Ringraziamenti:

Desidero ringraziare il prof. Cerroni, relatore di questo elaborato, per la disponibilità dimostratami, e per l'aiuto fornito durante la stesura.

Un sentito ringraziamento ai miei genitori, che, con il loro sostegno sia morale che economico, mi hanno permesso di raggiungere questo traguardo.

Desidero inoltre ringraziare i fondatori del gruppo d'interesse sulla sicurezza informatica "Ce.Se.NA Network & Application", ed in particolare Marco, per avermi dato la possibilità di collaborare ad un progetto ambizioso come IENA.

Un ultimo ringraziamento va ai compagni di studi e agli amici per essermi stati vicino sia nei momenti difficili che nei momenti felici.

ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

SPERIMENTAZIONE ED ESTENSIONE DEL MODELLO IENA PER LA SICUREZZA DI UNA RETE LOCALE

Introduzione

L'informatica e le reti di telecomunicazioni hanno oramai una lunga storia alle spalle. I primi calcolatori nascono negli anni '40 e poco tempo dopo vedono la luce i primi linguaggi di programmazione. La storia delle reti di calcolatori ha inizio con il progetto *ARPANET* voluto fortemente dal Dipartimento della Difesa degli Stati Uniti. Inizialmente il progetto fu sviluppato da laboratori di ricerca e aziende in collaborazione con le maggiori università degli Stati Uniti, e portò alla costruzione della prima rete geografica. L'obiettivo principale della prima rete di calcolatori era quello di realizzare un sistema che consentisse lo scambio di informazioni e la condivisione di risorse (dati), tra macchine lontane kilometri una dall'altra con una velocità inedita fino a quel momento. Con il passare degli anni, uno studio di ricerca molto attivo, e la possibilità di disporre di sempre più avanzate risorse di calcolo e soprattutto la diffusione esponenziale dell'uso dei calcolatori ha posto la comunicazione di rete tra i requisiti fondamentali per lo sviluppo dell'economia di un paese. La nascita di Internet è diretta conseguenza della necessità di espandere l'interconnessione di calcolatori a reti sempre più ampie e della necessità di possedere uno strumento che potesse portare i tradizionali servizi e attività economiche alla portata di tutti. Negli stessi anni però, emerge un nuovo fenomeno di criminalità che va via via diffondendosi attraverso varie tipologie di malintenzionati che mirano trarre profitto danneggiando economicamente e non, società ed enti che offrono i loro servizi su Internet. I termini *"hacker", "crackers" e "lamer"*, descrivono tre livelli di conoscenza tecnica dei criminali che operano in rete. Con lo sviluppo di Internet e la diffusione dei dei servizi web, il numero delle azioni illecite crebbe talmente tanto da rendere necessaria una regolamentazione nell'uso della rete e l'impiego di sempre più avanzati sistemi di sicurezza. Un sistema di sicurezza è un insieme di servizi che proteggono un sistema informatico costituito da una o più reti di calcolatori. Esso ha come principale obiettivo la protezione delle risorse custodide dal sistema informatico e far si che non venga compromesso il funzionamento dei servizi che il sistema informatico offre. Perché si realizzi tutto questo, il sistema di sicurezza

offre un continuo controllo sugli accessi alla rete dall'esterno, evitando l'accesso a persone o entità non autorizzate. Un'analisi condotta recentemente negli Stati Uniti, ha evidenziato che la metà degli attacchi subiti dai sistemi informatici delle reti aziendali provengono dall'esterno, nonché da Internet, ad opera di malintenzionati dotati tecnicamente, con differenti modalità d'attacco. La tipologia d'attaccante definita dal termine "*hackers*", citato precedentemente, indica in realtà l' insieme di persone che non compiono degli attacchi veri e propri, ma che provano piacere nello scomporre i sistemi software per verificare la presenza di possibili punti deboli o falle di sicurezza. Generalmente però, il termine *"hacker"* è stato usato per indicare un'altra tipologia di attaccante: il *"cracker"*. Lo scopo principale di un *"cracker"* è quello di trarre direttamente profitto dalla manomissione software dei suoi sistemi vittima, avvalendosi di ottime conoscenze tecniche nel campo, a differenza del *"lamer"* che è poco dotato tecnicamente, spesso un adolescente, che con l'utilizzo di software molto comuni in rete cerca di procurare danni o estorcere informazioni per pura gloria e divertimento.
Le tipologie d'attacco si differenziano tra:

- La diffusione di *"malware"* (come virus, worm);
- Il *"denial of service"*;
- L'accesso illecito alle risorse di un sistema.

I *"malware"* non sono altro che dei software che riescono a modificare e/o distruggere dei file in un sistema. Vengono diffusi principalmente attraverso servizi come la posta elettronica e tool che permettono lo scambio diretto di files. Un esempio di *"malware"* ben conosciuto dalla comunità mondiale sono i "*virus*".
Il *"denial of service"* altro non è che il sabotaggio dei servizi offerti da un sistema informatico. Nella maggior parte dei casi, l'attacco è costituito dall'invio contemporaneo di una numerosa quantità di richieste ad un server, in modo da bloccarlo per saturazione.
L'accesso illecito alle risorse di un sistema d'informazione può anch'esso avere come scopo la distruzione o la copia di dati, ma talvolta viene sfruttato per trarre direttamente benefici economici nelle transazioni di denaro.

Purtroppo non esistono tecniche di difesa tali da poter rendere sicura al 100% una rete di calcolatori contro gli attacchi provenienti dall'esterno, ma esistono numerose tecniche per la difesa delle reti più o meno efficaci a seconda del contesto in cui è inserita la rete e dal tipo di attacchi ai quali è esposta. I tradizionali strumenti software di sicurezza come Antivirus, Firewall, Intrusion Detection System (IDS), Intrusion Prevention System (IPS), si basano su un modello orientato a politiche di difesa che potremmo definire *"chiuse"*, come l'accesso limitato e strettamente controllato alle risorse mediante selezione dei servizi essenziali (*hardening* di rete) e l'aggiornamento continuo dei sistemi, del software e della conoscenza delle possibili vulnerabilità. Tuttavia aggiornare Antivirus, installare firewall e IDS/IPS non sempre risulta essere il modo migliore per fronteggiare tutti i pericoli ai quali una rete è esposta su Internet.

Ai tradizionali paradigmi di attacco/difesa per la sicurezza informatica citati sopra, si contrappone uno schema alternativo, proposto da Marco Ramilli, ingegnere informatico dell'Università degli studi di Bologna. Lo schema proposto utilizza un approccio opposto agli attuali canoni di sicurezza, implementando una politica preventiva che potremmo definire di *"totale apertura"* per combattere attacchi diretti alla *"service exploitation"*. Tale modello è stato chiamato "*IENA*" e può essere applicato ad una rete di calcolatori (o ad una singola macchina) in concomitanza con altri strumenti di sicurezza esistenti. Un approccio di questo genere incrementerebbe notevolmente l'affidabilità ed il livello di sicurezza dei sistemi informativi e di comunicazione che fanno uso della rete Internet. IENA non ha la presunzione di eliminare in modo definitivo la minaccia d'attacchi esterni, ma costituisce sicuramente una grossa difficoltà da superare per hackers, crackers e lamer.

Capitolo 1

"Io sono me più il mio ambiente
e se non preservo quest'ultimo
non preservo me stesso."
(José Ortega y Gasset)

TECNICHE E MODELLI CLASSICI PER LA SICUREZZA DI RETE

In questo capitolo saranno analizzate alcune delle innumerevoli tecniche sviluppate e commercializzate nel mondo per proteggere una rete di calcolatori dalle intrusioni. Verrà inoltre definito il paradigma o modello classico nel quale sono applicate le tecniche di difesa descritte.

1.1 Progettazione di rete orientata alla sicurezza

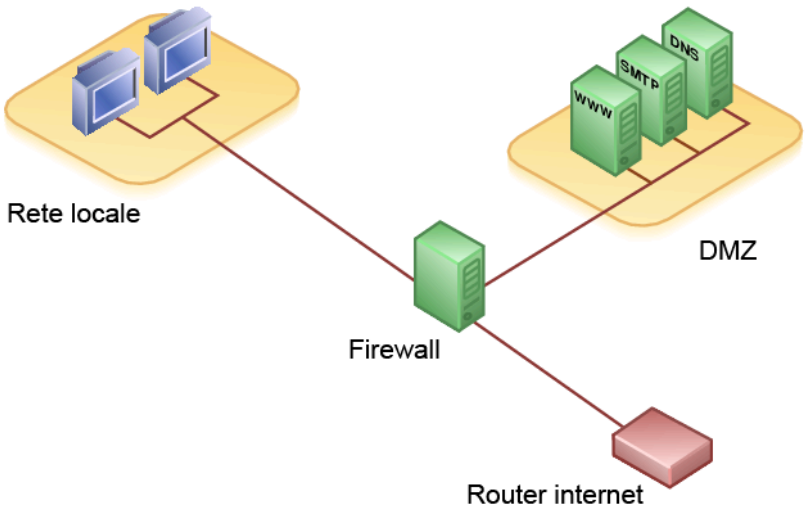


Fig. 1: Rete aziendale orientata alla sicurezza [bib.21]

Progettare una rete che ha gli strumenti base per garantire un grado minimo di sicurezza, significa studiare e costruire la sua topologia e la sua struttura formale seguendo il classico schema di progettazione che prevede la presenza di una *"Demilitarized Zone (DMZ)"* logicamente separata della rete locale interna.
La divisone di una rete aziendale in due zone logiche permette di isolare gli host sui quali l'azienda offre i suoi servizi verso l'esterno. La DMZ è raggiungibile sia dalla rete locale interna che dalle reti, permette però connessioni esclusivamente verso l'esterno, quindi gli host attestati sulla DMZ non possono connettersi alla rete aziendale interna.
I passi da seguire prevedono:

1. Installazione di un Firewall perimetrale per la protezione della DMZ;
2. Installazione dei server nella DMZ della rete;
3. Dividere (se necessario) la DMZ in più sottoreti tramite switch e firewall;

La DMZ, verrà in seguito configurata con apposite restrizioni e permessi che possano garantire un adeguato livello di sicurezza.

1.2 Il firewall

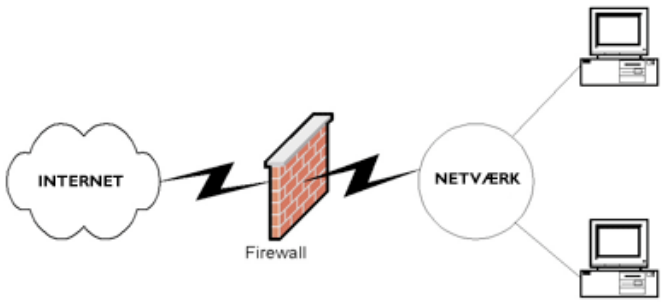


Fig. 2: Firewall

Il *"firewall"* è un sistema di difesa per gli host e per le reti largamente utilizzato nella progettazione di reti orientate alla sicurezza. Il firewall può essere implementato sia da

un software che da un dispositivo hardware, e ad oggi la maggior parte dei sistemi operativi posseggono un firewall software tra le applicazione di sistema. I firewall software effettuano un controllo di tutti i programmi che tentano delle connessioni sia in locale che verso internet e permettono all'utente di configurare le *"rules"* che lo stesso firewall deve seguire nel concedere o negare connessioni ad internet create a livello applicativo. La differenza sostanziale tra firewall software e firewall perimetrali è rappresentata dai parametri utilizzati per discriminare i pacchetti sotto controllo. Nei firewall software è l'utente che decide sul momento quali applicazioni possono creare connessioni verso l'esterno tramite il protocollo IP, nei firewall perimetrali la discriminazione avviene in base ad indirizzo IP sorgente, porta e indirizzo IP di destinazione.

Nelle reti locali, al firewall viene destinata una posizione strategica per il controllo del traffico e il monitoraggio dei pacchetti che lo compongono. Ha diverse funzionalità, a seconda delle quali viene classificato in diverse categorie: **Packet filter, Stateful inspection, Application layer firewall.**

Il *"packet filter"* analizza direttamente gli header dei pacchetti, e fungendo da filtro, in base alle *"rules"* specificate dall'amministratore, ne permette il transito oppure scarta gli stessi impedendone l'accesso alla rete locale che costituisce il sistema da proteggere. Esso è in grado quindi di classificare i pacchetti in base al protocollo, discriminandoli tra quelli appartenenti ad una connessione già stabilita, e quelli appartenenti ad una connessione non ancora creata.

Lo "*Stateful inspection"*, riesce a ricostruire lo stato di tutte le connessioni aperte, riconoscendo talvolta pacchetti TCP che non appartengono a nessuna delle connessioni e che quindi verranno scartati.

Gli *"Application Layer Firewall"* hanno la capacità di analizzare le connessioni a livello di Applicazione. Un esempio molto diffuso è rappresentato dai *"proxy"* che permettono di intercettare in modo selettivo tutte le connessioni che vanno dalla rete privata alla rete pubblica e viceversa permettendone alcune e impedendone altre. La

tipica collocazione di un *"proxy"* prevede che faccia da tramite tra sottoreti private e pubbliche. Ne esiste anche una generazione che riesce ad analizzare le connessioni fino al settimo strato del modello OSI riconoscendo e scartando i pacchetti contenenti virus o worm o altro malware.

Il modo più semplice per realizzare un firewall dedicato prevede l'utilizzo di una macchina con due schede di rete e un sistema operativo come Linux.

Alcuni firewall possono effettuare operazioni di *"logging"* sia per fini statistici che per tenere traccia dei problemi diagnosticati registrando i pacchetti rifiutati e talvolta favorendo la rilevazione di intrusioni.

1.3 Arpwatch

Arpwatch è un ottimo strumento per monitorare il traffico ARP in una rete locale.

Arpwatch va installato su un server per il monitoraggio in modo tale da poter in qualsiasi momento intercettare i broadcast ARP che giungono sulla scheda di rete. L'obiettivo principale di Arpwatch è quello di rilevare tutte le modifiche del mac address della scheda di rete di un nuovo host. Il monitoraggio dei mac address permette di scoprire tempestivamente se un IP passa da un host ad un altro. Nelle segnalazioni delle modifiche vengono sempre specificati indirizzo IP, vecchio mac address e nuovo mac address. La notifica delle suddette variazioni può avvenire tramite e-mail o altro.
Potendo diagnosticare attività di *"arp poisoning"* in rete, Arpwatch diventa uno strumento essenziale per alzare notevolmente il livello di sicurezza delle reti.

1.4 Intrusion Detection System (IDS)

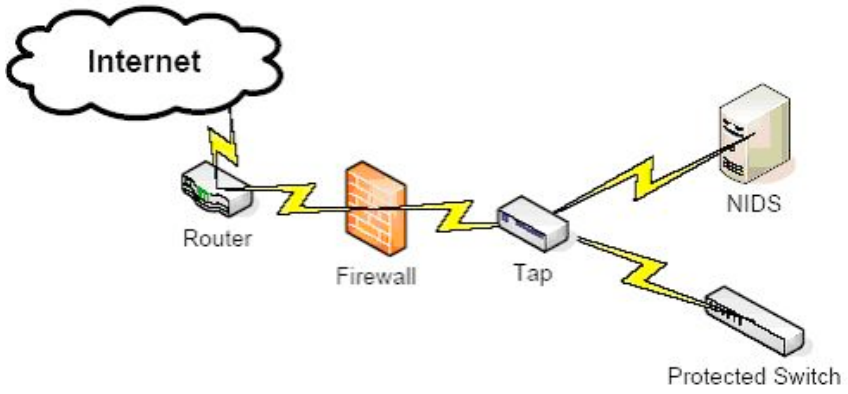


Fig. 3: Network IDS [bib.23]

L'**Intrusion Detection System (IDS)** è il più comune dispositivo per l'identificazione delle intrusioni nelle reti locali. Un IDS è capace di rivelare attacchi di tipo "denial of service", "malware" ed accessi non autorizzati. Un IDS hardware è generalmente costituito da una console per il monitoraggio della rete; da dei sensori che ricevono tutte le informazioni relative agli host e alla stessa rete; da un componente chiamato "motore" che ha il compito di individuare le falle di sicurezza interpretando opportunamente le informazioni che giungono dai sensori. Tutto il sistema IDS fa riferimento ad un database interno per identificare istantaneamente le eventuali tipologie di intrusioni. A seconda delle funzionalità che svolgono, abbiamo differenti classi di sistemi IDS: sistemi che operano negli host, a livello di rete e sistemi ibridi.

Per esempio per "**Network Intrusion Detection System"** si intende un sistema per la rivelazione delle intrusioni nelle reti locali. Un NIDS analizza il traffico e rivela un'intrusione non appena viene riscontrato del traffico anomalo. Ovviamente l'ambiente in cui opera un NIDS non può essere che una DMZ di una rete nella quali sono attivi i principali servizi che offre il sistema informatico. Un NIDS è anche capace di controllare la topologia della rete al fine di segnalarne eventuali modifiche inaspettate. Il sistema NIDS più conosciuto è *"Snort"*. Come per le reti locali, anche per i singoli

host esistono sistemi per la rivelazione delle intrusioni classificati come "**Host based Intrusion Detection System"**. Un HIDS, come per esempio *"Aide"*, esegue un controllo periodico sull'host che protegge di alcuni importanti file del sistema e di tutte le modifiche che vengono effettuate in termini di password, database degli users, permessi sui file. La rivelazione avviene soprattutto scrutando periodicamente file di log e chiamate alle system call. Un terzo tipo di IDS sono gli "**Hybrid Intrusion Detection System",** dei quali il più conosciuto è *"Prelude"*, che costituisce una via intermedia tra i due sistemi descritti in precedenza. Essi effettuano un continuo controllo sia degli host che della rete.

Un IDS, in base ai compiti che assolve, può essere attivo o passivo. Fanno parte degli IDS attivi tutti coloro i quali, rivelata l'intrusione, avviano una serie di operazioni che mirano a bloccare la stessa modificando opportune regole di rete nei firewall. Oltre a bloccare l'intrusione, un IDS attivo, notifica tutte le rilevazioni al suo amministratore di sistema sia loggando su file che inviando istantaneamente messaggi informativi (e-mail, sms). Gli IDS passivi, una volta rilevata l'intrusione, provvedono alla sola notifica.

Le tre tipologie di sistemi IDS più diffuse sono:

- Il*"signature based intrusion detection system"* rivela tutte le intrusioni che risutano tra le tipologie d'attacchi noti e registrati nel database. Lo svantaggio del sistema SBIDS consiste nell'incapacità di rilevare nuove forme di attacco o varianti se pur minime delle forme di attacco conosciute;
- Gli *"anomaly based intrusion detection system"* sopperiscono parzialmente all'incapacità dei SBIDS di rilevare le nuove tecniche di attacco analizzando il comportamento del sistema protetto con ampio utilizzo di tecnologie di intelligenza artificiale. In modo del tutto autonomo il sistema ABIDS è capace di capire se le anomalie riscontrate rappresentano dei pericoli per la sicurezza del sistema.

1.5 Intrusion Prevention System (IPS)

Gli **Intrusion Prevention System** sono spesso integrati in molti IDS, ma hanno particolari funzionalità che li contraddistingue. Un IPS infatti, ha la capacità di monitorare qualsiasi connessione bloccando quelle malevole all'esterno della rete. Ha dalla sua parte il vantaggio di controllare tutto il traffico prima che giunga nella rete protetta, ma per questo stesso motivo può rallentare notevolmente la velocità delle connessioni causando non pochi problemi. Si può dire che esso controlli gli accessi alla rete imitando sommariamente il comportamento di un firewall, ma lavorando più a livello di applicazione che a livello di rete, effettuando quindi un accurato controllo degli accessi da parte di utenti e di entità software risultando efficace nella rilevazione di *"malware"*.

1.6 Honeypot

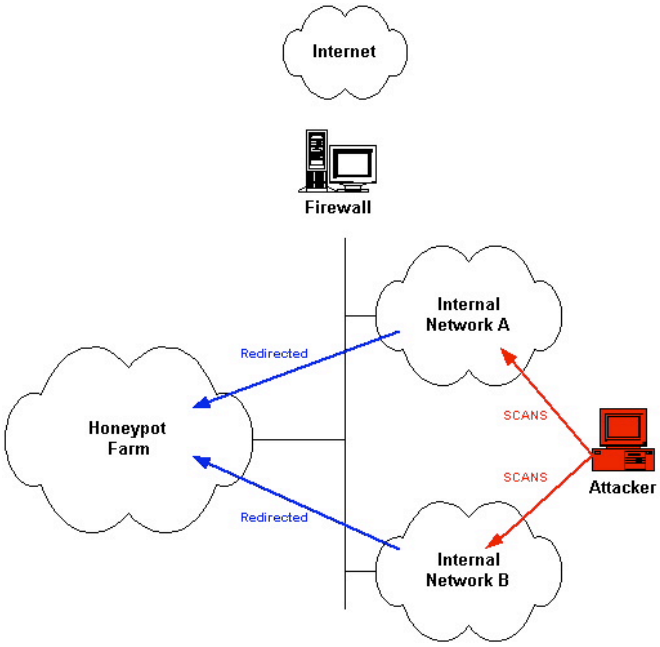


Fig. 4: Honeypot farm [bib.22]

Gli "**Honeypots"** sono particolari sistemi per lo studio e l'analisi delle tecniche d'attacco messe in atto in Internet. Essi infatti sono dei sistemi totalmente passivi, con l'obiettivo di scoprire ed analizzare gli attacchi in corso sulla rete in cui sono installati.
Le reti locali sulle quali sono installati gli Honeypot prendono il nome di Honeynet. Ogni Honeypot installato offre all'esterno un falso servizio, che funge da esca per qualunque attaccante. L'attaccante che prepara un attacco verso la Honeynet è inconsapevole del fatto che sta per essere ingannato da un sistema alternativo per lo studio delle tecniche utilizzate da hackers e crackers. Tuttavia una Honeynet può costituire un arma a doppio taglio. Infatti se malauguratamente uno di essi riuscisse a prendere il controllo della rete, potrebbe sfruttarla per effettuare nuovi attacchi e danneggiare altri sistemi informatici. Per ovviare a questo problema, l'architettura di una Honeynet prevede uno strato di "Data control" che ha l'obiettivo di lasciare il più possibile libertà d'azione agli attaccanti tenendoli sotto controllo e monitorando costantemente tutte le modifiche ai dati effettuate dai crackers. Oltre al controllo del sistema, un Honeypot deve tener conto di eventuali danni che può causare l'attaccante e

soprattutto deve far credere allo stesso di trovarsi di fronte ad un comune sistema informatico con tutte le vulnerabilità del caso. Per il recupero, l'analisi e lo studio dei dati e delle informazioni riguardanti un attacco, l'honeypot si affida allo strato di "Data capture". Questa parte dell'architettura, si occupa del salvataggio delle informazioni che saranno poi oggetto di studio, nascondendole in un sistema sicuro e generalmente lontano dalla macchina su cui agisce l'attaccante. E' facile intuire che, se l'attaccante avesse modo di scoprire di essere spiato, lo stesso potrebbe modificare le informazioni in uscita dal sistema honeypot locale. La cattura dei dati si articola in diverse operazioni, che vanno dal salvataggio di dati in uscita alla cattura del traffico di rete generato durante l'attacco.

