Scritti da
Angelo Galavotti.

Appunti di Lab. Sicurezza

Lezioni del 23/02/2022 – Introduzione

Definizione di rischio

Con **rischio** si intende il prodotto fra la probabilità che avvenga un certo danno e l'impatto di tale danno. Per esempio: se un danno da 15000€ con probabilità che mi avvenga 4% all'anno, ho un rischio di 600€ all'anno. Posso usare il valore del rischio come un particolare indicatore. Infatti, attraverso il rischio, posso stabilire se tale valore è *accettabile* (e in tal caso non si fa nulla), se non è accettabile invece penso a delle **contromisure** (che possono ridurre il costo o la probabilità che avvenga tale danno). Le misure, ovviamente, hanno senso solo se sono convenienti.

Il processo continuo della sicurezza

La sicurezza non consiste nel valutare la situazione presente e comprare un prodotto, bensì sta nel definire un processo per tenere tracia delle continue evoluzioni dei rischi e dell'efficacia delle contromisure.

Proprietà di sicurezza

La sicurezza, in pratica, può essere scomposta in 3 proprietà chiavi, riassunte dalla sigla **CIA**:

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

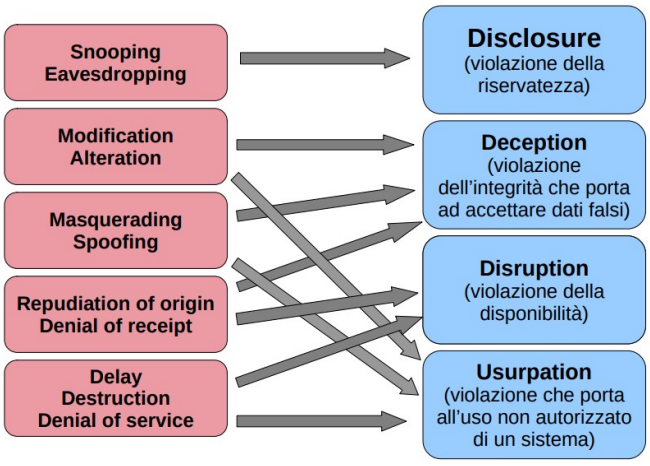
Definzione di minaccia

Una **minaccia** (*threat*) è una condizione che potenzialmente può compromettere una o più delle proprietà di sicurezza. Una minaccia esiste indipendentemente dal fatto che venga concretizzata.
Abbiamo poi:

- Attacco (attack): l'azione che porta al concretizzarsi di una minaccia.
- Attaccante (attacker): l'entità che sferra l'attacco.

Le minacce sono strettamente legate alle intenzione degli attacanti (script kiddie = bimbominchia hackerino coglioncino).

Tipologie di attacchi e minacce (sempre definizioni)



- **Snopping e Eavesdropping**: sono attacchi "passivi", che attaccano la riservatezza delle informazioni. (Dunque la minaccia consiste nella disclousre, ovvero di svelare informazioni).
- **Modification/Alteration**: si viola l'integrità dei dati, portanod ad accettare anche dati falsi (porta a Deception, ovvero inganno). Stessa cosa vale per la mincaccia di **masquerading** e **spoofing**, che consistono nella falsificazione dell'identità e delle informazioni rispettivamente.
- **Repudiation of Origin, Denial of Receipt**: è un attacco più aggressivo, consiste nel negare una parte attiva di un protocollo e porta alla violazione della disponibilità dei dati.
- **Denial of Service, Destruction**: porta alla'uso non autorizzato di un sistema.

Politiche e Meccanismi

Per la 2312 volta, si intende:

- Una **politica di sicurezza** (security policy) è la dichiarazione di ciò che è consentito o proibito fare. Cambiano in base alle nostre esigenze.
- Un **meccanismo di sicurezza** (security mechanism) è un metodo, uno strumento o una procedura per far rispettare una politica di sicurezza. Cambiano in base alle tecniche, budget etc...
- Non sono necessariamente tecnici, anzi molto spesso, tra i più importanti, ci sono comportamenti e regole di interazione tra persone.

