide, https://suricata.readthedocs.io/en/latest/
[49] Wireshark user's guide, https://www.wireshark.org/docs/wsug_html_chunked/
[50] Mitmproxy docs, https://docs.mitmproxy.org/stable/
[51] Metasploit Get Started, https://www.metasploit.com/get-started