Durante l'attacco le Honeynet possono passare attraverso diversi livelli di rischio, come già accennato, la honeynet potrebbe essere utilizzata come strumento per effettuare ulteriori attacchi, bisogna quindi gestire le intrusioni nel miglior modo possibile evitando in qualsiasi modo eventualità di questo genere. La honeynet potrebbe essere scoperta dall'attaccante, fallendo così inevitabilmente l'obiettivo principale per la quale è stata costruita. Una honeynet potrebbe subire danni a livello software, e l'attaccante prendendo il controllo del sistema potrebbe mettere fuori uso le funzionalità per la cattura e il salvataggio delle informazioni. La honeynet potrebbe infine, all'insaputa dell'amministratore di sistema, essere utilizzata per l'esercizio di attività illecite, delle quali a risponderne in prima persona sarà comunque e sempre lo stesso amministratore.

Per questi motivi, nelle honeynet non bisogna mai delegare completamente il controllo della loro attività a sistemi automatizzati per il monitoraggio, ma costantemente devono essere controllate ed analizzate dallo stesso amministratore. Spesso una attenta analisi dei dati da parte di un buon amministratore, risulta più utile dell'azione del sistema di analisi automatico dell'honeypot. Generalmente, la forza e l'efficacia di una honeynet, sta nella sua originalità e nella sua personalizzazione, poichè gli stessi cracker conoscono nei dettagli architettura e funzionamento degli honeypot. Da questo si evince che mettere in attività ed effettuare costantemente operazioni di manutenzione richiede una enorme quantità di tempo a disposizione e quindi un lavoro continuo degli amministratori della rete.

1.7 Modello “classico” per la gestione della sicurezza

Il modello *“classico”* descrive *“step to step”* quali operazioni un ICT Security Manager deve eseguire per tenere alto il livello di sicurezza di un sistema informatico. Il compito di un ICT Security manager è oneroso e spesso ripetitivo, inoltre richiede uno sforzo non indifferente per la corsa sfrenata alla caccia dell’ ultima patch e dell’ ultimo security tool. Negli ultimi anni si è sempre più automatizzata tale ricerca, ma sempre e comunque risulta essere onerosa e ridondante soprattutto per quanto riguarda la configurazione e l’ installazione dei nuovi metodi di sicurezza scaricati.
Viene riportato di seguito il diagramma UML che descrive, secondo il modello classico, l’iter delle operazioni eseguite da un ICT Security manager:

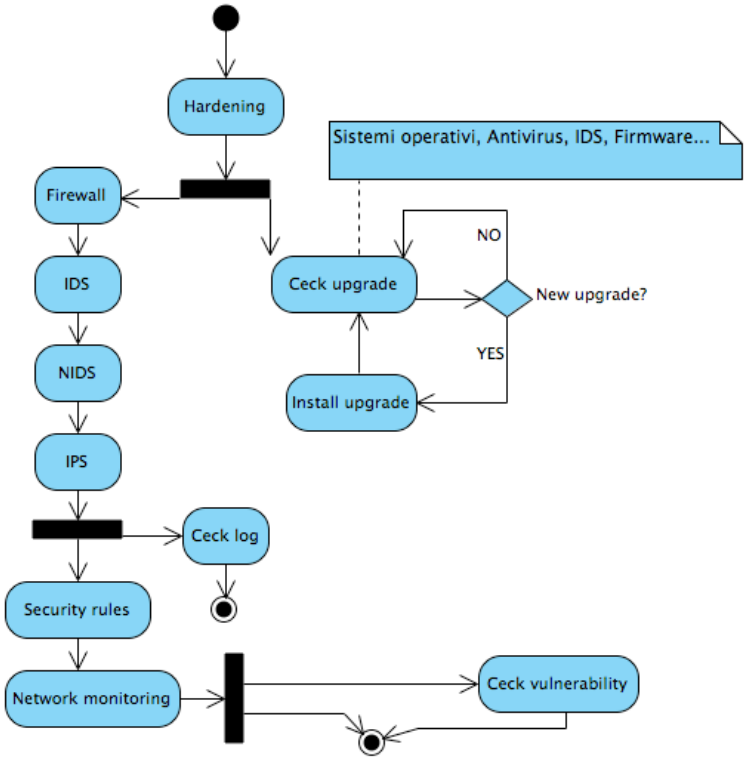


Fig. 5: Diagramma UML del modello “classico”

Capitolo 2

IL MODELLO IENA

In questo capitolo sarà illustrato il nuovo modello per la gestione della sicurezza di una rete locale denominato IENA.
IENA sovverte completamente i vecchi canoni di sicurezza, è uno strumento di prevenzione dagli attacchi esterni astuto ed innovativo.

2.1 L'idea innovativa

Sistemi di sicurezza come AntiVirus, Firewall, IDS (Intrusion Detection System), NIDS (Network IDS), proteggono i sistemi informatici attuando politiche di *"chiusura"*. Accesso controllato, comunicazioni criptate, chiusura di servizi, contribuiscono a "blindare" fisicamente gli accessi verso il sistema.
Il modello IENA si oppone al modello classico degli odierni sistemi di sicurezza, presentando una nuova concezione di sicurezza orientata alla pura prevenzione degli attacchi. *"Prevenire è meglio che curare"* diventa la parola d'ordine ed il mezzo per costruire un nuovo approccio alla sicurezza di rete, un approccio opposto a quello fino ad ora attuato nelle politiche *"chiuse"*.
L'unico modo per combattere efficacemente gli attacchi è quello di troncarli sul nascere. Prevenire significa fare tutto il possibile per evitare che un attacco inizi, questa è la semplicità e la forza del modello IENA.
Per capire cosa significa "prevenzione" in termini di sicurezza informatica, non c'è modo migliore che mettersi dalla parte dell'attaccante e studiare passo per passo le mosse che egli deve compiere prima di sferrare il vero e proprio attacco.
La prima mossa di un attaccante prevede la raccolta di informazioni sul sistema vittima, ed in particolare l'individuazione degli eventuali punti deboli che possono rappresentare dei comodi punti d'accesso per sferrare l'attacco. In questo caso, prevenire significa impedire al malintenzionato di acquisire alcuna informazione riguardo il sistema vittima, in pratica vuol dire impedire all'attaccante l'utilizzo delle comuni tecniche che

rivelano importanti informazioni sul sistema. Prima di accedere a fonti d'informazione come registri ripe, tecniche di footprinting, EDGAR research, indagini sui POC, analisi record MX, l'attaccante deve necessariamente conoscere quanti e quali servizi sono offerti all'esterno dalla macchina vittima. Il primo passo tecnico per l'acquisizione dei servizi offerti è il *"port mapping"*, e non esiste altro modo per ottenere questo tipo di informazioni, ad oggi, che non sia *l'ingegneria sociale*[1] .

Un *"port mapping"* (o *"port scanning"*), alla luce delle considerazioni precedenti, è di fondamentale importanza per un attaccante, poiché è quasi impossibile attaccare un sistema del quale non si conosce nulla. Durante un port mapping una serie di connessioni partono dalla macchina dell'attaccante verso le 65535 porte (o anche meno) dell'host vittima. Per concretizzare la prevenzione descritta dal modello IENA, l'host vittima, deve recuperare l'identità dell'attaccante (rappresentata dall'indirizzo IP), e provvedere al completo rifiuto di tutte le future connessioni generate da quell'indirizzo IP. Un provvedimento di questo tipo, rappresenta una prima difficoltà per gli attaccanti, poiché perdono, anche se temporaneamente, la possibilità di costruire attacchi dalla "postazione" utilizzata per effettuare il port mapping. Tuttavia, lo stesso attaccante è stato in grado di ottenere le informazioni che cercava, ovvero l'elenco delle porte aperte sull' host vittima.

A questo punto l'attaccante, cosciente di essere stato *"scoperto"* e *"ammanettato"*, cambia identità (indirizzo IP), e forte delle conoscenze acquisite con il port mapping decide di tentare un attacco sfruttando uno dei punti deboli rilevati.

Ammettiamo per ipotesi che tra tutte le porte aperte dell'host vittima, l'attaccante sia particolarmente attratto dalla 12345 (classica porta del famoso troyan *"NetBus"*). La ghiotta possibilità di accedere facilmente all'host vittima attraverso la porta 12345, convince l'attaccante a tentare un attacco tramite la 12345 per poter prendere possesso del sistema. Il nuovo attacco tramite la porta 12345 non andrà secondo i piani dell'attaccante, poiché la 12345 nasconde un falso servizio, ovvero una nuova trappola

[1] Ingegneria sociale: dall'inglese *social engineering* , studio del comportamento individuale di una persona al fine di carpire informazioni. Un ingegnere sociale riesce con l'inganno ad estorcere informazioni nascondendo la propria identità e rubandola alla sua vittima.

chiamata IENA, che provvederà nuovamente ad uccidere tutte le future connessioni generate dal suo nuovo IP.

2.2 Progetto

La realizzazione di uno strumento software che implementi le caratteristiche del modello IENA prevede la presenza di un vincolo sull'uso del servizio di port mapping. Di fatti uno spontaneo port mapping in futuro sarà considerata azione illecita che preannuncia un possibile attacco. La sostanziale differenza tra IENA e i classici software per la sicurezza, è l'oggetto su cui agiscono. IENA lavora su connessioni "illecite", i software classici lavorano su connessioni "lecite". La connessione "illecita" ha come end-point la porta del falso servizio che la IENA offre all'esterno.
In generale, sull'host da proteggere si potranno attivare diverse entità IENA su diverse porte per simulare falsi servizi. In questo modo non si "chiude" la rete, ma si sfrutta la conoscenza degli attacchi per anticipare le mosse dell'attaccante. Tale sistema resterà sempre indipendente dalle nuove vulnerabilità, annullando la snervante corsa al tool più aggiornato e al sistema con più patch di sicurezza.
Il software IENA opererà per sua natura in ambienti distribuiti, pertanto, a livello architetturale il sistema vedrà la cooperazione di due entità principali:

1. Un server IENA;
2. Un (o più) client IENA.

Il client IENA offrirà il falso servizio su una porta specificata dall'amministratore di rete e sarà suo compito notificare al server IENA qualsiasi richiesta di connessione dall'esterno comunicando il parametro identificatifivo dell'attaccante (indirizzo IP).
Il server IENA sarà in ascolto dei suoi client, e prenderà i dovuti provvedimenti in seguito ad una notifica d'attacco da parte di un client IENA. In particolare il server, ricevuta la notifica dal client, eseguirà le seguenti operazioni:

1. Recupero della data e dell'ora della notifica;
2. Recupero IP attaccante, IP e porta della iena attivata;
3. Log su file dei parametri dell'attacco;

4. Modifica delle regole della rete.

Un sistema esterno di notifica, indipendente da IENA, avrà il compito di analizzare periodicamente il file di log e notificare attraverso opportune modalità (e-mail, sms ecc…) gli eventuali nuovi attacchi rilevati all'amministratore di rete.

Il compito del sistema di notifica può essere assolto da uno dei tanti *"sistemi di warning"* già esistenti.

Viene riportato di seguito l'Activity Diagram che descrive gli stati del sistema IENA:

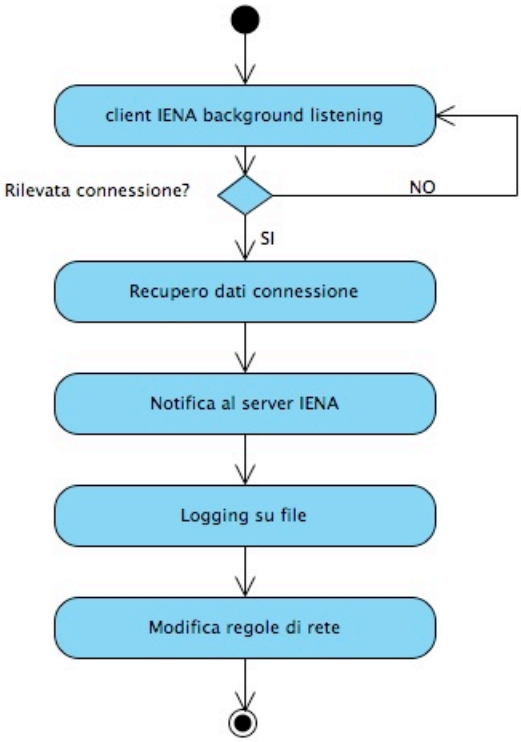


Fig. 6: Diagramma UML del modello IENA

Viene riportato di seguito il Sequence Diagram che descrive il flusso del controllo del sistema IENA:

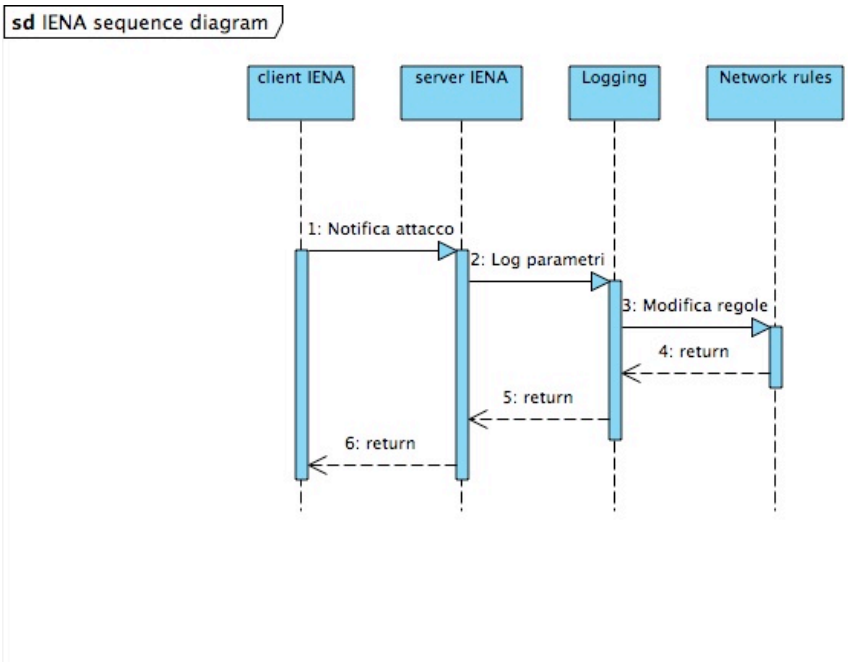


Fig. 7: Sequence diagram di IENA

2.3 Architettura

IENA, in questa prima progettazione, è rappresentato da un sistema distribuito completamente automatico e che quindi non prevede, dopo l'installazione, alcun tipo di interazione con l'amministratore del sistema.

Di seguito è riportato il class diagram che descrive l' architettura logica nel primo livello della progettazione.

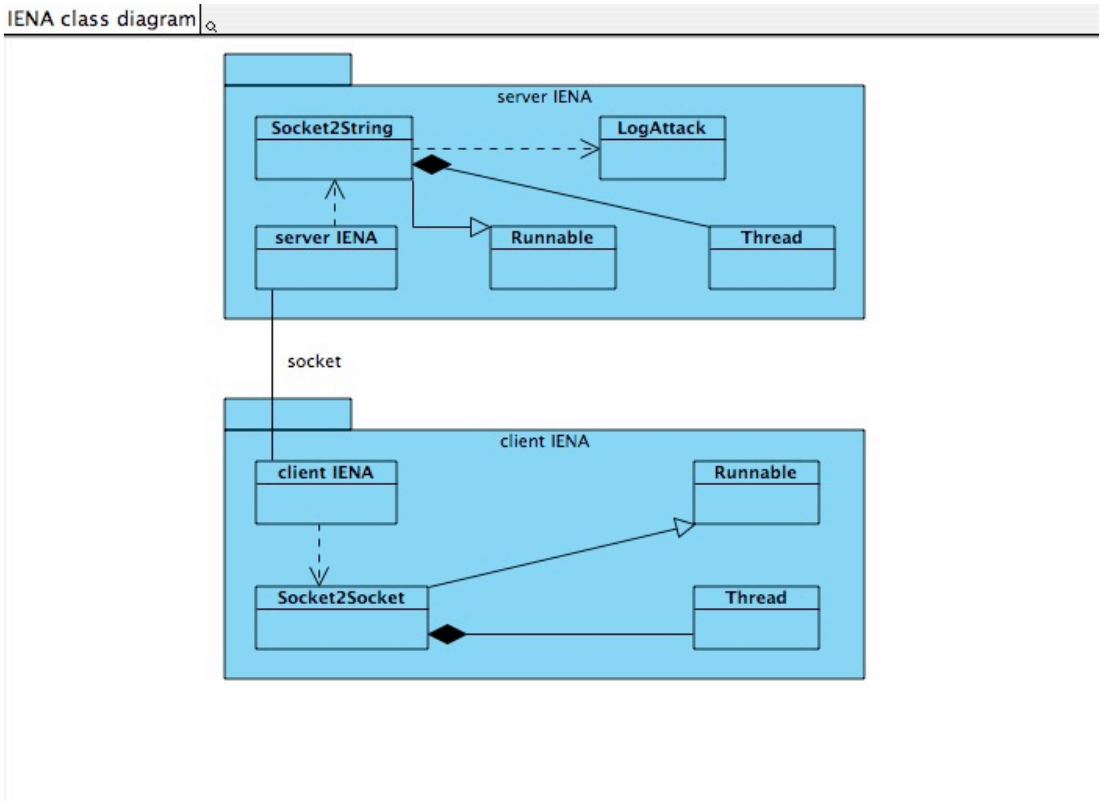


Fig. 8: Class diagram di IENA

I due package rappresentano le due identità che formano il sistema IENA:

1. Server IENA
2. Client IENA

La classe **ClientIENA** avrà un socket in ascolto sulla porta destinata al falso servizio. **ClientIENA** non appena rileverà un tentativo di connessione, invocherà la classe **Socket2Socket**, plasmata in modo tale da recuperare dal *"socket trappola"* i parametri necessari per la notifica e instaurare una nuova connessione socket con **ServerIENA** per comunicare il pacchetto informativo appena creato.

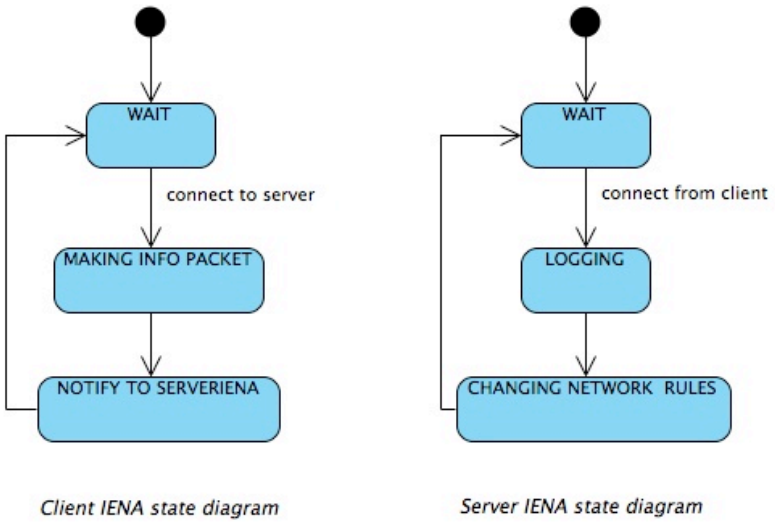
Socket2Socket implementa i thread necessari alla lettura e scrittura delle informazioni dal socket verso l'esterno al socket della connessione con il **ServerIENA**.

Il processo interno al package **ClientIENA** è perennemente nello stato di "wait" fino alla comparsa di una richiesta di connessione dall'esterno. A questo punto avviene il cambiamento di stato automatico ed immediato con il passaggio nella fase di creazione

del pacchetto informativo da inviare al server IENA. Notificato l'attacco al **ServerIENA**, **ClientIENA** transita nuovamente nello stato di wait.