Le politiche dichiarano qual è il *fine* della sicurezza, i meccanismi specificano il *mezzo* per contrastare gli attacchi, e possono combinare diverse strategie:

- **Prevenzione** (prevention): l'attacco deve fallire (aka implemento un meccanismo tale per cui l'attacco fallisca). La porta e la chiave è un meccanismo di prevenzione, infatti chi non ha la chiave semplicemente non entra. Questi meccanismi solitamente sono invasivi, e l'implementazione è inalterabile e non aggirabile.
- **Rilevazione** (detection): l'attacco potrebbe avere successo, ma deve essere notato e riportato. Non ci importa quindi che l'attacco avvenga. Ad esempio, potrei avere un documento in cui c'è bisogno di una firma, e uno potrebbe falsificare la firma. Se però io ho un calligrafo molto bravo che può notare subito che la firma è falsa e che poi strappa il documento, allora non mi importa che l'attacco (ovvero la falsificazione dell'identità) possa avere successo. Questo metodo è inefficace rispetto ad alcune minacce, es. disclosure.
- **Reazione** (response): l'attacco rilevato viene mitigato per ridurre la gravità o l'estensione del danno.
- **Ripristino** (recovery): le conseguenze dell'attacco vengono ridotte o azzerate, ripristinando le proprietà di sicurezza violate. Ovviamente, a volte è possibile e a volte no.

Superficie d'attacco

Una **superficie d'attacco** è l'insieme dei **vettori d'attacco** (ogni modo reso accessibile ad un possibile attaccante di interagire con il nostro sistema), ovvero di tutti i modi possibili in cui il mio sistema può essere raggiunto. L'insieme dei vettori d'attacco si divide in tre grandi aree:

- Tramite le persone (*social engineering*, convincere le persone a violare le politiche di sicurezza).
- Parte cyber/telecomunicazioni (ovvero via cavo e via wireless)
- Fisico (accesso materiale alle risorse)

Si ha una vulnerabilità quando si hanno difetti nell'implementazione di un meccanismo o quando si ha una definizione incompleta delle politiche di sicurezza.

Vulnerabilità

Se le politiche e i meccanismi di protezione di un sistema fossero perfetti, le minacce non potrebbero concretizzarsi, e quindi si neutralizzano i vettori di attacco. Questo però non è mai vero. Gli attacchi hanno successo se esistono errori:

- Nell'individuazione della superficie di attacco (si parla **porosità** – un vettore esiste là dove non dovrebbe).
- Nella definizione di una politica o nell'implementazione di un meccanismo (**vulnerabilità** / vulnerability). Un difetto del genere può essere strutturale nell'hardware o software, oppure può dipendere da un uso scorretto.
- Con **exploit** si intende uno strumento per trarre vantaggio da una vulnerabilità concretizzando una minaccia. Può essere tecnico (cracking), o umano (social engineering).

Difesa

Esistono *framework* e *metodologie* per aiutare nella sistemizzazione della messa in sicurezza.
Ci sono poi le *certificazioni*, che permettono di dare evidenza che misure efficaci sono state adottate per mettere effettivamente in sicurezza il sistema.

Lezioni del 25/02/2022 – Introduzione all'offensive security, terminologie

Cos'è l'offensive security

L'**offensive security** consiste nel porsi nel ruolo dell'attaccante per verificare l'esistenza di una vulnerabilità, stimare con precisore l'impatto degli attacchi e per testare l'efficacia delle contromisure.

Usare le stesse tecniche degli attacanti può essere qualcosa di complicato. Infatti, bisogna stare ovviamente molto attenti a non fare danni.

Quando facciamo penetration testing, non possiamo dimostrare veramente l'assenza di problemi nella sicurezza di un sistema. L'unica cosa che possiamo fare è, invece, sollecitare il sistema nel modo più completo possibile, in modo da trvare eventuali problemi esistenti.

Livelli di approfondimento

Abbiamo tre livelli di approfondimento:

- Vulnerability Assessment
- Penetration Testing
- Red Team Operations

Vulnerability Assessment

Il VA si basa sul fatto che la comunità di "sicuristi" pubblica le informazioni su una vulnearbilità scoperta, secondo un principio detto di Vulnerability Disclosure (o ***responsible disclosure***).
Di norma, la comunità viene avvisata DOPO aver avvisato il produttore del software.
Ad ogni exploit, viene dato un indicatore di "gravità" della vulnerabilità, e non sempre l'exploit viene reso pubblico.
Esistono software per cercare vulnearbilità nei sistemi direttamente (es. OpenVAS) e database di exploit pronti per essere sfruttati (exploit-db, cvedetails…).
[CVE = Common Vulnerability and Exposures].