La classe **ServerIENA** avrà un socket in ascolto su una porta nota ai **ClientIENA** installati sulla rete locale. Il processo interno al package **ServerIENA** è perennemente nello stato di "wait" fino alla richiesta di connessione di un **ClientIENA**. A questo punto avviene il cambiamento di stato con il passaggio nella fase di log delle informazioni su file invocando la classe **Socket2String**, plasmata in modo tale da recuperare i parametri notificati da **ClientIENA** e scrivere su file anteponendo data e ora. **Socket2String** implementa i thread necessari alla lettura delle informazioni dal socket con il **ClientIENA** e alla costruzione di un oggetto *"String"*. Successivamente al log, **ServerIENA** transiterà nello stato di modifica delle regole di rete. Terminate le modifiche alle regole di rete, **ServerIENA** transita nuovamente nello stato di wait.

Di seguito vengono riportati rispettivamente i diagrammi di stato del **ClientIENA** e del **ServerIENA**:



Client IENA state diagram *Server IENA state diagram*

Fig. 9: State diagram di IENA

2.4 IENA vs Honeypot.

Fino ad oggi, l'idea di costruire sull'inganno uno strumento per la sicurezza di una rete (o un host), è stata sviluppata da sistemi come *"Honeypot"*. L'obiettivo principale di Honeypot è quello studiare nuovi paradigmi e tecniche d'attacco spiando direttamente le mosse degli attaccanti. Intrattenere l'intruso all'interno del sistema Honeypot simulando un falso servizio è senza dubbio il modo migliore per scoprire le ultime strategie d'attacco sulla rete. Esistono diverse tipologie di HoneyPots, ma ognuna di esse è realizzata da software in locale che logga ogni singolo movimento dell'intruso.

Il modello HoneyPot, pur basandosi sull'inganno (simulazione di falsi servizi), ha finalità completamente differenti dal modello IENA.

IENA è un sistema distribuito che ha come unico obiettivo la prevenzione delle intrusioni. IENA è un software leggero, plasmato per un ambiente distribuito, indipendente dal sistema operativo, integrabile con qualunque software di notifica e manipolazione dinamica delle regole di rete.

IENA, semplifica notevolmente la gestione della sicurezza di rete. Con l'utilizzo di IENA, viene sfoltito sensibilmente l'*"activity diagram"* che descrive il modello "classico" descritto nel paragrafo 1.7.

Sistemi di sicurezza come Firewall e NIDS (Network Intrusion Detection System), potrebbero passare in secondo piano nelle esigenze per la protezione di una rete.

Viene riportato di seguito il nuovo "activity diagram", conseguenza diretta della prevenzione attuata dal modello IENA:

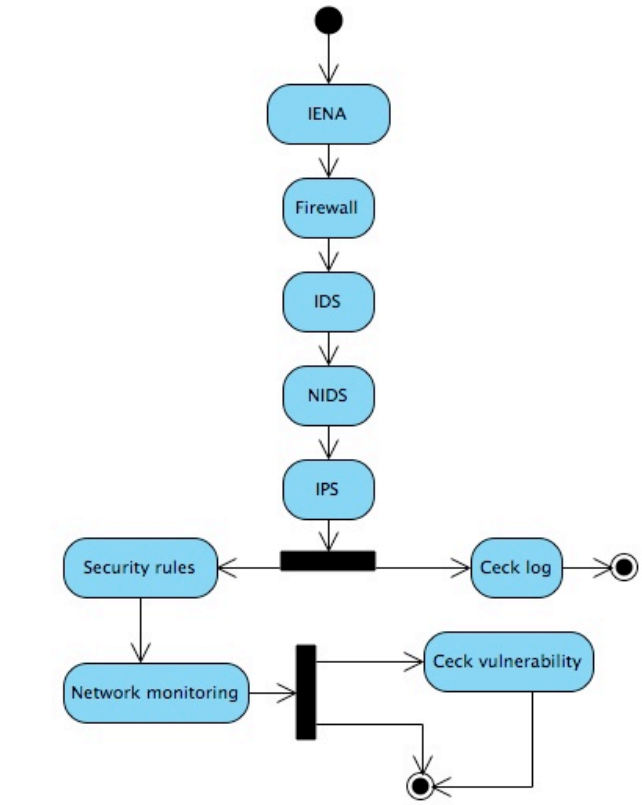


Fig. 10: Modello IENA per la sicurezza

2.5 Sviluppi ed estensioni di IENA

Il progetto principale, descritto in questo capitolo, descrive il nucleo del modello IENA. Per avere un prodotto finito, il nucleo deve essere corredato da estensioni opportune che sopprimano le eventuali vulnerabilità e che migliorino compatibilità e facilità d'uso del software. Tra gli sviluppi necessari, possiamo elencare:

1. L'implementazione di una connessione sicura attraverso il protocollo SSL/TLS, opportunamente corredata di mutua autenticazione tra ClientIENA e ServerIENA;
2. Lo sviluppo di *"plug-in"* per l'interazione tra IENA e i software di configurazione e controllo delle regole di rete;

3. Lo sviluppo di un'interfaccia grafica (GUI) per facilitare e automatizzare l'installazione e la configurazione del sistema IENA;
4. Lo sviluppo e l'integrazione di tecniche per il detecting di *"stealth scan"*;
5. Lo sviluppo e integrazione di tecniche per la differenziazione delle tipologie d'attacco;
6. La configurazione integrata di una rete basata su IENA e su Knock per i servizi attivi;
7. L'integrazione di IENA con sistemi IDS e IPS.

Capitolo 3

PROTOCOLLO SSL/TLS E CONNESSIONI SICURE

In questo capitolo sarà descritta la tecnologia scelta per la realizzazione della connessione sicura tra ClientIENA e ServerIENA, oggetto della prima estensione del progetto.

3.1 SSL (Secure Socket Layer)

Secure Sockets Layer (**SSL**) è il protocollo per la realizzazione di connessioni sicure in ambiente distribuito sviluppato negli anni ’90 dall’azienda che vanta lo sviluppo di uno dei più famosi e diffusi browser: Netscape. SSL con il passare degli anni è divenuto uno standard di fatto, e come generalmente avviene in questi casi, è stato standardizzato formalmente nel ’96 con il rilascio della sua ultima versione, la 3.0, che prende il nome di **Transport Layer Security**, alias **TLS**.
SSL/TLS è stato sviluppato per garantire la privacy nelle comunicazioni su Internet. Questo protocollo infatti, consente alle applicazioni client/server di scambiare dati prevenendo le intrusioni e qualunque tentativo di alterazione dei dati scambiati.

Il protocollo SSL/TLS basandosi sulla crittografia, fornisce un adeguato livello di sicurezza nelle comunicazioni tra le applicazioni client/server, consentendo alle stesse di comunicare prevenendo fenomeni come la manomissione di dati e l’eventuale intercettazione degli stessi. Il principale obiettivo del protocollo è quindi quello di rendere sicura ogni comunicazione tra le applicazioni che lo implementano, e per permettere ciò lo stesso protocollo deve fornire tre garanzie fondamentali come: l’autenticazione, la confidenzialità e l’affidabilità. Per garantire la prima, SSL utilizza la crittografia a chiave pubblica, descritta in dettaglio nella parte finale di questo capitolo. L’autenticazione garantisce l’identità degli interlocutori, rappresentati in questo caso

dalle applicazioni, attraverso la verifica di file che svolgono la stessa funzione svolta dalla carta d'identità nella vita quotidiana: i certificati. Generalmente è prevista la verifica della sola identità del server, ma se richiesta può essere implementata anche l'autenticazione del client. Per garantire confidenzialità nella comunicazione, SSL/TLS prevede l'utilizzo della crittografia simmetrica, ma solo dopo aver completato una fase chiamata "*handshake*" nella quale avviene la negoziazione della chiave segreta per la cifratura. Rendere affidabile una comunicazione significa garantire l'integrità dei messaggi scambiati. SSL/TLS provvede all'integrità dei dati utilizzando funzioni di hash come SHA, MD5, le quali permettono di capire se i dati sono stati volontariamente modificati durante la loro trasmissione.

SSL/TLS è generalmente utilizzato per rendere sicuri protocolli applicativi quali SMTP per l'invio della posta elettronica e HTTP per il web. HTTP con il supporto SSL/TLS prende in nome di HTTPS, che è molto utilizzato per rendere sicure le pagine web per il commercio elettronico. Tuttavia, grazie alle sue caratteristiche architetturali, SSL/TLS, è il protocollo ideale per blindare qualunque protocollo basato su connessioni TCP.

Come accennato precedentemente, SSL fu sviluppato dalla Netscape Communication Corporation negli Stati Uniti d'America, ma alle origini, fu oggetto di numerose controversie pubbliche. Di fatti il governo americano, pose da subito dei limiti nella diffusione delle versioni sempre più avanzate del protocollo, questo per mantenere una certa supremazia tecnologia nei confronti del resto dei paesi del mondo. Nelle prime implementazioni del protocollo, la chiave simmetrica utilizzata non superava i 40 bit. Essa produceva una cifratura che poteva essere forzata tramite l'utilizzo di tecniche come gli attacchi di forza bruta (*"brute force"*). Lo stesso governo americano era consapevole del fatto che, tramite autorità giudiziarie, istituzioni e servizi segreti, poteva agevolmente condurre vantaggiose indagini sul traffico criptato con chiavi così deboli. Dopo l'ammissione della disponibilità di nuove chiavi di cifratura, e la nascita di numerose cause legali, il governo fu costretto a modificare notevolmente le restrizioni imposte sulle chiavi simmetriche. Si passò quindi alle nuove implementazioni SSL/TLS che utizzano chiavi simmetriche superiori a 128 bit.

3.2 Architettura di SSL/TLS

In questo paragrafo, verrà descritta sommariamente, l'architettura aggiornata all'ultima versione del protocollo SSL/TLS: TLSv1. Come è facile notare in Fig. 11, SSL è composto da uno strato superiore, interfacciato con i protocolli applicativi, rappresentato dai protocolli di *"handshake",* e da uno strato inferiore, interfacciato con il protocollo TCP, rappresentato dal protocollo *"record"*. Proprietà molto importante dell'architettura del protocollo SSL/TLS è la completa compatibilità verso l'alto, ovvero la sua completa indipendenza dai protocolli di applicazione.
I protocolli di *"handshake"* si dividono in *"Alert protocol"*, *"Change Cipher Spec Protocol"* e *"Handshake protocol"*. Lo strato implementato dai protocolli di *"Handshake"* consente l'autenticazione tra server e client, la scelta dell'algoritmo e delle chiavi per la crittografia. Il protocollo Record effettua operazioni di incapusulamento dei dati che provengono dallo strato superiore, ed è in grado di comunicare alla perfezione con il protocollo di trasporto.

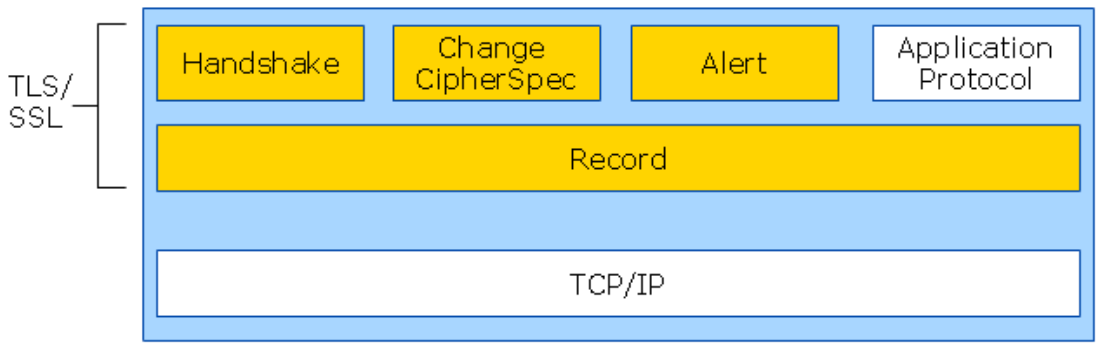


Fig. 11: I due layer del protocollo SSL (in giallo): SSL Record ed SSL Handshake [bib.25]

Il protocollo SSL/TLS è stato sviluppato in modo tale da garantire le proprietà fondamentali che deve avere una connessione sicura realizzata da un protocollo non

proprietario, ovvero: trasparenza, estendibilità, efficienza. Di fatti la sicurezza nel collegamento deve essere corredata dalla possibilità di replicare, estendere ed integrare lo stesso protocollo in sistemi eterogenei. Inoltre è fondamentale che l'architettura e l'implementazione del protocollo fornisca una completa compatibilità verso il basso e verso l'alto. Questo per garantire che in futuro, vuove versioni e nuovi metodi per la crittografia possano essere facilmente integrati nel protocollo e per far si che sopravvivano ragionevolmente nel tempo anche le connessioni realizzate con il protocollo nelle sue pubblicazioni precedenti. Bisogna dire che, poichè SSL impiega notevoli risorse di calcolo, in particolare per le operazioni di crittografia con le chiavi segrete, gli sviluppatori hanno pensato bene di incorporare un meccanismo software di *"session caching"*, che incrementi, nei limiti delle possibilità, l'efficienza del protocollo.

Nel protocollo SSL, connessioni e sessioni sono due concetti ben distinti. Di fatti, una "SSL connection" è una relazione tra un client e un server che appartiene ad una determinata "SSL session". Le sessioni vengono di volta in volta create dall' Handshake protocol e contengono una lista di parametri normalmente condivisi da più connessioni. Ogni qual volta è necessaria la modifica dei parametri di una sessione SSL, il client e d il server, si affidano all'Handshake protocol che ne ricrea una nuova. Non appena viene stabilita una nuova sessione SSL, durante l'Handshake protocol, restano sospesi temporaneamente gli stati di lettura e scrittura, che tornano ad essere attivi alla conclusione della fase di Handshake. Lo stato di una sessione SSL è indicato da parametri quali: Session identifier, Peer certificate, Compression method, Cipher Spec, Master secret, Is resumable. Essi rappresentano nell'ordine, l'identificatore di sessione rappresentato con una determinata sequenza di byte, il certificato del peer nel formato X509 (server o client), l'algoritmo per la compressione dei dati, gli algoritmi utilizzati per produrre il MAC, la chiave di 48 byte condivisa tra i peer ed infine il flag che indica se la stessa sessione può generare nuove connessioni. A sua volta, ogni connessione possiede dei parametri che ne definiscono lo stato. Tra i parametri delle connessioni SSL abbiamo: *server random e client random, server write mac secret e client write*

mac secret, server write key e client write key, initialization vectors, sequence numbers. I *random* altro non sono che delle sequenze di byte, generate casualmente e condivise tra i peer per identificare la connessione. I *write mac secret* sono le chiavi segrete che generano il MAC per l'invio dei dati al peer. Le *write key* rappresentano le chiavi di encryption convenzionali, ovvero le chiavi per criptare i dati da inviare al peer. *Initialization vectors* è il vettore inizializzato dal protocollo di handshake e che verrà utilizzato per salvare un blocco di testo cifrato che proviene dal protocollo record. I *sequence numbers* non sono altro che i numeri sequenziali, incapsulati per ogni connessione, che permettono di ricostruire nel giusto ordine i messaggi ricevuti dal peer.

3.2.1 Protocollo RECORD

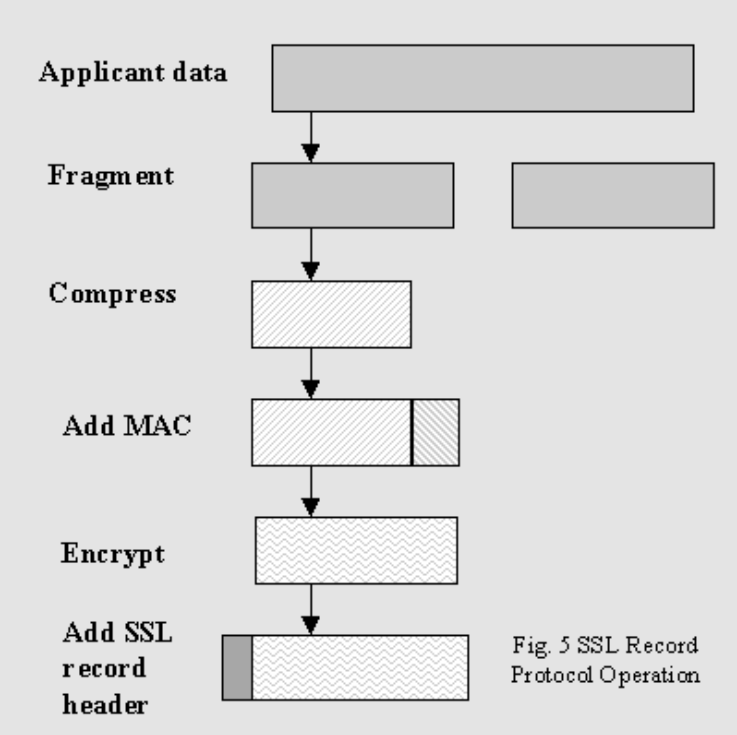


Fig. 12: Operazioni del protocollo RECORD [bib.25]

Lo strato inferiore del protocollo SSL/TLS è a sua volta un protocollo con funzionalità specifiche chiamato *"Record protocol"*. Lo scopo principale del protocollo Record è quello di fornire dei servizi per le singole connessioni SSL. I servizi offerti vanno dalla garanzia di confidenzialità della connessione alla salvaguardia dell'integrità dei dati trasmessi. Nelle operazioni svolte dal protocollo *"Record"* rientrano: la frammentazione dei messaggi provenienti dal livello superiore, la loro compressione, l'applicazione del MAC, la cifratura e l'aggiunta dell'header prima della consegna al protocollo di trasporto. I messaggi provenienti dal protocollo di trasporto, subisco le operazioni complementari in ordine inverso rispetto ai messaggi che viaggiano nel senso opposto. Di fatti, prima di inviare un pacchetto informativo al TCP, il Record protocol decripta il pacchetto, lo verifica, lo decomprime per poi riassemblarlo prima della consegna al livello superiore. Nella fase di frammentazione, ogni messaggio ricevuto dall'Application Level, viene suddiviso in blocchi non più grandi di 2^14 byte, per poi passarlo alla fase di compressione. Dopo esser stato compresso opportunamente, viene assemblato al pacchetto il suo MAC, che è stato generato utilizzando la chiave segreta. Il MAC non è altro che il codice di autenticazione del messaggio (Message Authentication Code), che consentirà al peer di verificare l'integrità dei dati contenuti nel messaggio stesso. L'ultimo *step* prima dell'invio, prevede una cifratura con chiave simmetrica, che può estendere il contenuto del messaggio fino a 1024 byte. Gli algoritmi utilizzati per la cifratura e la lunghezza delle rispettive chiavi sono riportati in figura 13:

Block Cipher		Stream Cipher	
Algoritmo	**Key Size**	**Algoritmo**	**Key Size**
IDEA	128 bit	**RC4-40**	40 bit
RC2-40	40 bit	**RC4-128**	128 bit
DES-40	40 bit	**DES**	56 bit
3DES	168 bit		
Fortezza	80 bit		

Fig. 13: Algoritmi di cifratura del protocollo RECORD [bib.25]

La lunghezza della chiave di un algoritmo, è fondamentale per la sicurezza dei dati cifrati. Normalmente, per quanto possa essere complesso, l'algoritmo è noto a tutti coloro che si occupano di crittografia. Per violare un algoritmo, è sufficiente un attacco di forza bruta, e questo fa capire quanto nella crittografia sia importante la lunghezza della chiave utilizzata. Perciò nei moderni algoritmi, la sicurezza dell'algoritmo di cifratura è totalmente affidata alla lunghezza e alla segretezza delle sue chiavi.
Gli algoritmi comunemente utilizzati nel mondo nel protocollo SSL/TLS sono il DES, l'RC4 e l'RC2, che vengono utilizzati nel protocollo Record per la cifratura dei blocchi.
Solitamente, prima di criptare un blocco contenente il MAC del messaggio, lo stesso viene paddato, cioè vengono aggiunti in coda dei byte che completano il blocco prima di passare alla fase successiva. Per permettere la distinzione dei byte che rappresentano i dati dai byte aggiunti nell'operazione di padding, viene aggiunto in coda un byte che indica la lunghezza esatta del padding. Per l'aggiunta dei padding byte esiste inoltre un metodo ben preciso, esso prevede che la quantità di byte da aggiungere serva esattamente a rendere il blocco un multiplo della lunghezza del blocco elementare voluto dall'algoritmo che avrà in pasto il messaggio. Ammettiamo per esempio che il testo in chiaro occupi una lunghezza di 88 byte e il che il MAC ne occupi 30. Se il messaggio da consegnare all'algoritmo dovrà essere il multiplo di un blocco da 8, ai 118 byte del messaggio verranno aggiunti altri due byte perchè arrivi a una lunghezza

complessiva di 120. L'ultima operazione che il protocollo Record deve effettuare prima dell'invio è quella della preparazione dell'intestazione (header) che dovrà sostanzialmente contenere quattro informazioni: la sigla del protocollo del livello superiore che dovrà essere utilizzato per interpretare il messaggio (lunga 8 bit), il numero che convenzionalmente indica l'ultima versione del protocollo SSL/TLS utilizzata, il numero che convenzionalmente indica la versione più vecchia del SSL utilizzabile e la lunghezza finale in byte che il testo in chiaro dovrà avere. Finora sono state definite le seguenti sigle per indicare il protocollo superiore destinazione del messaggio in uscita dal Record: change_cipher_spec, alert, handshake, e application_data.

3.2.2 Protocolli di HANDSHAKE

3.2.2.1 Change Cipher Spec Protocol

Questi procolli minori, ma non per questo meno importanti, vengono denominati *"specific protocols"* in virtù delle funzioni che assolvono all'interno dei protocolli di Handshake. Sono interfacciati direttamente con il protocollo Record, dal quale ricevono e forniscono messaggi. Il Change Cipher Spec Protocol ha un'unica semplice funzione: aggiornare le cosiddette "cipher suites" che vengono utilizzate per una specifica connessione SSL.

3.2.2.2 Alert Protocol

L'Alert protocol ha come obiettivo il trasporto dei messaggi di Alert. Ovviamente, anche i messaggi di Alert vengono criptati. essi sono formati da due byte il primo dei quali indica il tipo di gravità del messaggio che può essere uno "warning" o un "fatal" che notificano rispettivamente un errore trascurabile e un errore fatale. Quando l'errore è fatale la connessione viene interrotta bruscamente, e questo avviene per esempio quando l'autenticazione del peer fallisce. Le altre connessioni aperte nella medesima sessione non vengono uccise, ma la sessione non potrà in futuro crearne di nuove. Il

secondo byte indica convenzionalmente il codice che sta a identificare un determinato Alert.

3.2.2.3 Handshake protocol

Il protocollo più complesso del livello superiore è l'handshake protocol, esso infatti svolge una serie di funzioni fondamentali per preparare l'instaurazione delle connessioni sicure con SSL/TLS. Esso si divide in quattro fasi consecutive formate dallo scambio di messaggi per l'autenticazione del peer e la negoziazione dell'algoritmo di cifratura. Tutti i messaggi che invia l'handshake protocol hanno un formato fisso: <type, lenght, value>. Andiamo nel dettaglio delle quattro fasi che formano l'handshake protocol.

[Fase 1] Fase informativa delle capacità dei peer

In questa fase, i peer decidono quali caratteristiche saranno le caratteristiche della connessione sicura che intendono instaurare. Il primo messaggio è inviato dal client (ovviamente, poichè sono i client a richiedere la connessione) ed è chiamato *client_hello_message.*

Il *client_hello_message* contiene informazioni riguardanti la versione del protocollo SSL/TLS, gli identificatori di sessione, le suites degli algoritmi cifratura supportati e i metodi di compressione. Contiene inoltre una struttura dati generata casualmente che evita la proliferazione di richeste di connessione che avvengono negli attacchi denial of service. Per identificatore di sessione si intende il parametro *sessionID,* che indica se la richiesta in corso si riferisce a una domanda per l'aggiornamento dei parametri di una connessione già creata oppure se la richiesta si riferisce ad una connessione ex-novo. Una informazione fondamentale è rappresentata dall'elenco delle suites degli algoritmi di cifratura. In effetti il client è tenuto ad informare il server sulle sue preferenze in merito di algoritmi di cifratura elencando le suites in ordine decrescente rispetto alla sua volontà di adoperarle. Infine, nel *client_hello_message* sono indicati tutti i metodi di compressione supportati.

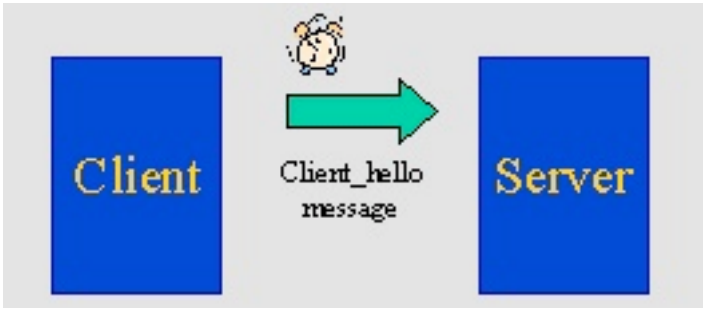


Fig. 14: Client hello message [bib.25]

In risposta al client_hello_message, il server prepara un server_hello_message da inviare al peer, includendo tutte le informazioni necessarie per una conclusione positiva della fase di hello. Il server_hello_message contiene lo stesso tipo di informazioni del client_hello_message, ma specificate tramite specifiche convenzioni. Per esempio, la prima delle informazioni incluse che riguarda la versione SSL/TLS supportata, deve essere strutturata in modo tale da riportare la più vecchia delle versioni supportate dal client e l'ultima versione supportata dal server. Anche il server_hello_message contiene la sua struttura dati random da inviare al client che è completamente diversa da quella inviatagli dallo stesso. Per l'informazione che indica il sessionID il server ha una doppia possibilità: se la richiesta del client si riferiva ad un aggiornamento dei parametri di una connessione, il server riscrive gli stessi parametri ricevuti dal client, altrimenti lui stesso scrive i valore che identifica la nuova sessione. Nel server_hello_message, l'informazione relativa alle *"cipher suites"* non è più rappresentata da un elenco, ma il server stesso sceglie e notifica al client una sola delle suites suggerite dal peer. Tra le informazioni relative alle suites degli algoritmi di cifratura, il server notifica al client quale metodo è stato scelto per le operazioni di compressione e quale metodo è stato scelto per lo scambio delle chiavi. In genere, i metodi per il key_exachange sono scelti tra RSA, Diffie-Helman e le sue varianti e Fortezza. Per ultima viene inclusa l'informazione riguardante il *"cipher spec"*, in altre parole algoritmo per la generazione del MAC, il tipo e l'algoritmo di cifratura e altri parametri che lo caratterizzano.

Fig. 15: Server hello message [bib.25]

Gli schemi seguenti riassumono e descrivono sommariamente la struttura del client_hello_message e del server hello_message.

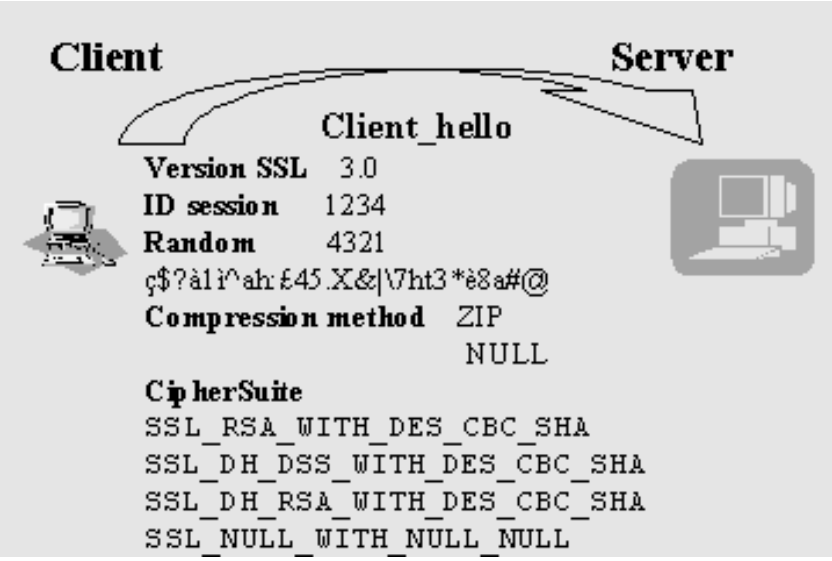


Fig. 16: Esempio di Client_hello_message [bib.25]

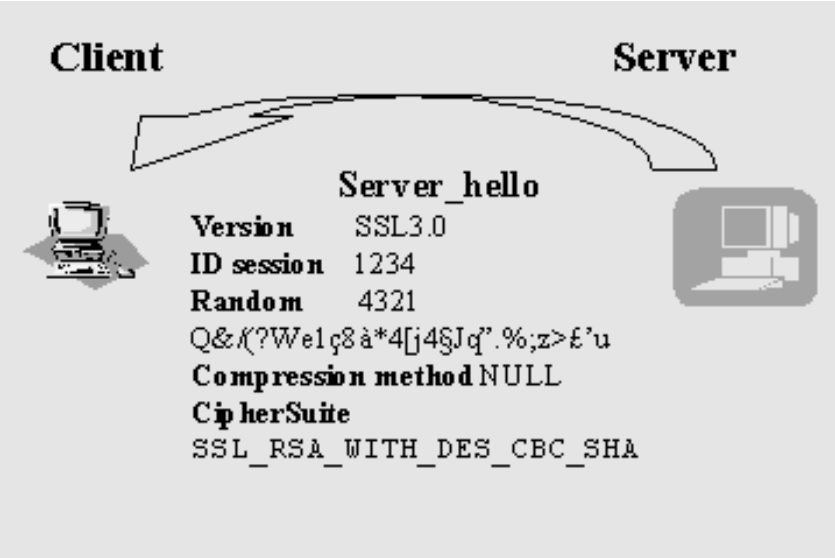


Fig. 17: Esempio di Server_hello_message [bib.25]

[Fase 2] **Autenticazione del server e scambio della chiave**

La prima operazione che il server compie durante la fase di autenticazione è l'invio del suo certificato al client. Questa operazione si ripeterà ogni volta che i due peer dovranno scambiare la chiave. Dopo l'invio del certificato, nel caso in cui non venga usato l'RSA per lo scambio della chiave, il server invia un *server_key_exchange*; in caso contrario il server va al passaggio successivo. A questo punto lo stesso server invia la richiesta di certificato attraverso un *certificate_request*. Il certificate_request definisce l'algoritmo utilizzato per il certificato e un elenco delle authority. Per chiudere la fase di autenticazione e scambio della chiave il server invia un messaggio vuoto chiamato *server_done_message*.

Fig. 18: Fase 2 dell'Handshake [bib.25]

[Fase 3] **Autenticazione del Client e scambio della chiave**

Non appena riceve il server_done_message, il client verifica immediatamente la validità del certificato inviatogli dal server, e si assicura che tutti i parametri per l'instaurazione della connessione inviati dal server siano corretti. A questo punto, se il certificato del server risulta valido e se il server ha richiesto autenticazione anche da parte del suo peer, il client invia con il suo certificato tramite un certificate_message. Nel caso il client non disponga di alcun certificato, lo notifica al server inviandogli un no_certificate_alert. A questo punto il client non può far altro che inviare il client_key_exchange e se necessario inviare anche un certificate_verify.

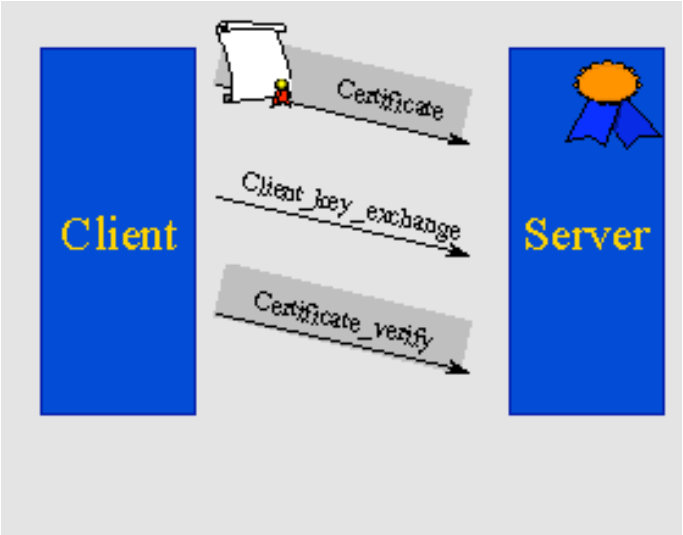


Fig. 19: Fase 3 dell'Handshake [bib.25]

[Fase 4] **Terminazione**

Perchè venga completato l'iter del protocollo di handshake, i due peer si scambiano informazioni tramite il change cipher spec protocol, che sancisce la fase di chiusura. Nel dettaglio il client invia un change_cipher_spec message e aggiorna il CipherSpec.

Successivamente invierà un messaggio di tipo finished che indica la fine dell'iter di autenticazione e notificherà al peer che lo scambio della chiave è andato a buon fine.
L'ultima parola spetta al server che risponde al client con il suo messaggio change_cipher_spec aggiornando la ciphersuite e chiudendo la pratica. Dopo il completamento di questa fase dell'handshake, i due peer possono cominciare lo scambio diretto dei dati ad opera del livello di applicazione.

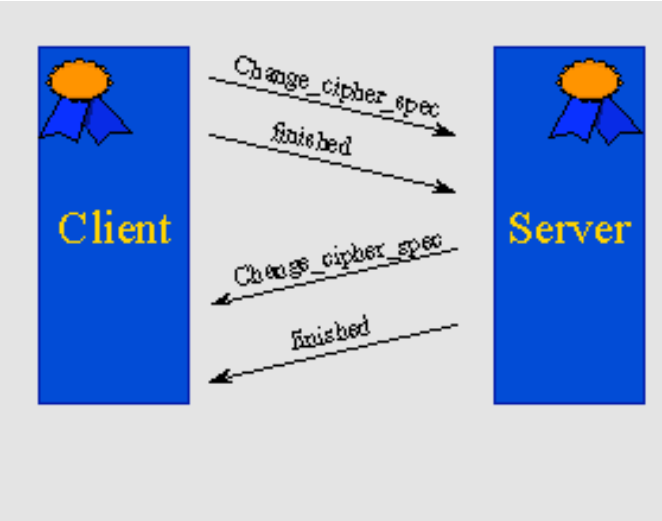


Fig. 20: Fase 4 dell'Handshake [bib.25]

3.3 Sistemi e algoritmi crittografici

Un *sistema crittografico* è in grado di cifrare e decifrare un messaggio attraverso l'uso di un algoritmo (metodo di calcolo) e di una chiave (una stringa segreta alfanumerica). Il messaggio che dovrà essere cifrato viene chiamato testo in chiaro (plaintext) mentre il risultato dell'*algoritmo crittografico* testo cifrato (ciphertext).

Fig. 21: Il sistema crittografico

Un principio fondamentale della crittologia moderna afferma che: *"La sicurezza di un crittosistema non deve dipendere dalla segretezza dell'algoritmo usato, ma solo dalla segretezza della chiave"*. Quasi tutti gli algoritmi crittografici moderni, utilizzati nelle più disparate tecnologie, vengono rilasciati con i codici sorgenti. I sistemi crittografici si dividono in due categorie:

- Sistemi **simmetrici**: utilizzano una sola chiave, sia per cifrare che per decifrare.
- Sistemi **asimmetrici**: utilizzano una coppia di chiavi, una per cifrare e una per decifrare.

3.3.1 Algoritmi a chiave privata (crittografia simmetrica)

La crittografia simmetrica è caratterizzata dall'uso di un'unica chiave per entrambe le operazioni ci cifratura e decifratura. Come accennato dei paragrafi precedenti, la sicurezza di un algoritmo di crittografia simmetrica è affidata completamente alla segretezza e alla lunghezza in termini di byte della chiave che utilizza. Per resistere ai tentativi di violazione della crittanalisi, un algoritmo a chiave privata, deve avere una

chiave il più lunga possibile, poichè solo così si possono eludere eventuali attacchi di forza bruta. In generale, la crittografia simmetrica predeve algoritmi molto più veloci rispetto a quelli utilizzati nella crittografia asimmetrica, ed infatti, gli algoritmi per la crittografia simmetrica vengono adoperati quando il primo requisito per l'operazione di cifratura è la velocità. Nonostante tutto, alcuni algoritmi a chiave simmetrica hanno la possibilità di utilizzare chiavi così lunghe da raggiungere un livello di sicurezza altissimo. Tra i più famosi algoritmi per la cifratura a chiave privata abbiamo il DES, l'IDEA, l'RC5, il Rijndael, padre del famoso AES (Advanced Encryption Standard). Descriveremo di seguito l'algoritmo DES, che viene utilizzato spesso nel protocollo SSL/TLS.

Data Encryption Standard (DES)

Il piu utilizzato algoritmo di cifratura simmetrica a livello modiale è il DES, acronimo di Data Encryption Standard. Come i principali algoritmi di cifratura, nasce negli Stati Uniti negli anni '70, e fu soggetto a restrizioni ferree da parte del governo americano che alle origini pensò bene di tenerne nascosta l'architettura e di pubblicare una versione del DES poco sicura a causa della lunghezza insufficiente della sua chiave. Successivamente, l'algoritmo fu oggetto di studio di numerose università per la ricerca di nuovi tecniche per la crittografia simmetrica. A seconda degli usi e del modo in cui le applicazioni lo adoperano, il DES può essere un algoritmo debole, o altamente sicuro. Ad esempio, l'utilizzo di una chiave simmetrica lunga 56 bit, come avveniva anni fa, non assicurava un buon livello di sicurezza. Basti pensare che i sistemi di calcolo moderni sono in grado, attraverso attacchi di forza bruta, di decifrare il messaggio criptato nell'arco di una giornata. Le successive evoluzioni dell'algoritmo, come il triplo DES lo hanno però portato a livelli di sicurezza inaspettati. Il *"triple DES"* fu il primo passo in avanti in questo senso. Attualmente, molte applicazioni hanno rimpiazzato il DES con l'algoritmo AES, che a detta di molti risolve completamente alcune delle vulnerabilità del Data Encryption Standard.

3.3.2 Algoritmi a chiave pubblica (crittografia asimmetrica)

Nella **crittografia asimmetrica** per criptare i dati scambiati, viene utilizzata una coppia di chiavi per ognuno dei due enti coinvolti. Le chiavi si distinguono in:

- La chiave privata, tenuta segreta, è adoperata per la decodifica;
- La chiave pubblica, che appunto deve essere resa pubblica, è adoperata per crittare;

Il concetto alla base della crittografia asimmetrica è molto semplice da comprendere, basti pensare ad un esempio plasmato sullo scambio di documenti attraverso il classico servizio postale. Immaginiamo che ogni chiave pubblica sia un lucchetto e che la corrispondente chiave privata sia la chiave in grado di aprirlo. Assumiamo come mittente una donna di nome Alice e come destinatario un uomo di nome Bob. Per inviare un documento crittato a Bob, alice deve seguire alcuni semplici passi. Per prima cosa Bob deve mandare ad Alice il suo lucchetto aperto (chiave pubblica), tenendo con se, ben custodita, la chiave che apre il suo lucchetto (chiave privata). La richiesta del lucchetto aperto (chiave pubblica), avviene attivamente da parte di Alice, di fatti Bob, inviando il lucchetto, risponde soltando ad un requisito chiesto da Alice. Ricevuto il lucchetto di Bob, Alice lo utilizza per chiudere la scatola contenente il documento da inviare a Bob e invia la scatola. Ricevuta la scatola, precedentemente chiusa con il suo lucchetto, Bob utilizza la sua chiave per aprirla. Se Bob è l'unico possessore della chiave che apre la scatola, allora solo lui può aprirla. Perchè Bob possa inviare un documento nello stesso modo ad Alice, deve seguire passo passo le stesse operazioni di: richiesta del lucchetto di Alice aperto, chiusura della scatola contenente il documento tramite il lucchetto di Alice ed inviarla. Il punto debole della crittografia simmetrica, è il canale di trasmissione della chiave che come è noto deve restare assolutamente segreta, in questo caso, grazie alla asimmetria di questo metodo non è previsto l'invio di chiavi segrete e questo è il punto di svolta costituito dalla crittografia asimmetrica. Tra i più famosi algoritmi per la cifratura a chiave pubblica abbiamo l'RSA, il Diffie-Hellman e il DSS. Descriveremo di seguito l'algoritmo RSA, che viene utilizzato anch'esso nel

protocollo SSL/TLS.

RSA (Rivest Shamir Adleman)

RSA è l'acronimo costituito dalle iniziali degli sviluppatori dell'algoritmo, che nei primi anni '80 lo brevettarono negli Stati Uniti: Ron Rivest, Adi Shamir, Len Adleman (ricercatori al MIT). Bisogna dire che RSA richiede una quantità notevole di risorse di calcolo e per questo motivo è un algoritmo "lento". Nonostante tutto questo, viene adoperato per codificare le chiavi segrete. Per la comunicazione diretta vengono preferiti algoritmi a crittografia simmetrica come AES, utilizzando le stesse chiavi codificate con l'RSA. Un utilizzo molto frequente dell'RSA è viene riscontrato nelle firme digitali.

La firma digitale si ottiene invertendo il procedimento descritto nel paragrafo precedente per l'invio dei messaggi tra Bob e Alice. Il messagio infatti viene cifrato usando la chiave privata, in questo modo chiunque lo decifri utilizzando la mia chiave pubblica, è sicuro che a decifrarlo può essere stato solo chi possiede la relativa chiave privata. Se il messaggio è troppo esteso, può risultare comodo e più veloce crittare con la chiave privata un hash del messaggio allegandola allo stesso a mò di firma. Ciò però può non essere un metodo sicuro al 100% poichè dipende anche dalla sicurezza che garantisce il condice hash.

RSA è il più diffuso algoritmo per la crittografia asimmetrica ed è basato su un concetto matematico complesso come la fattorizzazione dei numeri primi. Il funzionamendo dell'algoritmo non è semplice da comprendere, tuttavia proviamo a descriverlo brevemente per dare l'idea di quale sforzo computazionale debba compiere un calcolatore per utilizzarlo.

Per prima cosa vengono scelti due numeri molto grandi e diversi tra loro, che noi chiameremo P e Q. Successivamente viene calcolato il loro prodotto semplice che chiameremo modulo (N). Un terzo numero, "E", molto più piccolo degli altri due, sarà

compresso con l'espressione (P-1) (Q-1). "E" verrà chiamato esponente pubblico e verrà utilizzato nel calcolo del numero D nel seguente modo: E*D=1 mod ((P-1)((Q-1)).