I dati di questi db possono essere human readable (con dettagli leggibili da persone) oppure machine readable, che contengono dati usati automaticamente delle macchine per esempio durante i testing delle vulnerabilità automatici.

I limiti del VA stanno nel fatto che i VA possono trovare solo vulnearbilità già note, e non può procedere oltre.
Non sfrutta quindila vulnerabilità, bensì dice solo se è presente. Un vero attaccante, invece, potrebbe benissimo usarla per accedere ad una vista più profonda del sistema, magari svelando altre vulnerabilità nel processo e per prendere il controllo del sistema.
Il VA si ferma solo sulla superficie, mentre un vero attaccante andrebbe a fondo.

Inoltre, il processo di VA va tarato per fare in modo che non faccia abbastanza rumore: per esempio, se ho un server di una versione vecchia ma con le vulnerabilità patchate, devo fare in modo che queste vengano ignorate etc… è un problema di tuning dei parametri che può essere faticoso e suscettibile ad errori.

Nel VA, solitamente il tester non è umano.
Dunque, se vogliamo aumentare il grado di profondità di analisi della sicurezza, arriviamo **al penetration testing**.

Penetration Testing

Nel PT, un tester *umano* avanza fin dove può, sfruttando le varie vulnerabilità per mezzo di un

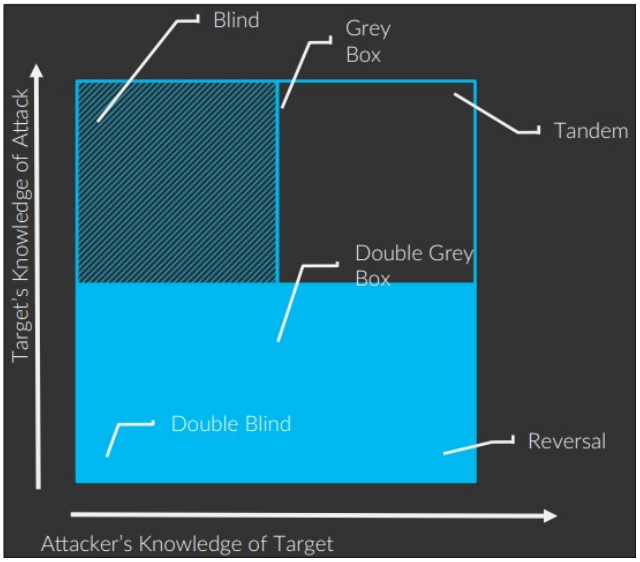
exploit.
Essenzialmente quindi, dopo che un VA fa i test in modo automatico, il penetration tester prende i risultati e li va a spulciare e ad approfondire.
Potrebbe:

- Trova una vulnerabilità, e c'è l'exploit: lo lancia e guarda fin dove può arrivare
- Trova una vulnerabilità, e non c'è l'exploit: se lo segna e poi lo esamina. Magari si costruisce l'exploit.

Mentre il PT è più realistico, è anche più rischioso (potrebbe danneggiare il sistema nel mentre) e il report che dovrà scirvere sarà ancora più dettagliato [che sbatta sti report fra…]

Per fare PT, devo avere dei punti di partenza:

- Prima di tutto, devo **valutare il mio target**: questo vuol dire che devo considerare la mia *postura* (come attaccante, sono parte dell'azienda oppure sono all'esterno?) e *visibilità* (devo lavorare alla cieca, ovvero non so nulla del sistema, oppure no? Devo invece lavorare in doppio cieco, in cui oltre che a non sapere nulla del sistema, il reparto di difesa dell'azione non sa che sta per essere attaccato?). A questo normalmente si aggiunge anche la mappatura del mio sistema, valutazione delle priorità… [NOTA: mentre l'attacco doppio cieco sembra quello più plausibile, solitamente invece viene considerata una perditemo per i PT, siccome deve scoprire dei dettagli che potrebbe sapere già dal proprio cliente]



- Devo stare attento a non danneggiare il bersaglio, quindi devo dare una certa importanza alla **protezione del mio bersaglio** durante il processo. Dunque, potrei provare ad usare un sistema replicato, ma non sempre è così semplice e potrei perdere dettagli importanti.