Il numero D sarà chiamato esponente privato e servirà alla generazione delle nostre chiavi. I due insiemi (N,E) e (N,D), saranno rispettivamente la nostra chiave pubblica e la nostra chiave privata. I numeri Q e P andrebbero eliminati, ma in alternativa possono essere salvati dentro la nostra chiave privata. La caratteristica inviolabile dell'algoritmo è che non è possibile risalire ai numeri D ed E senza conoscere i numeri P Q dai quali è stato ricavato. La fattorizzazione dei numeri primi è un calcolo tanto impegnativo quanto più lungo è il numero N utilizzato. Passiamo adesso al processo di cifratura. Il messaggio, che chiameremo M, è criptato tramite l'operazione aritmetica M exp(E (mod N)), e un altro messaggio che chiameremo C viene decriptato tramite C exp(D) che deve essere uguale a M exp(ED) = M exp(1) = mod(N). Tutto il processo termina con successo solo se il numero E, usato per criptare, e il numero D, usato per decriptare dipendono dalla relazione E*D=1 (mod N). Il risultato è che un messaggio cifrato con una delle due chiavi, può essere decifrato solo adoperando la sua chiave gemella. Alcuni studiosi sostengono che l'algoritmo sia debole poichè è basato su un'assunzione non dimostrabile. Di fatti, la radice Nesima di C (modN), se N è un numero dai fattori sconosciuti è computazionalmente non calcolabile.

3.4 I Certificati digitali

Il ruolo svolto da un certificato digitale è del tutto simile a quello svolto nella vita quotidiana dei cittadini dai documenti d'identità. Un documento d'identità attesta, o meglio certifica, l'identità di un cittadino nello stesso modo in cui il certificato attesta l'identita di un ente, un programma, o un utente. Per avere una validità riconosciuta, il documento d'identità deve essere rilasciato da un'autorità della quale tutti possano fidarsi. L'autorità che rilascia i documenti d'identità è il comune di appartenenza, mentre l'equivalente per i certificati digitali è la Certification Authority (CA). I certificati digitali quindi certificano l'appartenenza delle chiavi pubbliche, unico strumento per l'identificazione d'appartenenza delle informazioni digitali. Un certificato digitale contiene per convenzione un elenco di informazioni relative all'entità o alla persona che certifica. Tra le informazioni contenute abbiamo: la chiave pubblica, il nome della persona o dell'entità che corrisponde alla chiave, la sigla dell'algoritmo che ha generato la chiave, il nome della Certification Authority che lo ha rilasciato inclusa la sua firma digitale, il numero seriale del certificato e la data di scadenza dello stesso.

I certificati digitali, possono essere generati in diversi formati, tuttavia la maggior parte di essi segue lo standard X.509. Le Certification Authority, su richiesta esplicita, hanno la possibilità di stampare in modo "indelebile" la data di creazione sui documenti importanti (come i brevetti), sfruttando la riconosciuta identità della firma digitale.

Capitolo 4

TECNOLOGIE UTILIZZATE PER L'IMPLEMENTAZIONE

In questo capitolo saranno illustrate le tecnologie per l'implementazione del progetto generale IENA e alla sua prima estensione riguardante l'instaurazione di una connessione sicura tra client IENA e server IENA attraverso il protocollo SSL/TLS.

4.1 Albero delle scelte

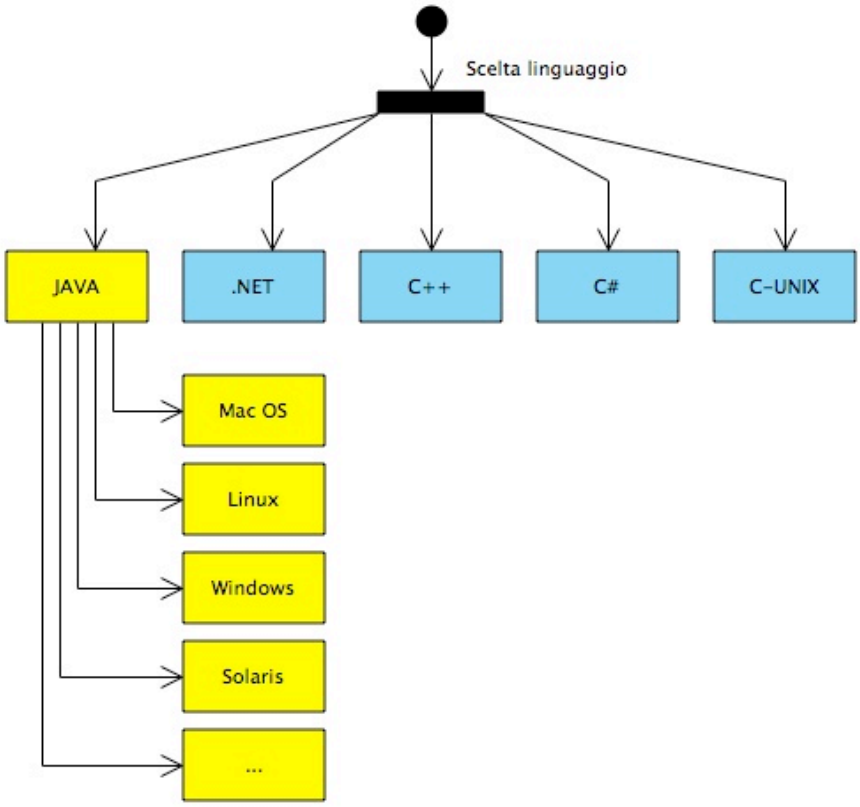


Fig. 22: Scelta del linguaggio

Il linguaggio Java è stato scelto per le seguenti caratteristiche, vantaggiose per la realizzazione del progetto IENA:

1. **Indipendenza dalla piattaforma.** L'esecuzione di programmi Java ha di fatti lo stesso tipo di comportamento su hardware e sistemi operativi differenti. Grazie alla portabilità possiamo potenzialmente scrivere il programma una sola volta per poi lanciarlo su qualunque piattaforma. Tutto ciò grazie alla compilazione del codice in un linguaggio *bytecode*, molto vicino a quello che può essere il linguaggio macchina. Il bytecode viene interpretato dalla *virtual machine*, e ciò rende Java un linguaggio interpretato. Nonostante la sua completa portabilità, il linguaggio è totalmente vincolato alla presenza e al corretto funzionamento della virtual machine. Significa che per ogni piattaforma esiste una implementazione della virtual machine diversa con bug diversi per ogni piattaforma.Da ciò è stata diffusa una simpatica parodia dello slogan di Java "write once, run everywhere", che è diventato "Scrivi una volta, fai il debug ovunque".
2. **E' un linguaggio di programmazione *open source*.** Dal 13/11/06 la Sun (azienda sviluppatrice del linguaggio) ha rilasciato Java sotto licenza GPL;
3. **Include librerie per il networking.** Include oltre al supporto per maggior parte dei protocolli della famiglia IP (Socket), anche il supporto per i programmi con multithreading, necessario per molte applicazioni che usano la rete. Inoltre, il framework JSSE semplifica notevolmente l'implementazione di connessioni sicure.
4. **E' un linguaggio ad oggetti.** Il concetto fondamentale dei linguaggi ad oggetti consiste nel rappresentare il software con l'uso di entità definite astratte o reali che sono chiamate oggetti. Gli oggetti, hanno delle particolari proprietà descritte da variabili e da metodi. Una programmazione strutturata in questo modo semplifica notevolmente la gestione di progetti enormi che ne guadagnano soprattutto in qualità e riusabilità del codice.

4.2 Java Secure Socket Extension (JSSE)

La versione 1.4 della *Java 2 Standard Edition* presenta notevoli modifiche alle caratteristiche della sicurezza del linguaggio Java. Con il rilascio della 1.4 j2sd tutto il framework per la sicurezza in Java e tutte le estensioni di sicurezza che nella versioni precedenti non erano presenti di default sono integrate in un sola piattaforma.
Sono state così raggruppate le librerie della Java Cryptography Extension, della Java Secure Socket Extension e della Java Authentication and Authorization Service.
La JSSE, con l'implementazione del protocollo SSL/TLS fino alla versione TLSv1, offre sia metodi per l'autenticazione che per la protezione dei dati.
A differenza della JCE, la JSSE opera direttamente a livello di rete consentendo l'applicazione diretta degli algoritmi crittografici. Le *"Application Program Interface"* della Java Secure Socket Extension sono incluse nelle librerie ***javax.net*** e ***javax.net.ssl***.
Ad occuparsi delle funzioni di crittografia esiste il provider "SunJSSE".
Le figure che seguono rappresentano rispettivamente l'architettura e l'handshake dal punto di vista della JSSE:

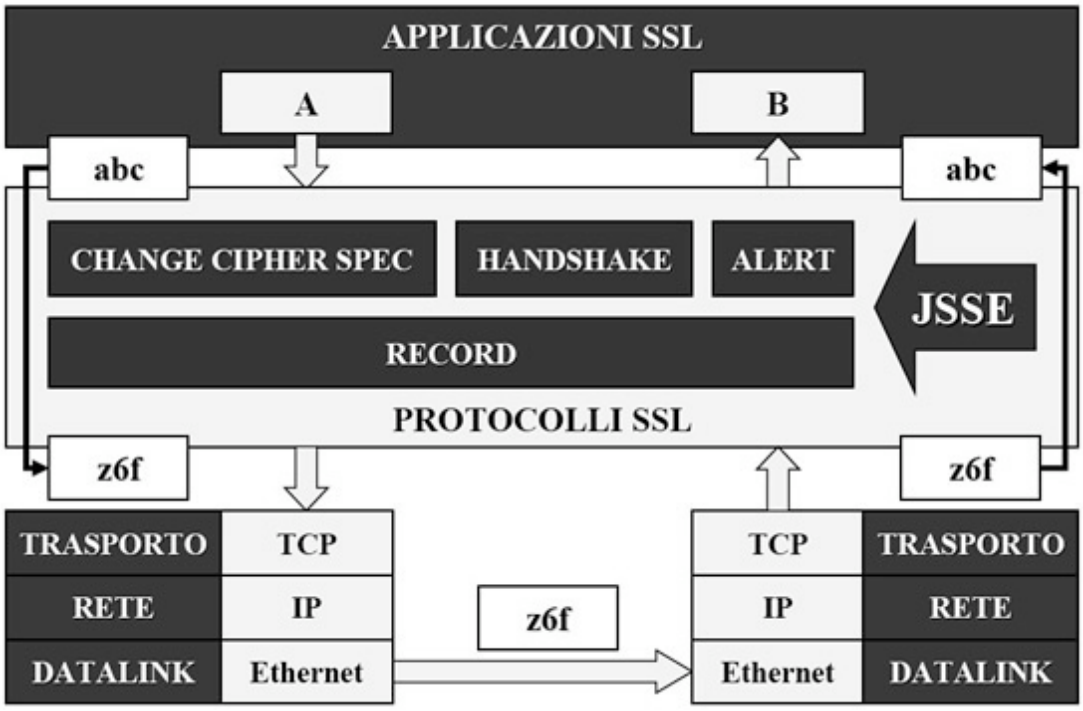


Fig. 23: Architettura SSL/JSSE [bib.26]

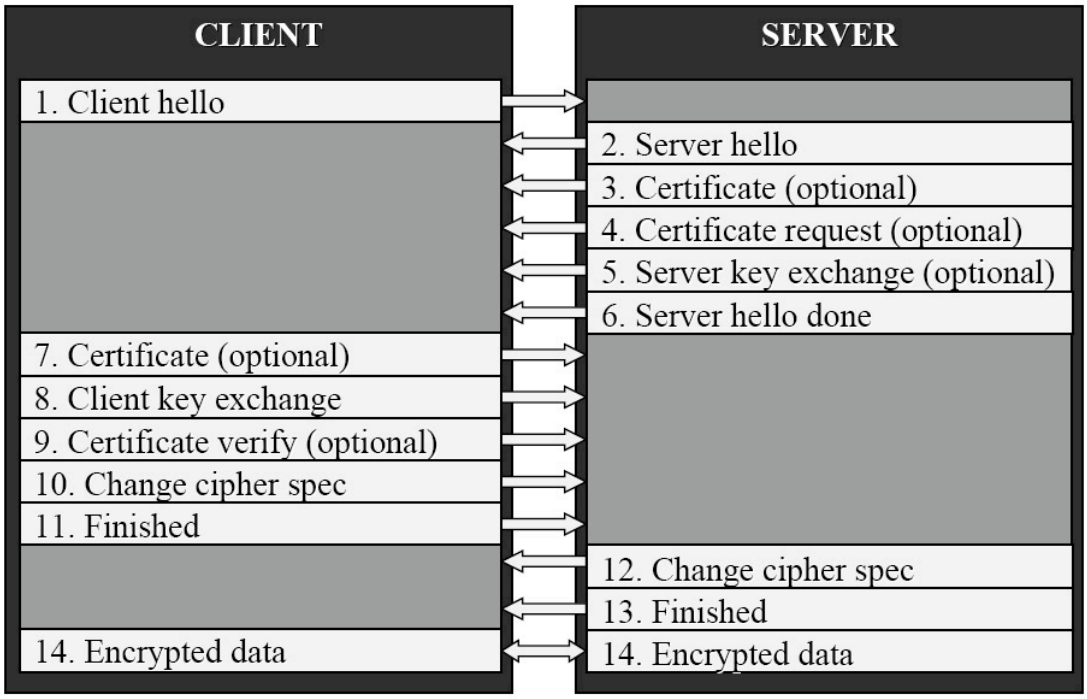


Fig. 24: Handshake SSL/JSSE [bib.]

Fanno parte del framework JSSE le classi per la crezione dei socketSSL, le classi per la creazione del contesto SSL, le interfacce che gestiscono le chiavi e i file in cui sono contenute, e una classe per la gestione delle sessioni SSL.
Di seguito lo schema delle relazioni tra le classi del framework:

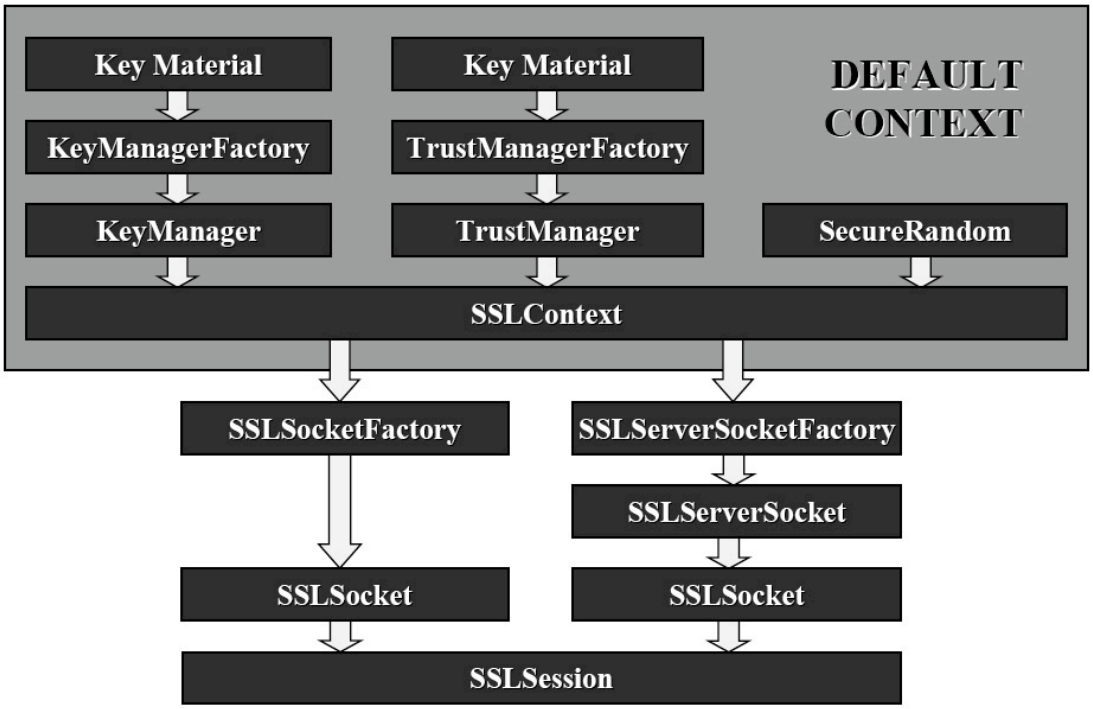


Fig. 25: Classi del Framework JSSE [bib.26]

4.3 Java Keytool - Key and Certificate Management Tool

I tool di sicurezza disponibili con il framework Java consentono di effettuare operazioni riguardanti certificati e firme digitali con estrema semplicità. Con il kit della Java2 SDK vengono forniti strumenti quali *Keytool*, *Jarsigner e Policy tool.* Lo strumento che è stato utilizzato per l'implementazione di IENASSL è keytool. Questo tool Java viene lanciato direttamente dalla linea di comando e consente la gestione di chiavi e certificati tramite il *"Key and Certificate Management tool"*. Tra le operazioni consentite abbiamo la generazione di chiavi pubbliche e private, la richiesta e la distribuzione della

propria chiave pubblica, l'importazione diretta dei certificati nei keystore e la gestione stessa dei trustStore e dei keyStore. Questi due file non sono altro che dei mini database costruiti appositamente per contenere sia i certificati che le chiavi loro associate. Come in ogni database che si rispetti, i dati sono memorizzati ordinatamente secondo delle entries. Abbiamo due tipi differenti di entries: quelle per le proprie chiavi private e i propri certificati e quelle per le chiavi pubbliche e i certificati di altre entità o persone. Le chiavi private, come è noto, vengono memorizzate in modo sicuro in modo tale che non siano accessibili ad entità non autorizzate e sono corredate dei loro certificati. I certificati che contengono le chiavi distribuite da altre entità sono memorizzati nelle entries chiamate *"trusted certificate"*. Per distinguere una entries da un'altra, keytool utilizza un apposito sistema di alias definiti dall'utente al momento della generazione della chiave. Per la generazione di una coppia di chiavi si usa l'opzione –genkey, mentre per l'importazione di uno o più certificati viene usata l'opzione –import. Ogni volta che viene usato il comando keytool con queste due opzioni, viene creato un nuovo keystore. Ogni keystore ha il compito di preservare le chiavi contenute al suo interno mediante l'utilizzo di password definite dall'utente al momento della loro generazione. Altre importanti opzioni di keytool sono: –export, che esporta il certificato indicato dall'alias in un file esterno per rendere possibile la validazione del certificato in keyStore appartenenti ad altre entità; -list che stampa a video tutto il contenuto del keyStore; -storepasswd che permette di modificare la password che protegge il keyStore; -keypasswd che consente di modificare la password della singola chiave identificata tramite il suo alias; -delete che consente di svuotare l'intero contenuto del keyStore. Il comando keytool si presenta nel classico formato: keytool <opzione1> <opzione2> <...>. Ogni qualvolta generiamo una coppia di chiavi con keytool, di default viene utilizzato come algoritmo il DSA. Se invece stiamo firmando un documento, l'algoritmo scelto da keytool dipende direttamente da quello che è stato utilizzato per generare la chiave con cui viene effettuata la firma digitale. Per esempio, se possediamo una chiave generata con l'algoritmo DSA, le firme digitali che effettueremo con quella chiave utilizzeranno la suite "SHA1withDSA". Al contrario, se la nostra chiave è stata generata con l'algoritmo RSA, allora firmeremo i nostri

documenti con la suite “MD5withRSA”. Per essere il più vicino possibile agli standard, keytool adotta come formato per i certificati l’X.509. In alternativa consente l’utilizzo di formati come il Public Key Criptography Standard numero 12. Di fatti, la maggior parte delle operazioni utili che si possono effettuare tramite keytool hanno come oggetto certificati appartenenti al formato PKSC#12 oppure all’X.509. L’esportazione dei certificati può avvenire in diversi formati come ad esempio OpenPGP, PKCS#12 e Base64. Bisogna precisare che per l’importazione dei certificati, il comando –import può effettuare sia l’importazione del certificato nella lista di quelli validati (trusted) che l’importazione di un certificato ricevuto da una Certification Authority. Il base all’alias usato, il processo d’importazione segue strade differenti. Infatti se l’alias indicato è una delle key entries, il sistema capisce che si tratta di una replica e agisce di conseguenza controllando che i due certificati abbiano la stessa chiave pubblica prima di passare alla vera e propria importazione. Se invece l’alias non esiste ancora come key entry, keytool interpreta l’operazione come l’importo di un certificato validato.

Capitolo 5

IMPLEMENTAZIONE DI IENA

Fig. 26: Logo IENA

In questo capitolo è descritta una delle possibili implementazioni del progetto IENA alla sua prima estensione riguardante la costruzione di una comunicazione sicura tra il server e il client.

5.1 Implementazione del server IENA

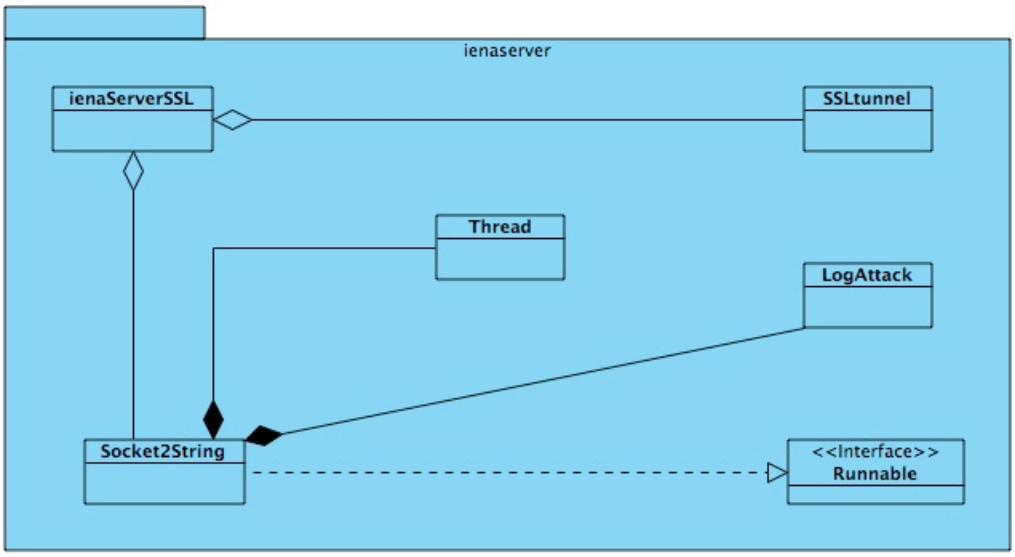


Fig. 27: Class diagram del serverIENA

Viene riportato di seguito il codice delle singole classi ***ienaServer, SSLtunnel, Socket2String, LogAttack***.