Metodologie per il Penetration Testing

Seguire una metodologia mi permette di essere oggettivo e di ripetere il test più volte sullo stesso sistema per capire se effettivamente è cambiato qualcosa.

Una delle metodologie più diffuse è **Open Source Security Testing Methodology Manual (OSSTMM)**. Consente a qualsiasi tester di sicurezza di fornire idee per eseguire test di sicurezza più accurati, attuabili ed efficienti. Inoltre, utilizzabile da chiunque senza royalties: permette la libera diffusione delle informazioni e della proprietà intellettuale. Questa metodologia è abbastanza generica, ma ce ne sono alcune specifiche in base al contesto. Tra queste abbiamo:

- Open Web Application Security Project (OWASP), specifico per applicazioni web
- Payment Card Industry Data Security Standard (PCI DSS), settore finanziario
- Technical Guide to Information Security Testing and Assessment (NIST800-115), uno standard ufficiale del governo degli Stati Uniti.

Preparazione del Penetration Testing

1. **Reconnaissance** (ricognizione). In questa fase vengono raccolte le informazioni utili e si estende il perimetro dei test.
2. **Enumeration** (enumerazione). Si delimita il perimetro del test e si fa una verifica puntuale delle risorse e delle loro proprietà.

OSINT (Open Source Intelligence)

È una delle prime parti del Penetration Testing, e se vogliamo una delle più "sociali". È l'uso di qualsiasi fonte comunemente accessibile (da qui Open Source) per avere delle informazioni su un

obiettivo specifico. È un campo molto più ampio rispetto alla cybersecurity.
L'OSINT può essere fatta:

- su altri, per esaminare l'intelligence (la superficie d'attacco) di una certa minaccia
- su sé stessi, per capire cosa posso scoprire sugli avversari e come possono essere usate queste informazioni.

Ci sono un sacco di strumenti online di OSINT per capire la collocazione fisica di una infrastruttua, la collocazione in rete e buchi di accesso ai servizi.

Enumeration su internet

Lo IANA è l'ente che assegna blocchi di indirizzi IP ai RIR (Regional Internet Registry), che a loro volta li suballocano ai LIR (Local Internet Registry), che tengono traccia di chi è l'effettivo responsabile di ogni blocco.
Posso usare "WHOIS" (il protocollo) su RIPE per esempio per sapere le informazioni di un certo IP.
Oltre agli IP, sono molto utili i DNS (di cui ancora se ne occupa lo IANA).
Ogni host raggiungibile via IP può esporre punti di accesso (ad esempio se il loro sistema ha un processo TCP in ascolto).

In particolare, i record DNS possono svelare

- gli IP registrati dall'obiettivo
- l'esistenza e la collocazione di specifici server applicativi
- l'esistenza di sottoreti non direttamente raggiungibili
- alias per sistemi collocati al di fuori del perimetro dell'obiettivo
  - risorse in cloud
  - sistemi legati da relazioni di fiducia es. domini di una foresta Active Directory

Questo consente un notevole risparmio di tempo rispetto alla forza bruta
La cosa peggiora quando si ha l'abilitazione di domain transfer, oppure permanenza di record rimossi nelle cache di qualche major DNS provider.
Alcuni strumenti utili per analisi DNS sono:

- per lookup di base: host, dig, nslookup
- strumenti di ricerca che includono guessing e forza bruta: dnsenum, dnsmap, dnsrecon, fierce…

Una volta che abbiamo individuato un blocco di indirizzi da analizzare, procediamo in questi modi:

- Facciamo un ping ad ogni indirizzo.
- Facciamo una scansione massiva con masscan
- Usiamo degli strumenti **su rete locale** (packet sniffing con tshark, wireshark, tcpdump…) o arping.