5.1.2 ienaServer

```
/*******************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.       **
** Iena is protected from copyrigth© 2005-2050 International licence.          **
**                                                                             **
**  IENA DEVELOPERS:                                                           **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                             **
**                                  marco.ramilli@studio.unibo.it              **
**                                  eth0up@rrsecurity.info                     **
**                                                                             **
**      IVANO MANCA                 ivano.manca@gmail.com                      **
**                                  ivano.manca@studio.unibo.it                **
*******************************************************************************/
package ienaserver;

import ienaserver.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;
import javax.net.ssl.*;

/*
 * IENA SSLSERVER PROJECT
 */
public class ienaServer{

       public static void main(String[] s) {

                String pw = "";                              // pw keystore
                SSLtunnel tunnel;
                Socket connectedSocket = null;               // Socket ritorno accept
               int serverPort = 20000;                      // porta socket
               //String logFile;                            // stringa path fileLog
               Socket2String s2s;                           // lettura dal socket

                if (s.length == 1){
                    pw = s[0];
```

```
                    System.out.println("ienaServer running with default params:");
                    System.out.println("ienaServerPort.................."+serverPort);
                    //System.out.println("ienaServerLogFilePath.......ienaLog.txt");
                }else if(s.length == 2){
                        pw = s[0];                                  // pw keystore
                      serverPort = Integer.parseInt(s[1]);        // Porta server
                        //ienasrvks = s[1];                         // pw keystore
                        //logFile = s[1];                           // file di log
               }else {
                    System.out.println("Wrong input parameters, use:");
                    System.out.println("java -jar ienaServer.jar <ienaKeystorePassword>
<ienaServerPort>");
                    System.out.println("Note: To run with default params, " +
                            "type: java -jar ienaServer.jar <ienaKeystorePassword>");
                    System.exit(-1);
               }
                // Creazione contesto tramite tunnel
                tunnel = new SSLtunnel(pw, pw);
                try {
                    // creazione del socket sulla serverPort
                    tunnel.createSSLSocket(serverPort);
                    System.out.println("ienaServer running on port: "+serverPort);
                } catch (IOException e) {
                    System.out.println("ienaServer: SSL socket error");
                }//catch

                // configurazione algoritmi di cifratua
                tunnel.cipherSuitesOn();

                // attivazione richiesta autenticazione del client
                tunnel.clientAuthOn();
               try{
                    //riuso di ip del socket
                    tunnel.socketReuseAddressOn();
               }catch(SocketException e){
                       System.out.println("ineaServer: socket option set error\n");
               }
               while(true) {
                       try{    // accetta la richiesta di connessione del client
                              connectedSocket = tunnel.socketAccept();
                                 // recupero della stringa ricevuta dal client iena
                                 // completamento delle operazioni di log
                                 s2s = new
Socket2String(connectedSocket.getInputStream());
```

```
                    }catch(IOException e){
                            System.out.println("ienaServer: accept error");
                    }
                }
        }//main
}//class
```

5.1.3 SSLtunnel

```
/*******************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.        **
** Iena is protected from copyrigth© 2005-2050 International licence.           **
**                                                                              **
**  IENA DEVELOPERS:                                                            **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                              **
**                                  marco.ramilli@studio.unibo.it               **
**                                  eth0up@rrsecurity.info                      **
**                                                                              **
**      IVANO MANCA                 ivano.manca@gmail.com                       **
**                                  ivano.manca@studio.unibo.it                 **
*******************************************************************************/
package ienaserver;

import ienaserver.*;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.util.*;
import javax.net.ssl.*;
/**
 *
 * @author ivanomanca
 */
public class SSLtunnel {
    SSLServerSocket serverSocket;   // socket SSL
    int serverPort;                 // porta di ascolto del socket
    KeyManagerFactory kmf;          // key manager
    KeyStore ks;                    // key store
```

```
TrustManagerFactory tmf;        // trust manager
SSLContext ctx;                 // contesto SSL
String pwK;                     // pw keystore
String pwT;                     // pw truststore
/**
 * Costruttore SSLtunnel per il server
 * in ingresso le pw degli stores
 */
public SSLtunnel(String serverKeyStorePw, String serverTrustStorePw) {
    this.pwK = serverKeyStorePw;
    this.pwT = serverTrustStorePw;
    this.kmf = null; // key manager
    try {
        kmf = KeyManagerFactory.getInstance("SunX509");
    }
    catch(NoSuchAlgorithmException e) {
        System.out.println("ienaServer: SunX509 not supported");
    }

    this.ks = null; // key store
    try {

        ks = KeyStore.getInstance("JKS");
        //recupero password keystore del server
        char[] passwd = pwK.toCharArray();
        ks.load(new FileInputStream("ienasrvkeystore"), passwd);
    }
    catch (NoSuchAlgorithmException ex) {
        ex.printStackTrace();
    }
    catch (CertificateException ex) {
        System.out.println("ienaServer: certs file not valid");
    }
    catch(KeyStoreException e) {
        System.out.println("ienaServer: SunX509 not supported");
    }
    catch(FileNotFoundException e) {
        System.out.println("ienaServer: Certs file not found");
    }
    catch(IOException e) {
        System.out.println("ienaServer: certs file or keystore password not valid");
    }

    char[] kpasswd = pwK.toCharArray();
    try {
```

```
            kmf.init(ks, kpasswd);
        } catch (NoSuchAlgorithmException ex) {
            ex.printStackTrace();
        } catch (KeyStoreException ex) {
            ex.printStackTrace();
        } catch (UnrecoverableKeyException ex) {
            ex.printStackTrace();
        }

        // Dichiarazione della trust manager factory
        this.tmf = null;
        try {
            //inizializzazione e lancio
            tmf = TrustManagerFactory.getInstance("SunX509");
            tmf.init(ks);
        }catch (KeyStoreException ex) {
            ex.printStackTrace();
        }
        catch(NoSuchAlgorithmException e) {
        System.out.println("ienaServer: SunX509 not supported");
        }

        //inizializzazione del contesto ssl
        this.ctx = null; //contesto ssl
        try {
            ctx = SSLContext.getInstance("SSL");
        }
        catch(NoSuchAlgorithmException e) {
        System.out.println("ienaServer: SSL context setting failure");
        }
        try {
            //lancio del contesto
            ctx.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        } catch (KeyManagementException ex) {
            System.out.println("ienaServer: SSL context init failure");
        }
    }

    /*
     *  Crea il nuovo socket SSL
     *  in ingresso la porta di ascolto
     */
    public void createSSLSocket(int serverPort) throws IOException{
        this.serverPort = serverPort;
        SSLServerSocketFactory sslsocketfactory = ctx.getServerSocketFactory();
```

```
        serverSocket = (SSLServerSocket)
sslsocketfactory.createServerSocket(serverPort);
    }
    /*
     * Configura le suites per la cifratura
     *
     */
    public void cipherSuitesOn(){
        // recupero degli algoritmi di cifratura supportati
        String[] scs = serverSocket.getSupportedCipherSuites();
        // stampa degli algoritmi di cifratura supportati
        //for (int i=0; i<scs.length; i++) System.out.println(scs[i]);

        /*  ATTIVAZIONE ALGORITMI DI CRITTOGRAFIA */
        String[] ascs = new String[scs.length];
        int ascsl = 0;

        for (int i=0; i<scs.length; i++) {
        if (scs[i].indexOf("_anon_") > 0)
        ascs[ascsl++] = scs[i];
        }
        String[] oecs = serverSocket.getEnabledCipherSuites();
        //stampa algoritmi oecs
        //System.out.println("++++++++++++++ OECS +++++++++++++++++");
        //for (int i=0; i<oecs.length; i++) System.out.println(oecs[i]);

        String[] necs = new String[oecs.length + ascsl];
        System.arraycopy(oecs, 0, necs, 0, oecs.length);
        System.arraycopy(ascs, 0, necs, oecs.length, ascsl);

        //Scelta della cipher suite
        //String[] csuite = {"SSL_RSA_WITH_RC4_128_MD5"};
        serverSocket.setEnabledCipherSuites(necs);
        //serverSocket.setEnabledCipherSuites(csuite);
    }
    /*
     * Richiede l'autenticazione del client
     *
     */
    public void clientAuthOn(){
        serverSocket.setNeedClientAuth(true);
    }
    /*
     * Setta il riuso dell'ip
     *
```

```
     */
    public void socketReuseAddressOn() throws SocketException{
        serverSocket.setReuseAddress(true);
    }
    /*
     * Esegue l'accept del serverSocket
     *
     *
     */
    public Socket socketAccept() throws IOException{
        return serverSocket.accept();
    }
    /*
     * Ritorna il socket
     *
     */
    public SSLServerSocket getSocket(){
        return serverSocket;
    }
}
```

5.1.4 Socket2String

```
/*******************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.        **
** Iena is protected from copyrigth© 2005-2050 International licence.           **
**                                                                             **
**  IENA DEVELOPERS:                                                           **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                             **
**                                  marco.ramilli@studio.unibo.it              **
**                                  eth0up@rrsecurity.info                     **
**                                                                             **
**      IVANO MANCA                 ivano.manca@gmail.com                      **
**                                  ivano.manca@studio.unibo.it                **
*******************************************************************************/
package ienaserver;

import java.io.*;
import java.net.*;
import java.lang.*;
import ienaserver.*;
/**
 *
 * @author ivanomanca
```

```
 */
public class Socket2String  implements Runnable {

        protected InputStream in;            // inputstream
        protected byte[] rec = new byte[1000];       // array inputstream
        protected int len;                       // lunghezza array inputstream
        String outString;                        // stringa di ritorno
         LogAttack attack;                        // classe per il log
         String logFile = "ienaLog.log";         // path del file per il log

        /**
          * Costruttore Socket2String
          * in ingresso le pw degli stores
          */
        public Socket2String (InputStream i){
                in=i;
                Thread TH1 = new Thread(this);
                 TH1.start();
         }//costruttore

        public void run() {
                try {
                        /* ciclo di lettura dal socket,
                          salvataggio su oggetto String e log su file*/
                          while ((len = in.read(rec)) >= 0) {
                               //System.out.println("inizio lettura da socket");
                               outString = new String(rec, 0, len);
                               attack = new LogAttack(outString);
                               attack.log(logFile);
                        }//while
                        //System.exit(1);
                }//try
                catch (Exception e) {
                     System.out.println("Warning: connection failed, unknown client
refused.");
                     //System.err.println("Socket2StringException " + e.getMessage());
                //System.exit(1);
                 }//catch
        }//run
}//Socket2String
```

5.1.5 LogAttack

```
/*****************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.        **
** Iena is protected from copyrigth© 2005-2050 International licence.           **
**                                                                              **
**  IENA DEVELOPERS:                                                            **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                              **
**                                  marco.ramilli@studio.unibo.it               **
**                                  eth0up@rrsecurity.info                      **
**                                                                              **
**      IVANO MANCA                 ivano.manca@gmail.com                       **
**                                  ivano.manca@studio.unibo.it                 **
*****************************************************************************/
package ienaserver;

import java.util.*;
import java.io.*;
import java.text.SimpleDateFormat;

/**
 *
 * @author ivanomanca
 */
public class LogAttack {

    String form = "EEE, d MMM yyyy HH:mm:ss";
    SimpleDateFormat formato = new SimpleDateFormat(form);
    String time;
    String flag;

    /** Creates a new instance of logAttack */
    public LogAttack(String flag) {
      this.flag=flag;
      GregorianCalendar calendario = new GregorianCalendar();
      Date dateDaCalendar = calendario.getTime();
      this.time = formato.format(dateDaCalendar);
    }

    public void log(String file) throws IOException, FileNotFoundException{
        PrintWriter pw = null;

        try{
```

```
        FileWriter fw = new FileWriter(file, true);
        BufferedWriter fb = new BufferedWriter(fw);
        pw = new PrintWriter(fb);
        pw.println("** WARNING! ** Date: "+time+" "+flag+"\n");
        pw.close();
        }
        catch(FileNotFoundException e){
            System.out.println("ienaServer:"+file+" file not found");
        }
        catch(IOException e){
            System.out.println("ienaServer: IOexception "+e);
        }
        finally{
            if(pw!=null){pw.close();}
        }

    }

}
```

5.2 Implementazione del client IENA

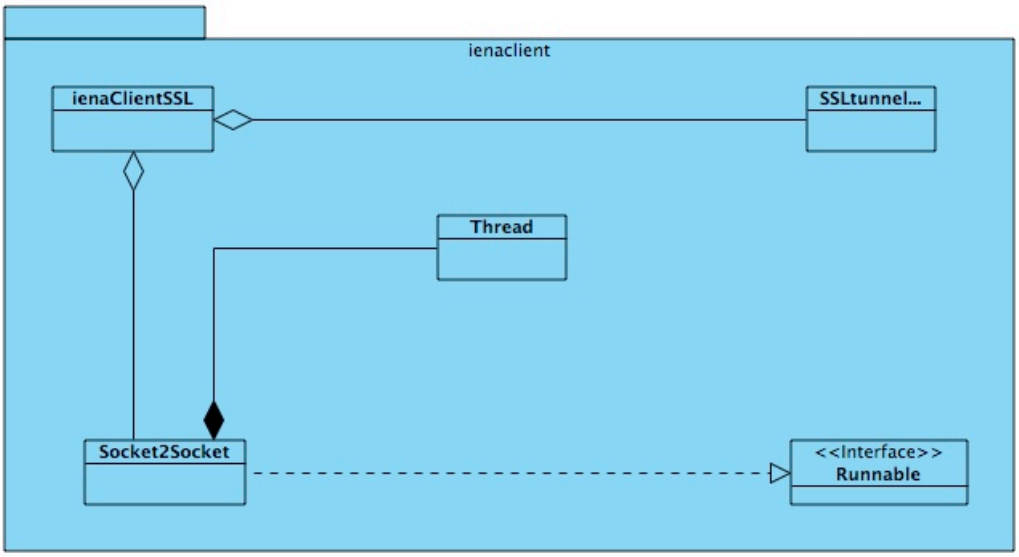


Fig. 28: Class diagram del clientIENA

Viene riportato di seguito il codice delle singole classi ***ienaClient, SSLtunnelClt, Socket2Socket***.

5.2.1 ienaClient

```
/*****************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.     **
** Iena is protected from copyrigth© 2005-2050 International licence.        **
**                                                                           **
**  IENA DEVELOPERS:                                                         **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                           **
**                                  marco.ramilli@studio.unibo.it            **
**                                  eth0up@rrsecurity.info                   **
**                                                                           **
**      IVANO MANCA                 ivano.manca@gmail.com                    **
**                                  ivano.manca@studio.unibo.it              **
*****************************************************************************/
package ienaclient;

import java.io.*;
import java.net.*;
import javax.net.ssl.*;
```

```
import ienaclient.Socket2Socket;

public class ienaClient{

        public static void main(String[] s){

                String pwK = "";                        //Password del keystore
                 String pwT = "";                        //Password del truststore
                 SSLtunnelClt tunnel;                    //Tunnel SSL
                ServerSocket ienaSocket = null;         //Socket per l'ascolto della iena
                Socket attackedSocket = null;           //Socket verso l'attacker
                int ienaPort = 80;                      //Porta default iena
                int ienaServerPort = 20000;             //Porta default ienaServer
                String ienaServerIP = "127.0.0.1";      //Ip di default del server iena


                 //controllo input
                 if(s.length == 2){
                     pwK = s[0];
                     pwT = s[1];
                     System.out.println("ienaClient running with default params:");
                     System.out.println("ienaPort.......................... 80");
                     System.out.println("ienaServerIP..................127.0.0.1");
                     System.out.println("ienaServerPort...................20000");
                 }
                 else if(s.length == 5){
                     pwK = s[0];
                     pwT = s[1];
                     ienaPort = Integer.parseInt(s[2]);          // Porta iena
                     ienaServerIP = s[3];                        // IP ienaServer
                     ienaServerPort = Integer.parseInt(s[4]);    // Porta ienaServer

                 }else {
                     System.out.println("Wrong input parameters, use:");
                     System.out.println("java -jar ienaClient.jar
<ienaClientKeystorePassword> " +
                             "<ienaClientTruststorePassword> <ienaPort> <ienaServerIP>
<ienaServerPort>");
                     System.out.println("Note: To run with default params, " +
                             "type: java -jar ienaClient.jar <ienaClientKeystorePassword>
" +
                             "<ienaClientTruststorePassword>\n");
                     System.exit(-1);
                 }
                 // Creazione contesto tramite tunnel
```

```
                tunnel = new SSLtunnelClt(pwK, pwT);
                // Apertura socket in attesa di connessione dall'attaccante
                try{
                        ienaSocket = new ServerSocket(ienaPort);
                }catch(IOException e) {
                        System.out.println("Iena: socket error on port "+ienaPort);
                }

                try{
                //sockopt in c riuso di ip
                        ienaSocket.setReuseAddress(true);
                          System.out.println("Iena is on port "+ienaPort+" waiting for an
attacker...");

                }catch(SocketException e){
                        System.out.println("Iena: socket options settings error");
                }
                while(true) {

                        // Accetta il tentativo di connessione dell'attaccante
                        try{
                                attackedSocket = ienaSocket.accept();
                                  System.out.println("---------------------------------");
                                  System.out.println("| Step(1) Attack detected!     |");
                        }catch(IOException e){
                                System.out.println("Iena: socket accept error");
                        }

                          /* TENTA UNA CONNESSIONE SSL CON IENASERVER*/
                          try {
                             tunnel.createSSLSocket(ienaServerIP, ienaServerPort);
                          } catch (IOException e) {
                              System.out.println("Iena: ienaServer SSL socket error");
                          }//catch

                          tunnel.cipherSuitesOn();

                          //threads per gestire la comunicazione tra client e superserver
                          System.out.println("| Step(2) Calling ienaServer... |");
                          new Socket2Socket (attackedSocket, tunnel.getSocket(), ienaPort,
true); // TH2 --> gestisce le richieste del client e le prolunga al superserver
                        System.out.println("| Step(3) Notified.             |");
                          System.out.println("---------------------------------");
```

```
                        //new Socket2Socket
(clientSocket,myNetdSocket,serverPort,false); // TH3 --> gestisce le risposte del
superserver e le prolunga al client
                }
        }
}
```

5.2.2 SSLtunnelClt

```
/*******************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.        **
** Iena is protected from copyrigth© 2005-2050 International licence.           **
**                                                                              **
**  IENA DEVELOPERS:                                                            **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                              **
**                                  marco.ramilli@studio.unibo.it               **
**                                  eth0up@rrsecurity.info                      **
**                                                                              **
**      IVANO MANCA                 ivano.manca@gmail.com                       **
**                                  ivano.manca@studio.unibo.it                 **
*******************************************************************************/
package ienaclient;

import java.io.*;
import java.net.*;
import javax.net.ssl.*;
import ienaclient.Socket2Socket;
/**
 *
 * @author ivanomanca
 */
public class SSLtunnelClt {
    SSLSocket ienaServerSocket = null;      //Socket per la comunicazione con ienaServer
    String ienaServerIP;
    int ienaServerPort;
    String pwK;
    String pwT;

    /** Creates a new instance of SSLtunnelClt */
    public SSLtunnelClt(String keystorepw, String truststorepw) {
                // RECUPERO PASSWORD KEYSTORE
                this.pwK = keystorepw;
                // RECUPERO PASSWORD TRUSTSTORE
                this.pwT = truststorepw;
```

```
                //NOME DEL FILE DEL KEYSTORE
                System.setProperty("javax.net.ssl.keyStore", "ienacltkeystore");
                // PASSWORD ACCESSO AL KEYSTORE
                System.setProperty("javax.net.ssl.keyStorePassword", pwK);
                // KEYSTORE IN FORMATO PKCS#12
                //System.setProperty("javax.net.ssl.keyStoreType", "PKCS12");
                // FILE DEL TRUSTSTORE
                System.setProperty("javax.net.ssl.trustStore", "ienaclttruststore");
                // PASSWORD TRUSTSTORE
                System.setProperty("javax.net.ssl.trustStorePassword", pwT);
                // TRUSTSTORE IN FORMATO JAVA KEYSTORE
                //System.setProperty("javax.net.ssl.trustStoreType", "JKS");

    }
    public void createSSLSocket(String serverIP, int serverPort) throws IOException{
        this.ienaServerIP = serverIP;
        this.ienaServerPort = serverPort;
        SSLSocketFactory sslsocketfactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
        ienaServerSocket = (SSLSocket) sslsocketfactory.createSocket(ienaServerIP,
ienaServerPort);
    }
    public void cipherSuitesOn(){
        // recupero degli algoritmi di cifratura supportati
        String[] scs = ienaServerSocket.getSupportedCipherSuites();
        // stampa degli algoritmi di cifratura supportati
        //for (int i=0; i<scs.length; i++) System.out.println(scs[i]);

        /*  ATTIVAZIONE ALGORITMI DI CRITTOGRAFIA */
        String[] ascs = new String[scs.length];
        int ascsl = 0;

        for (int i=0; i<scs.length; i++) {
        if (scs[i].indexOf("_anon_") > 0)
        ascs[ascsl++] = scs[i];
        }
        String[] oecs = ienaServerSocket.getEnabledCipherSuites();
        //stampa algoritmi oecs
        //System.out.println("++++++++++++++ OECS +++++++++++++++++");
        //for (int i=0; i<oecs.length; i++) System.out.println(oecs[i]);

        String[] necs = new String[oecs.length + ascsl];
        System.arraycopy(oecs, 0, necs, 0, oecs.length);
        System.arraycopy(ascs, 0, necs, oecs.length, ascsl);
```

```
        //Scelta della cipher suite
        //String[] csuite = {"SSL_RSA_WITH_RC4_128_MD5"};
        ienaServerSocket.setEnabledCipherSuites(necs);
        //ienaServerSocket.setEnabledCipherSuites(csuite);
    }
    public SSLSocket getSocket(){
        return ienaServerSocket;
    }

}
```