Una volta che si sono determinati gli host raggiungibili, scannerizziamo per vedere se ci sono delle porte aperte. Per fare ciò, potremmo fare uso di `nmap`, che scannarizza un range di indirizzi e di porte. Un altro tool usabile è `unicornscan`.

Evasione

Dobbiamo impedire che noi veniamo scoperti da altri.

Dobbiamo fare ricognizione in modo anonimo, e per farlo dobbiamo usare dei tool per rendere il nostro traffico anonimo (come ToR), creare account throwaway oppure usare delle VM diverse e sostituite periodicamente.
Anche l'enumerazione va fatta in modo anonimo: randomizzazione di indirizze e porte, e non

utilizzare lo stack TCP/IP dell'host d'origine per evitare il reverse fingerprinting.

Tentativi di accesso ai servizi

Una volta che abbiamo identificato quali sono i servizi attivi, possiamo fare i primi tentativi di accesso. Ci sono i client di base di tutti i protocolli applicativi più noti (SMTP, SMB, SNMP). Inoltre, è possibile fare **fuzzing**, che è una sorta di brute forcing applicativo che consiste nell'inviare dei payload randomizzati per tentare di sollecitare risposte impreviste (Es. `bed`, `doona` sono dei tool per questo).

[Ha spiegato tipi di attacco per WiFi, ma molto a grandi linee e con esempi più che altro]

Ripasso del filesystem UNIX

[scrivo solo le cose che non sapevo]
Le ACL su UNIX hanno valore diverso che sia un file o una directory.
Possiamo immaginare una directory come un file "speciale" che contiene una lista di nome-inode di altri file.

- R = read (lettura del contenuto)
  - Lettura di un file
  - Elenco dei file nella directory
- W = write (modifica del contenuto)
  - Scrittura dentro un file
  - Aggiunta/cancellazione/rinomina di file in una directory
- X = execute
  - Esegui il file come programma
  - Esegui il lookup dell'i-node nella directory (sapendo il nome, scoprire l'inode)

NOTA: il permesso 'W' in una directory consente a un utente di cancellare file sul contenuto dei quali non ha alcun diritto.

NOTA: l'accesso a un file richiede il lookup di tutti gli i-node corrispondenti ai nomi delle directory nel path → serve il permesso 'X' per ognuna, mentre 'R' non è necessario

Oltre a questi 9 bit per le ACL (3 per user, 3 per group, 3 per other), abbiamo anche 4 bit speciali.
Ogni volta che lanciamo un processo, ogni processo ha una quadrupla di identità:

- **real user id** (ruid) = utente che lancia il programma.
- **real group id** (rgid) = gruppo dell'utente che lancia il programma.
- **effective user id** (euid) = identità assunta dal processo per operare come soggetto diverso da ruid.
- **effective group id** (egid) identità di gruppo assunta dal processo per operare come soggetto diverso da rgid.

Normalmente, ruid=euid, egid=rgid, ma se io ho un programma contrassegnato con i bit SUID e SGID, il programma che viene lanciato fa in modo che il SO generi un processo che esegue con l'identità dell'utente che è proprietatrio del file (quidni euid = questo utente).

[altre informazioni molto alla carlona su cose che useremo durante il corso]

Lezioni del 02/03/2022 – Enumerazione, Cracking, Misconfiguration

Enumeration

L'**enumeration** consiste nel cercare più informazioni possibili sul nostro target e quindi mappare la superficie d'attacco del nostro target. Vogliamo quindi *cosa* esporre nel nostro target.
Per fare questo, cerchiamo di capire com'è fatta la nostra struttura del target. Ad esempio, ci chiediamo se una web app è collegata a un database, e se quindi questo db è locale o è remoto/in AWS. Questa fase solitamente è molto lunga, ma è anche la più importante, siccome ci permette di essere più verticali e quindi più specifici nella nostra superficie d'attacco.

Alla parte di enumeration, si collega la parte di misconfiguration, che ci permette capire se abbiamo

lasciato qualcosa di aperto o altro.

Google Dork

Google ha un linguaggio di query complesso che ci permette di fare delle ricerche molto più complesse. Questo viene chiamato **Google Dork**, che fa uso di comandi, anche detti operatori di ricerca, che permetto di trovare dei risultati molto più precisi.

Il Dorking potrebbe essere utile per un pentester nella fase di enumerazione.
In particolare, può aiutare a fornire informazioni approfondite quando si tratta di sicurezza e analisi della sicurezza di una architettura di un applicativo Web. Quindi, in modo legittimo, Google (e in generale tanti motori di ricerca che offrono questo tipo di funzionalità) offre uno dei migliori tool di OSINT a disposizione per "mappare" e/o "enumerare" tutte il possibile riguardo ad un preciso target.

Alcune keyword sono:

- **`cache`**: this dork will show you the cached version of any website, e.g. cache: securitytrails.com [es. una sorta di webarchive tramite Google]
- **`allintext`**: searches for specific text contained on any web page, e.g. allintext: hacking tools
- **`allinurl`**: it can be used to fetch results whose URL contains all the specified characters, e.g: allinurl client area
- **`filetype`**: used to search for any kind of file extensions, for example, if you want to search for jpg files you can use: filetype: jpg
- **`inurl`**: this is exactly the same as allinurl, but it is only useful for one single keyword, e.g. inurl: admin
- **`intitle`**: used to search for various keywords inside the title, for example, intitle:security tools will search for titles beginning with "security" but "tools" can be somewhere else in the page.
- **`intext`**: useful to locate pages that contain certain characters or strings inside their text, e.g. intext:"safe internet"
- **`link`**: will show the list of web pages that have links to the specified URL, e.g. link: microsoft.com
- **`site`**: will show you the full list of all indexed URLs for the specified domain and subdomain, e.g. site:securitytrails.com
- **`*`**: wildcard used to search pages that contain "anything" before your word, e.g. how to * a website, will return "how to…" design/create/hack, etc… "a website".

Alcuni esempi dell'uso:

- Explore LOG Files For Login Credentials – allintext:password filetype:log after:2020
- Dork command using two google operators – allintext:username filetype:log
- Explore Live Cameras – inurl:top.htm inurl:currenttime
- Explore Open FTP Servers – intitle:"index of" inurl:ftp

Possiamo difenderci dall'esposizione di informazioni sensibili grazie al file `robots.txt`. Questo file dice a Google cosa deve o non deve indicizzare del nostro sito.

DNS enumeration

Ora che so l'IP, so più o meno la composizione del target (se usa FTP oppure no etc…), posso adesso provare a sfruttare il DNS.

Il DNS ha degli specifici record che spiegano cosa può fare in base all'azione.
Possiamo usare i DNS records per enumerare delle informazioni importanti relative a un

determinato dominio. Ci sono un sacco di tool che permettono di fare DNS Enumeration, il più famoso è `nslookup` da linea di comando.
Questo ci permette di ricevere delle informazioni importanti sul nostro DNS. Possiamo anche fare un reverse lookup sull'ip.

Analisi dei servizi - Nmap

Dopo che abbiamo esplorato il nostro host, e abbiamo valutato correttamente la sua topologia, possiamo cominciare ad essere più verticali. Dunque, c'è il comando `nmap`, che mappa le porte aperte su un determinato host.

Subdomain Enumeration

Un altra cosa importante è vedere i subdomain di un certo URL.
L'enumerazione dei sottodomini permette di ampliare enormemente la superficie d'attacco durante una campagna di offensive security.
L'output tra i più interessanti da trovare in questa fase sono domini nascosti, bloccati o duplicati "minori". Dietro a questo sottodomini possono infatti nascondersi applicativi o servizi "dimenticati", non aggiornati o non difesi correttamente (si basano su security by obscurity, il peggior tipo di security). Una buona strategia di enumerazione sottodomini è quindi fondamentale.

Un Fully Qualified Domain Name (FQDN) rappresenta un dominio completo per uno specifico host. Un FQDN è ad esempio: miopc.ulisse.com.
miopc quindi è l'host di ulisse.com (subdomain).
Tuttavia:

- https://ulisse.com
- http://myaltropc.ulisse.com
- https://internal.accounts.ulisse.com
- https://internal.accounts.pannelloadmin.ulisse.com

NON sono subdomain di ulisse.com. Sono link a web application (http) hostati sulle porte 80 & 443 dei loro rispettivi host.