5.2.3 Socket2Socket

```
/*******************************************************************************
** Iena is a alternative security system, it was born in Cesena in 2005.        **
** Iena is protected from copyrigth© 2005-2050 International licence.           **
**                                                                             **
**  IENA DEVELOPERS:                                                           **
**      MARCO RAMILLI (aka eth0up)  gramill@tin.it                             **
**                                  marco.ramilli@studio.unibo.it              **
**                                  eth0up@rrsecurity.info                     **
**                                                                             **
**      IVANO MANCA                 ivano.manca@gmail.com                      **
**                                  ivano.manca@studio.unibo.it                **
*******************************************************************************/
package ienaclient;


import java.net.*;
import javax.net.ssl.*;
import java.io.*;

/*
 *
 * Classe ausiliaria per la comunicazione tra socket
 *
 */
public class Socket2Socket implements Runnable{

        private boolean isTH2;
        private Socket clientSocket;
```

```
        private SSLSocket serverSocket;
        private int serverPort;

        // Costruttore
        public Socket2Socket(Socket clientSocket, SSLSocket serverSocket, int serverPort,
boolean check) {

                this.clientSocket = clientSocket;
                this.serverSocket = serverSocket;
                this.isTH2 = check;
                this.serverPort = serverPort;
                Thread thr =  new Thread(this);
                thr.start();
        }

        public void run() {

                if (isTH2) {
                        Client2Server_run();
                } else { // TH3
                        Server2Client_run();
                }
        }

        private void Client2Server_run() {
                int len = 0;
                byte[] rec = new byte[1000];
                String flag = new String();
                try{
                        InputStream in = clientSocket.getInputStream();
                        OutputStream out = serverSocket.getOutputStream();
                        //while((len = in.read(rec))>0) {
                                //out.write(rec,0,len);
                        flag=   "\nATTACK FROM HOST:
"+clientSocket.getRemoteSocketAddress()+
                                   "\nTO HOST:"+clientSocket.getLocalAddress()+" IENA@PORT
"+serverPort;
                        out.write( flag.getBytes() );
                        //}
                        if(len<0){
                              System.out.println("Socket closed by ienaClient");
                              serverSocket.close();
                              clientSocket.close();
                          }
```

```
                    serverSocket.close();
                    clientSocket.close();

                    //in.close();out.close();//chiusura dei rimanenti stream
            }catch(IOException e){
                    System.out.println("Socket closed by ienaServerSSL");
            }

    }

    private void Server2Client_run() {
            int len = 0;
            byte[] rec = new byte[1000];

            try{
                    InputStream in = serverSocket.getInputStream();
                    OutputStream out = clientSocket.getOutputStream();
                    while((len = in.read(rec))>=0) {
                            out.write(rec,0,len);
                    }
                    if(len<0){
                          System.out.println("Socket close by ienaServer");
                          serverSocket.close();
                          clientSocket.close();
                      }
                    serverSocket.close();
                      clientSocket.close();
                      //in.close();out.close();
            }catch(IOException e){
                    System.out.println("Socket closed");
            }
    }
}
```

5.3 Avvio e configurazione di IENA

I file eseguibili del sistema IENA, interamente scritto in JAVA, si dividono tra un file JAR per il **clientIENA** e un file JAR per il *serverIENA*.

Per lanciare sia il server che il client ci serviamo della *"shell"* se siamo su un sistema Unix, o del *"prompt"* dei comandi se siamo su un sistema Windows.

Per lanciare il server bisogna conoscere la password del keystore che contiene le chiavi per l'autenticazione. Per lanciare il clientIENA oltre alle password dei keystores bisogna conoscere la password d'amministratore (in ambiente Unix). Oltre ai file JAR bisogna essere in possesso dei certificati del server e del client, se non se ne ha una copia, bisogna crearli e importarli nei keystores del peer (es: il certificato del client va importato nel keystore del server) con l'utilizzo di *"Keytool"* (vedi Cap 4, par. 3).

La stringa di lancio del **serverIENA** può essere di due tipi:

1. Stringa di default: [**java -jar ienaServerSSL <KeystorePassword>**];
2. Stringa di lancio completa: [**java -jar ienaServerSSL <KeystorePassword> <serverPort>**];

Nel primo caso, il **serverIENA** viene lanciato in ascolto sulla porta di default che è la numero 20000. Di seguito l'esempio del lancio del **serverIENA** su un S.O. Mac OS X utilizzando la stringa di default:

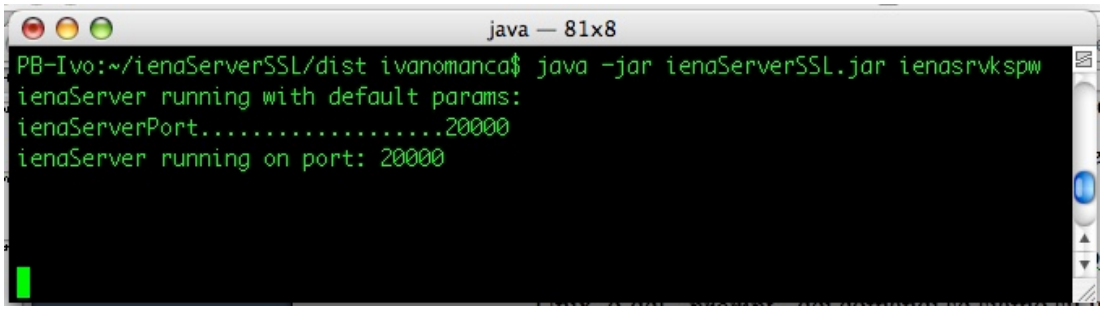


Fig. 29: Esecuzione default del serverIENA

Nel secondo caso, il **serverIENA** viene lanciato in ascolto sulla porta specificata dall'utente. Di seguito l'esempio del lancio del **serverIENA** su un S.O. Mac OS X

utilizzando la stringa completa e specificando la porta 12000:

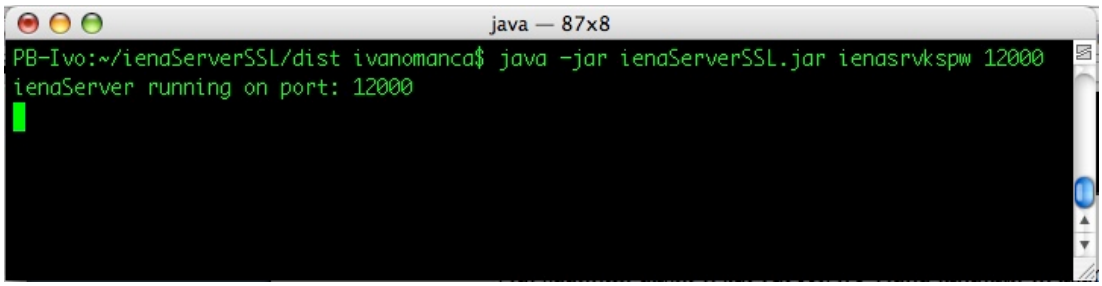


Fig. 30: Esecuzione parametrizzata del serverIENA

La stringa di lancio del **clientIENA** può essere di due tipi:

1. Stringa di default: [**sudo java -jar ienaClient.jar <ienaClientKeystorePassword> <ienaClientTruststorePassword>**];
2. Stringa di lancio completa: [**sudo java -jar ienaClient.jar <ienaClientKeystorePassword> <ienaClientTruststorePassword> <ienaPort> <ienaServerIP> <ienaServerPort>**];

Nel primo caso, il **clientIENA** viene lanciato in ascolto sulla porta di default che è la numero 80. I parametri di default per l'IP e la porta del server sono 127.0.0.1 e 20000. Di seguito l'esempio del lancio del **clientIENA** su un S.O. Mac OS X utilizzando la stringa di default:

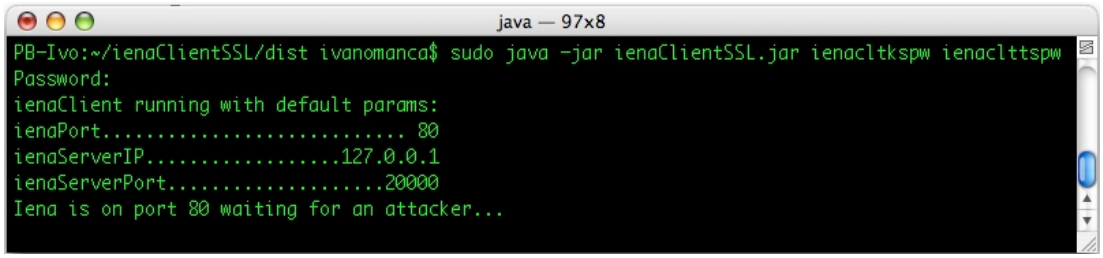


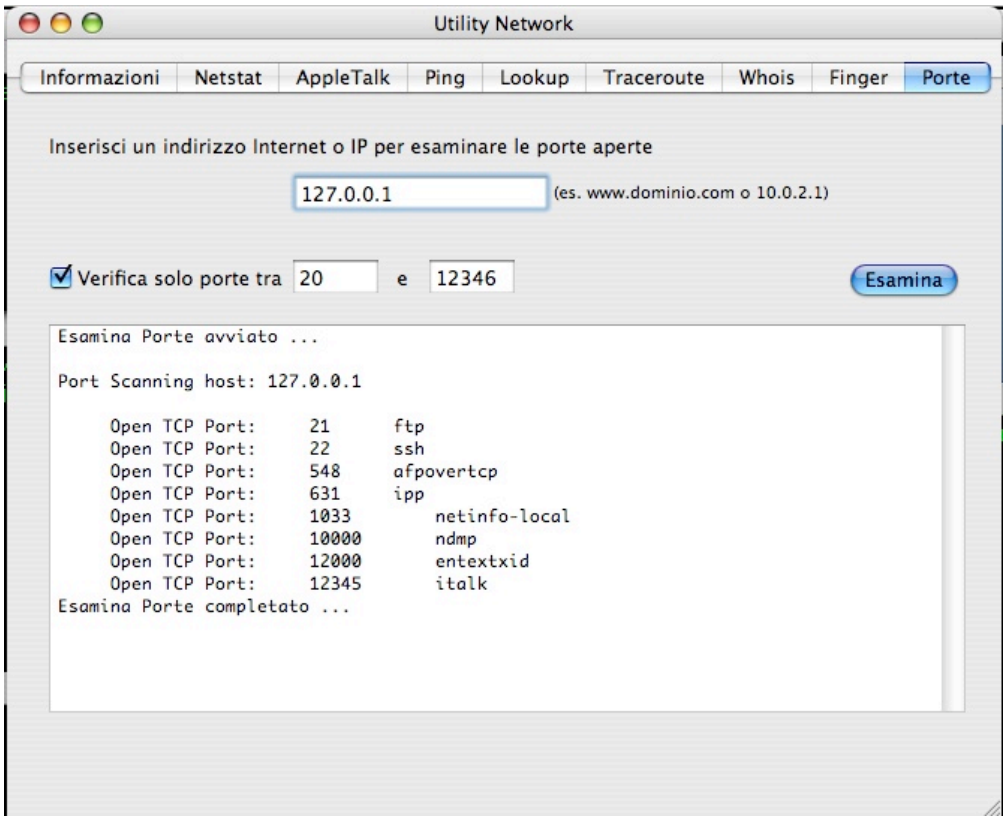
Fig. 31: Esecuzione default del clientIENA

Come nel caso del serverIENA, il **clientIENA** può essere lanciato in ascolto sulla porta specificata dall'utente, e ovviamente con IP e porta del **serverIENA**. Di fatti, dovrebbero essere lanciati più **ienaClient** (ognuno su una porta diversa) per assicurare un maggiore livello di sicurezza. In ogni caso è consigliata una configurazione tipica

che prevede la presenza dei **clientIENA** sulle porte:

- **21**: porta del protocollo ftp;
- **12345**: porta utilizzata da NetBus, uno dei trojan più diffusi;
- **10000** e **12000**: per il worm Nimda e la sua varinte 2;

Di seguito, grazie ad *"un port scan"*, (uno screenshoot di come) appaiono i falsi servizi generati dal lancio dei **clientIENA** (S.O. Mac OS X) sulle porte consigliate dalla configurazione tipica:



Tra le porte aperte, abbiamo le porte 21, 10000, 12000, 12345 con i relativi falsi servizi. Il *"port scan"* per la rivelazione dei servizi, ha risvegliato i quattro **clientIENA**, i quali, al tentativo di connessione hanno notificato (tramite i loro quattro pacchetti informativi) al **serverIENA** il tentativo d'attacco. Di conseguenza, il serverIENA logga sul file *ienaLog.log* i dati corrispondenti ad attaccanti e clientIENA risvegliati.
Di seguito il file di log del serverIENA dopo l'esecuzione del *"port scan"*:

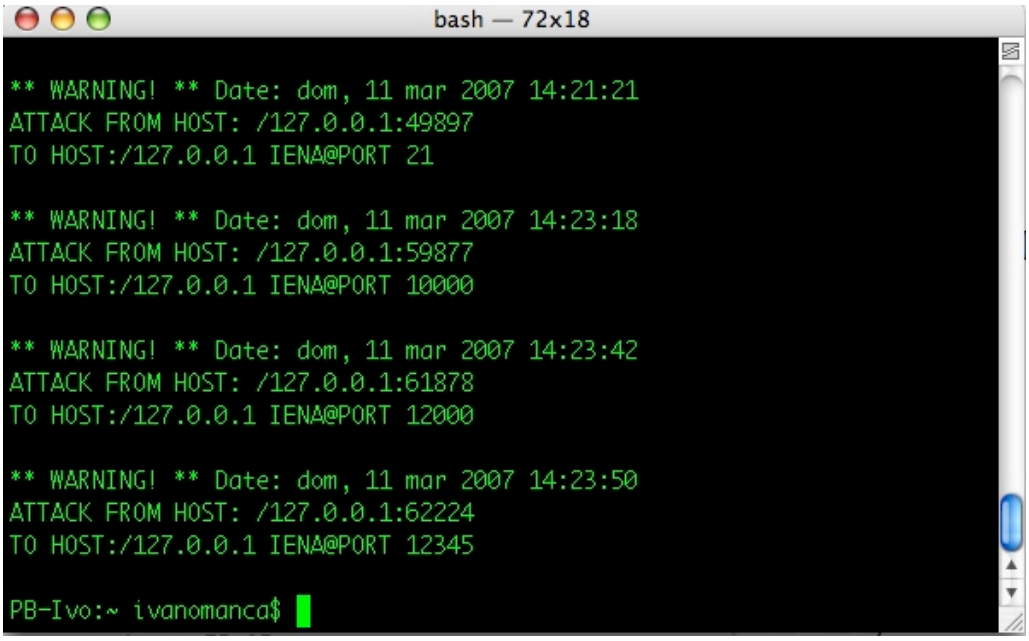


Fig. 32: Log file del serverIENA

E' evidente che la vicinanza dell'ora d'attacco e la sequenza crescente delle porte corrispondono all'azione di un *"port scan"*.

Capitolo 6

SPERIMENTAZIONE DI IENA

In questo capitolo è descritta una sperimentazione di base del comportamento IENA, e una verifica dello stesso in termini di autenticazioni e connessioni sicure tra serverIENA e clientIENA.

6.1 Verifica della connessione criptata

Come discusso nei capitoli precedenti, ogni qualvolta viene rivelato dal client un tentativo d'attacco, tra clientIENA e serverIENA viene instaturata una connessione sicura per lo scambio di pacchetti informativi che descrivono i dettagli dell'attacco stesso. Nella prima implementazione di IENA era prevista una comunicazione tramite l'utilizzo di semplici socket TCP che realizzavano una comunicazione non autenticata, e soprattutto in chiaro. Questo vuol dire che, qualunque server, o qualunque client, configurati opportunamente, potevano compromettere la comunicazione e modificare il comportamento del sistema IENA. Questo tipo d'attacco è noto con il nome di *"Man In The Middle"(vedi par. 6.1.2)*, letteralmente tradotto con *"Uomo Nel Mezzo"*.
E' interessante verificare come sia mutata radicalmente la comunicazione tra clientIENA e serverIENA nel passaggio tra le due implementazioni: IENA e IENASSL.
Con l'utilizzo di strumenti quali *"tcpdump"* e *"ethereal"* si ha la possibilità di sniffare il traffico nelle connessioni tra clientIENA e serverIENA, e di indagare sul contenuto dei pacchetti informativi scambiati tra i due peer. Nella figura seguente viene riportata una visualizzazione del contenuto dei pacchetti nella comunicazione in chiaro tra clientIENA e serverIENA relativi alla prima implementazione del progetto:

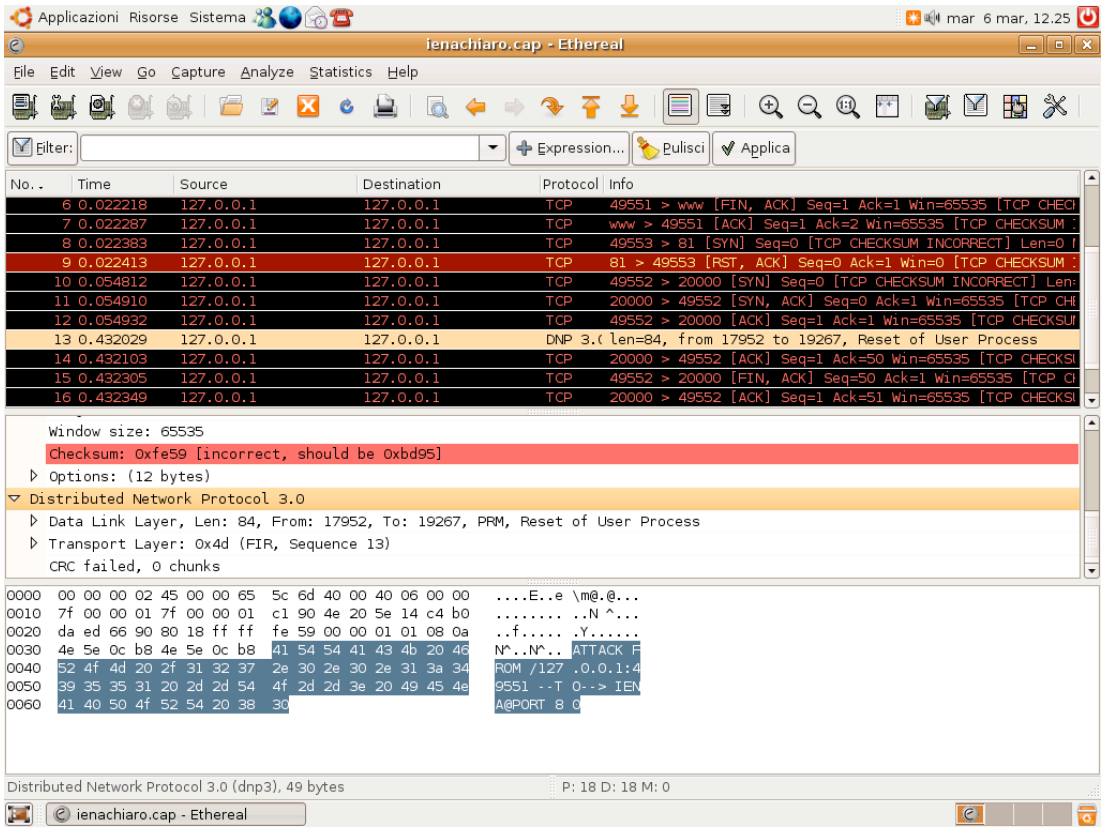


Fig. 33: Schermata di ethereal sulla comunicazione in chiaro

Grazie ad *ethereal*, riveliamo con semplicità, il contenuto della comunicazione in chiaro tra clientIENA e serverIENA. La zona in basso evidenziata in blu, rivela palesemente la stringa **"ATTACK FROM /127.0.0.1:49551 --TO→ IENA@PORT 80"**. Quest'utima altro non è che la stringa informativa contenente i parametri dell'attaccante inviata dal clientIENA.