C'è però da fare una considerazione:
Prendiamo ad esempio corsosec.ulisse.com. Dietro questo subdomain potrebbe non esserci nessun applicativo web dietro le classiche porte 80 e 443. Questo non significa però che non sia un subdomain valido! L'applicativo web infatti potrebbe infatti essere esposto sulla porta 8080, o su un'altra porta. Oppure dietro a quel dominio potrebbe essere hostato un altro tipo di servizio non necessariamente di tipo web. È quindi buona pratica mappare tutti i subdomain disponibili per un dominio, e risolverli provando a capirne il servizio che c'è dietro.

Ci sono due strategia per l'enumeration:

- **Passiva**: con la strategia Passiva facciamo delle query a dei dataset di DNS noti, che forniscono questi dati, per ottenere tutte le informazioni che cerchiamo.
- **Attiva**: vado direttamente dal nostro target, e testo ogni subdomain possibile. È più rumorosa però, e più invasiva.

Un'altra tecnica per il subdomain enumeration è il 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 certificato viene poi presentato al browser per l'autenticazione del sito a cui l'utente desidera accedere.
Negli ultimi anni, le CA e i certificati emessi si sono rivelati vulnerabili alla compromissione e alla manipolazione, a causa di falle strutturali nel sistema dei certificati HTTPS. Il progetto Certificate Transparency di Google è stato ideato per proteggere il processo di emissione dei certificati offrendo un framework aperto per il monitoraggio e il controllo dei certificati HTTPS.

Un log di Certificate Transparency è un server che implementa RFC 6962 e consente a qualsiasi parte interessata di inviare certificati che sono stati emessi da un'autorità di certificazione pubblicamente attendibile. Una volta che un log accetta un certificato, le proprietà crittografiche del log assicurano che la voce non potrà mai essere rimossa o modificata.

Ciò significa che tutti i certificati emessi dalla CA verrebbero aggiunti a un elenco pubblico comune. Avere un registro trasparente di tutti i certificati emessi è un'ottima soluzione per risolvere il problema dell'emissione fraudolenta di certificati, poiché i legittimi proprietari di domini hanno la possibilità di individuare i certificati emessi senza il loro consenso.

Password Cracking – Hash Crack e Brute Force

Password cracking è un processo di recupero delle password, mediante l'utilizzo di informazioni che sono state trasmesse da un sistema informatico. Un approccio comune (metodo forza bruta, o attacco brute force) è quello di tentare tutte le possibili combinazioni di caratteri, e di confrontarle con un hash crittografico della password. Un altro possibile approccio è il cosidetto **attacco a dizionario**; in questo caso viene generato o recuperato precedentemente un insieme di tutte le possibili soluzioni, da eseguire prima di un attacco a forza bruta. Essenzialmente si prova finché non si ottiene lo stesso hash per la parola e scoprire così la password.

Per fare l'attacco dizionario, le **wordlist** sono importantissime. Sono un insieme di entry che rappresentano le possibili combinazioni che vogliamo testare. Le wordlist più comuni sono quelle composte dalle password più comuni.

Possiamo anche generare personalmente delle wordlist, che sarebbe l'approccio più smart, in modo da creare delle wordlist specifiche per il nostro target.
`Cupp` è un tool che permette di generare automaticamente delle wordlist usando delle informazioni particolare. Il comdano `cupp -i` ci permette di avviarlo.
`cewl` permette di creare delle wordlist basandosi sulle informazioni di un certo sito web. Possiamo usarlo specificando un sito.

Per fare cracking di password, possiamo usare anche **John The Ripper**, che è uno strumento per controllare la sicurezza e per recuperare delle determinate password. Permette anche di fare crack di archivi Rar, Zip e di tante altre cose.

John The Ripper lavora anche senza wordlist, siccome lavora in modo molto intelligente.
Infatti, mi permette di craccare la password usando delle permutazioni basandosi anche solo, per esempio, sul nome dell'utente!
Questa modalità viene detta **Single Crack Mode**: questa modalità fa uso del nome di login, del nome completo, e delle home directory dell'utente come candidati per la password.