Ripetiamo l'esperimento sniffando il traffico tra clientIENA e serverIENA relativi alla seconda implementazione, ovvero alla versione provvista della comunicazione sicura attraverso protocollo SSL e mutua autenticazione tra i peer. Di seguito la schermata di *ethereal* simile alla precedente:

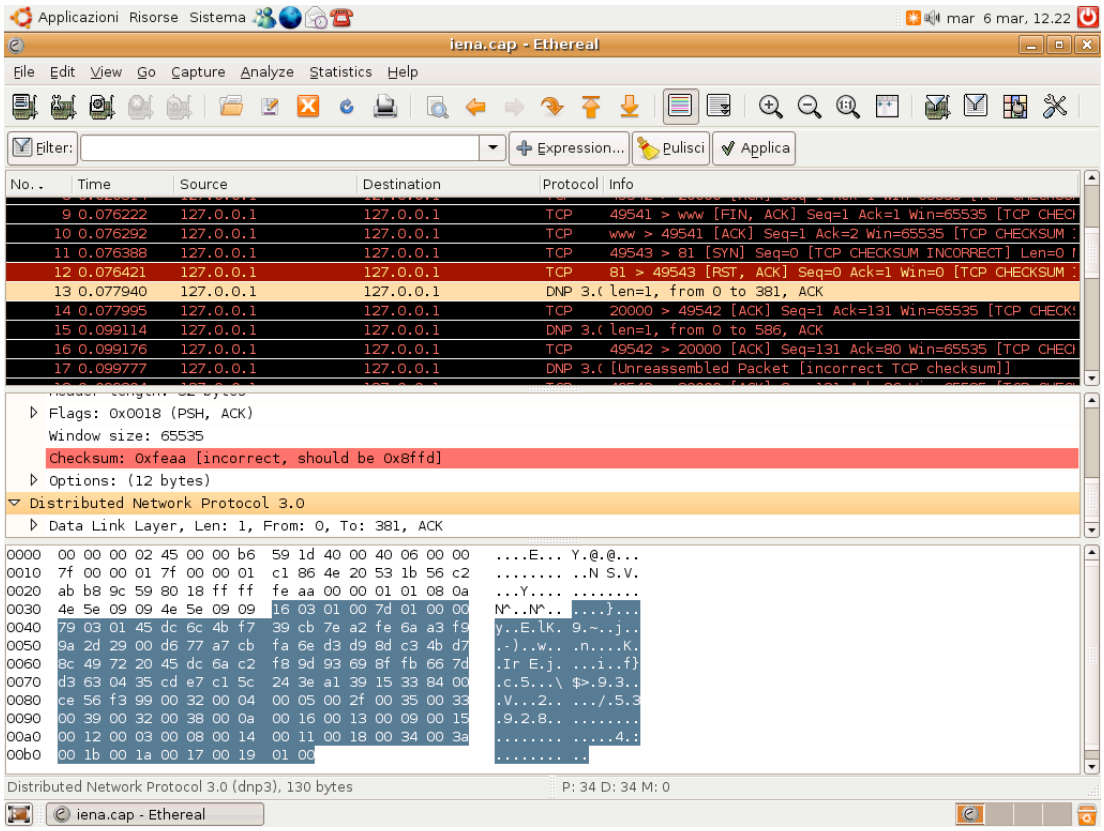


Fig. 34: Schermata di ethereal sulla comunicazione criptata

Il campo evidenziato in blu questa volta non rivela altro che una sequenza di caratteri incomprensibili non solo all'occhio umano, ma anche a qualsiasi interprete del protocollo, poiché si tratta di un testo cifrato. L'unico modo per decifrare questa comunicazione, è quello di essere un serverIENA, conosciuto dai suoi clientIENA attraverso il certificato, e possedere la chiave pubblica del clientIENA.

6.2 Man in the middle attack

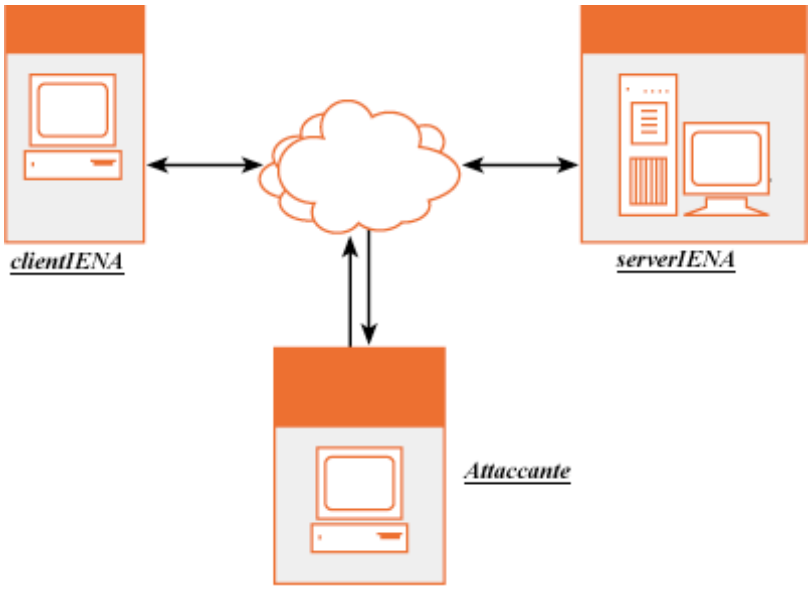


Fig. 35: Schema tipico di un MITMA

La tecnica *"Man in the middle"* è una tipologia d'attacco nella quale il cracker che la utilizza ha la capacità di cattuare e talvolta modificare a suo piacimento le informazioni scambiate tra due parti coinvolta senza che le stesse si accorgano minimamente di ciò che sta accadendo. Il traffico generato dalla comunicazione può essere analizzato attraverso appositi tool che ne rivelano il contenuto.
Un attacco Man in the middle rappresenta un problema piuttosto grave per i sistemi che implementano le connessioni in chiaro e purtroppo anche per sistemi che adottano crittografia asimmetrica. E' per questo motivo che per l'implementazione di IENA è stata scelta una cifratura a chiave pubblica con mutua autenticazione.

6.3 Verifica autenticazione del clientIENA

La nuova implementazione di IENA prevede una connessione sicura tra clientIENA e serverIENA con mutua autenticazione. Grazie al protocollo SSL/TLS i due peer, si

assicurano la confidenzialità nello scambio di dati tramite la verifica diretta dei certificati che attestano l'identità del peer con il quale va instaurandosi la connessione.
La figura seguente descrive in breve le fasi dell'handshake tra i peer:

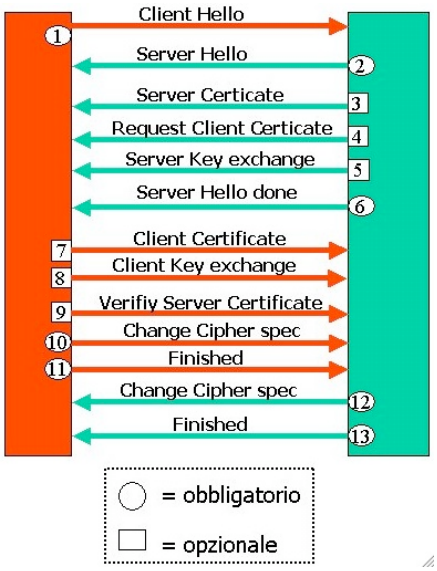


Fig. 36: Handshake nel protocollo SSL/TLS

Come è noto dall'architettura del protocollo SSL, durante l'handshake, i due peer si autenticano a vicenda e negoziano l'algoritmo di cifratura che sarà utilizzato in seguito per lo scambio dei dati. In questa prova, verificheremo il comportamento del serverIENA in presenza della richiesta di connessione da parte di un client a lui sconosciuto.
La figura che segue è la schermata della *shell* dove è in esecuzione il serverIENA provvisto del protocollo SSL. Il serverIENA riceve un tentativo di connessione da parte di un client sprovvisto di una connessione sicura con SSL/TLS:

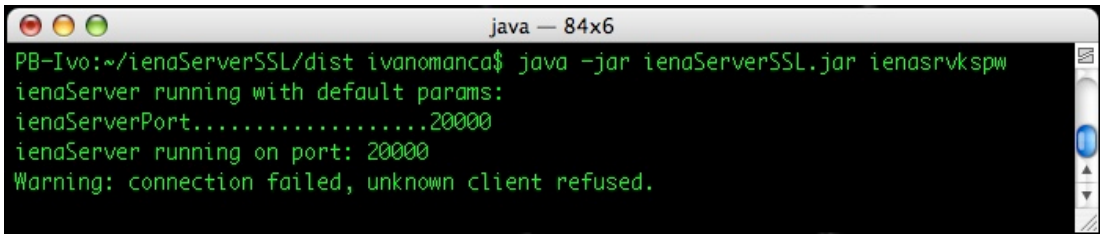


Fig. 37: Connessione da client sconosciuto

Il serverIENA rifiuta la connessione con il client sconosciuto, e torna nello stato di *wait*.
Il comportamento del serverIENA è esattamente quello voluto, quindi in questo caso il client non riesce a influenzare e/o a danneggiare il sistema perché è stato classificato dal serverIENA come client sconosciuto (*unknown client*).
Il client della prova appena terminata non era provvisto del protocollo SSL/TLS, era quindi abbastanza intuitivo il comportamento del serverIENA che a priori rifiuta la connessione. E' interessante verificare quindi, il comportamento del serverIENA davanti alla richiesta di connessione da parte di un client provvisto del protocollo SSL/TLS, ma che possiede un certificato sconosciuto al serverIENA, il quale si fida solo dei certificati validati nel suo keystore.
La figura seguente è la schermata della *shell* dove è in esecuzione il serverIENA provvisto del protocollo SSL:

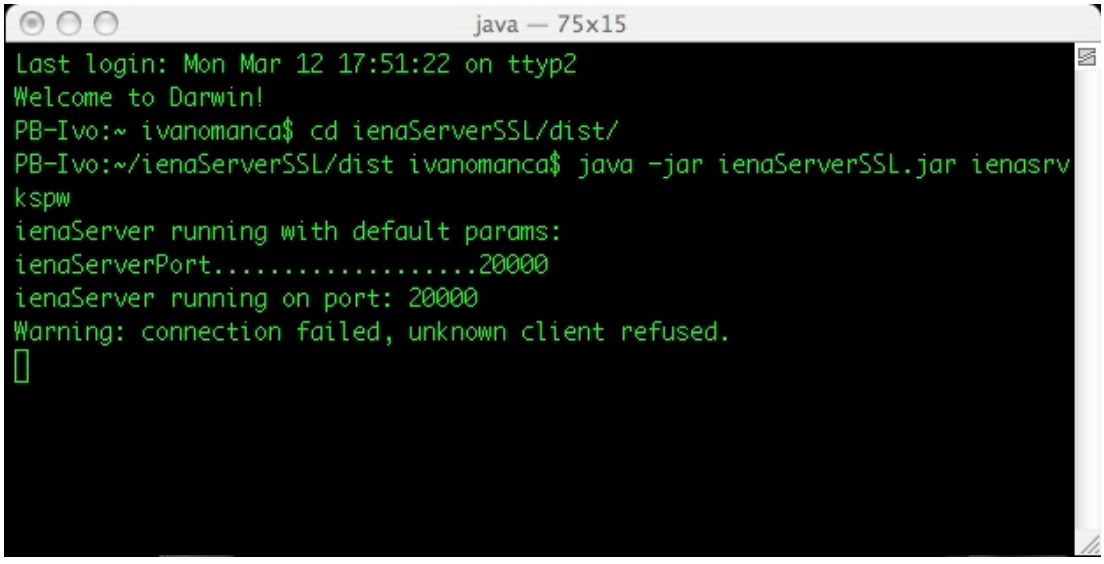


Fig. 38: Connessione da client sconosciuto - 2a prova

Il serverIENA rifiuta la connessione con il client, e torna nello stato di *wait*.
Il comportamento del serverIENA è identico al precedente, poiché l'eccezione lanciata da un certificato non valido è gestita nello stesso modo. Quindi resta comunque un

tentativo di connessione da parte di un client sconosciuto (*unknown client*). Cosa accade contemporaneamente dalla parte del client sconosciuto?

La figura che segue è la schermata della shell dove è in esecuzione il client sconosciuto da noi simulato:

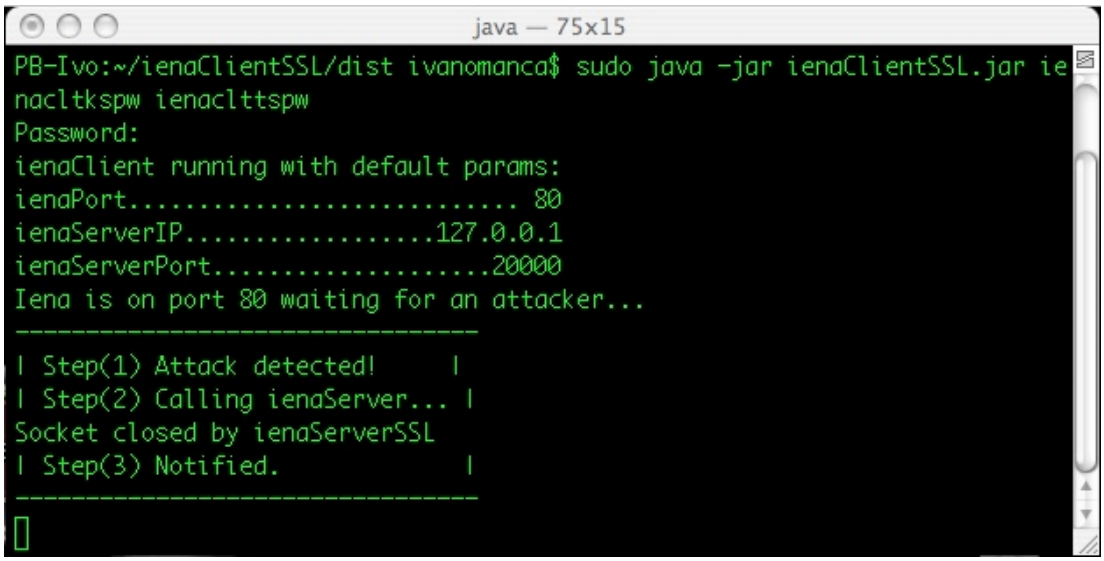


Fig. 39: Lato client

Il client dopo la chiamata al server, vede chiudersi il socket dalla parte del serverIENA. Questo costituisce un rifiuto della connessione da parte del serverIENA.

6.4 Verifica autenticazione del serverIENA

In questa seconda prova, verificheremo il comportamento del clientIENA in presenza di un serverIENA a lui sconosciuto.

La figura che segue è la schermata della *shell* dove è in esecuzione il clientIENA provvisto del protocollo SSL. Il falso serverIENA riceve un tentativo di connessione da parte del clientIENA:

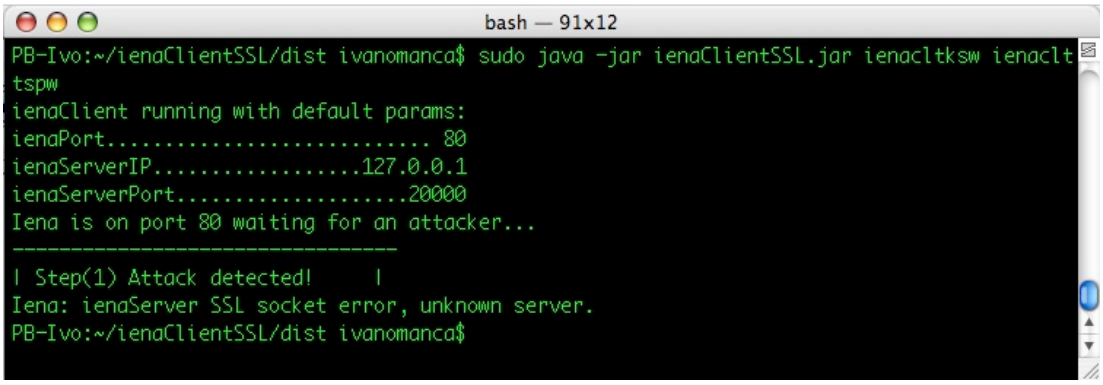


Fig. 40: Connessione su server sconosciuto

La connessione tra clientIENA e falso serverIENA non viene compiuta, o meglio, il clientIENA rileva un *"SSL socket error"* poiché il server è sconosciuto.

Il comportamento del clientIENA in presenza di un qualsiasi server che non sia il vero serverIENA è sempre lo stesso, poiché l'eccezione lanciata è gestita nello stesso modo. D'altra parte, se l'attaccante riuscisse a sostituirsi al serverIENA, il che è molto improbabile poiché il server è solo uno in esecuzione su un solo host della rete locale protetto a sua volta da clientIENA che ne impediscono l'accesso non autorizzato, avrebbe già il controllo dell'intero sistema informatico, ed in quel caso non avrebbe senso proteggere un sistema già violato.

Conclusioni

Lo sviluppo esponenziale dei sistemi informatici connessi ad Internet e la diffusione rapida dell'utilizzo dei servizi web ha portato in primo piano le problematiche relative alla sicurezza di rete. La protezione dei dati personali e la disponibilità permanente di importanti servizi in rete, dipendono quasi esclusivamente dal livello e dalle tecniche di sicurezza che adottano. Finora, il problema della sicurezza informatica è stato trattato utilizzando strumenti quali AntiVirus, Firewall, IDS, NIDS e IPS, che adottano una strategia basata su politiche *"chiuse"*. Per garantire livelli di sicurezza accettabili è fondamentale saper integrare i classici strumenti per la sicurezza tra loro e con strumenti come IENA che appartengono a una nuova concezione di difesa preventiva. Questo modello si differenzia dal tradizionale paradigma attacco/difesa, utilizzando un approccio opposto basato su politiche *aperte* per prevenire la *service exploitation*, e costituisce una novità concettuale di straordinaria efficacia.

Il progetto IENA, con la nuova implementazione presentata in questo elaborato, è divenuto Open Source ed oltre al codice sviluppato finora, è disponibile in rete tutta la relativa documentazione archiviata nello spazio web del progetto all'indirizzo http://iena.souceforge.net.

L'obiettivo principale nel futuro prossimo, è quello di coinvolgere una sempre più ampia comunità di sviluppatori che possano arricchire tecnicamente il progetto IENA portandolo ad un livello sempre più alto in termini di affidabilità, accessibilità e popolarità.

Bibliografia e sitografia

[1]. **Andrew S. Tanenbaum - Reti di Calcolatori - Prentice Hall**. *Vrije Universiteit Amsterdam Olanda. Pearson Education Italia s.r.l. (http://www.hpe.pearsoned.it);*

[2]. **Ing. Marco Ramilli, Prof. Walter Cerroni - IENA: un approccio aperto e distribuito exploitation prevention**;

[3]. **Ing. Marco Ramilli - IENA: un modello alternativo per la rivelazione delle intrusioni in una rete locale**;

[4]. **Prof**. **Claudio Salati - Dispense del corso di Reti di Calcolatori,** *Università degli Studi di Bologna;*

[5]. **http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html** *Guida di riferimento del Framework JSSE.;*

[6]. **http://docs.sun.com/source/816-6156-10/contents.htm** *Documento d'introduzione al protocollo SSL;*

[7]. **http://www.amagri.it/Sicurezza_MS_Windows/MS_IIS/SSL/dettagli.htm** *Riferimento SSL Handshake protocol;*

[8]. **https://ssa.usyd.edu.au/docs/eassec/eassec24.htm** *Implementare SSL in applicazioni JAVA;*

[9]. **http://www.javaworld.com/javaforums** *Forum sullo sviluppo in ambiente JAVA;*

[10]. **http://www.mindreef.com/support/** *Uso dei certificati SSL per la mutua autenticazione;*

[11]. **http://blog.bechelli.net/search/label/Note%2tecniche** *Forum di Luca Bechelli, SSL in Java;*

[12]. **http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html** *Riferimento della Sun a Keytool – Key and Certificate Management Tool;*

[13]. **http://openskills.info** *Riferimento per le problematiche di sicurezza su internet;*

[14]. **http://www.dia.unisa.it/~ads/corso-security/www/CORSO-0304/sicurezza_java/index.htm** *Corso d'introduzione alla sicurezza in Java del Prof.Alfredo De Santis – Università degli studi di Salerno;*

[15]. **http://www.di.unipi.it/~ricci/jsse_cont_lpr2006.pdf** *Programmazione di rete, corso del Prof. Paolo Mori - Università degli Studi di Pisa;*

[16]. **http://www.javaportal.it/** *Portale italiano sulla programmazione Java;*

[17]. **http://javaboutique.internet.com/tutorials/jkey/** *Tutorial – Uso del tool Java per la gestione dei certificati;*

[18]. **http://edenti.deis.unibo.it/LabICT/2005-2006/slide-x6/Sicurezza7.pdf** *Sicurezza di rete e JSSE, Prof. Enrico Denti – Università degli Studi di Bologna;*

[19]. **http://lia.deis.unibo.it/Courses/TecnologieSicurezzaAK/JDK-2p.pdf** *Tecnologie per la sicurezza di rete, Prof. Roberto Laschi – Università degli Studi di Bologna;*

[20]. **http://its.isti.cnr.it/** *Carlo Carlesi - Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo";*

[21]. **http://it.wikipedia.org/** *Reperimento di schemi per le reti e sistemi per la sicurezza;*

[22]. **http://www.securityfocus.com/infocus/1720** *Portale sulla sicurezza informatica;*

[23]. **x86computing.com/NIDS** *Network-Based Intrusion Detection Systems in the Small/Midsize Business - Daniel Owen - http://www.danielowen.com;*

[24]. **http://www.pcimprover.it/articolo/142/** *Articolo su Honeypot;*

[25]. **http://www.mokabyte.it/1999/10/ssl.htm** *Architettura del protocollo TLS;*

[26]. **http://www-lia.deis.unibo.it/Courses/TecnologieSicurezza/JDK-2p.pdf** *Dispense del corso di tecnologie della sicurezza tenuto dal prof. Laschi;*