Con la suite **`unshadow`** di John The Ripper, possiamo creare un file, basato su passwd e shadow che possa essere letto da John, Il processo in questione serve se gli hash delle password sono memorizzate in shadow.
Il file creato non è molto diverso da i file originali, ma semplicemente aiuta John The Ripper a estrarre gli hash che mi interessa craccare.

Possiamo anche usare una wordlist, usando il flag `--wordlist=PERCORSO_WORDLIST`.
Per craccare un file zip, c'è zip2john, che è un tool presente sempre dentro John The Ripper.

C'è un altro tool simile, anche se John è più ricco, che si chiama HashCat. Questo funziona secondo lo stesso principio di John, ma offre un supporto migliore per la GPU.

Misconfiguration

Per misconfiguration si intende i casi in cui abbiamo un'errata configurazione che permette ad un utente malintenzionato di poter effettuare un attacco con lo scopo di ottenere informazioni sensibili, oppure per ottenere un accesso privilegiato, o ancora di fare DoS (Denial Of Service).

Immaginiamo quindi di avere accesso ad una macchina da utente NON privilegiato. Cosa possiamo fare?

Un primo esempio è quello di guardare il file **sudoers**, che un file/gruppo che decscrive delle azioni privilegiate che può fare l'utente. Se lo modifichiamo, possiamo, per esempio:

- Concedere all'utente www-data di poter modificare i file del sito hostato dal server.
- Concedere un particolare comando di root ad un utente low level.

Il file in particolare è /etc/sudoers.
All'interno di questo file, se inseriamo la direttiva

```
user_test ALL=(root) NOPASSWD: /usr/bin/vi /var/www/html/*
```

In questo modo, concedo all'utente user_test di eseguire come root senza password il comando vi, su qualsiasi file nella cartella var/www/html/ .
In questo modo possiamo sfruttare questa configurazione per:

- Primo: Apriamo una shell da vi sudo /usr/bin/vi /var/www/html/file_a_caso :!bash
- Secondo: Apriamo il file /etc/passwd sudo /usr/bin/vi /var/www/../../../etc/passwd inseriamo una nuova entry (stessa cosa per /etc/shadow).
- Terzo: possiamo modificare uno script che viene esguito da root.

Inoltre, nei sistemi UNIX, la stringa root non è speciale, ma è speciale l'uid=0. Quindi, potre attribuire l'uid=0 a un altro utente del sistema, posso trasformare l'utente in root.

Un altro esempio molto comune sta nel guardare quali file di root hanno il SUID settato. Infatti, se il bit SUID è settato, significa che è possibile eseguire tale binario con i privilegi di root. Se troviamo un binario di questo tipo con una vulnerabilità, possiamo fare privilege escalation anche qui.

Possiamo testare questa cosa andando a settare il bit SUID a cp.

```
chmod u+s /bin/cp
ls -al /bin/cp
```

Noteremo quindi che:

```
-rwsr-xr-x 1 root root 146880 Feb 28 2019 /bin/cp
```

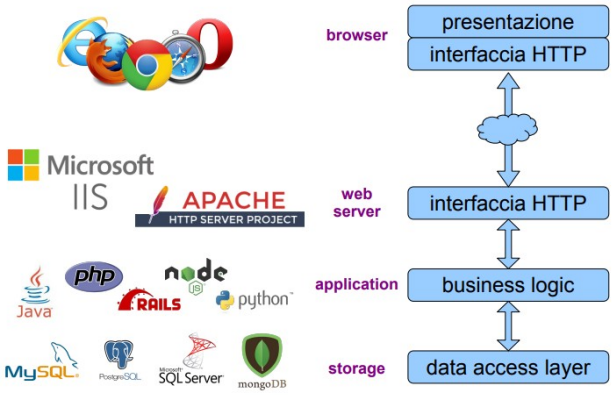
Notoremo che ora il binario è colorato di rosso. Ciò indica questo ha il bit SUID settato.

Possiamo usare questa vulnerabiltà, per esempio, per prendere il file `/etc/passwd`, copiarlo in una cartella a caso, modficarlo aggiungendo un utente, e poi copiarlo nuovamente in `/etc/passwd`.

Posso usare il comando `find / -perm /6000`, per listare tutti i binari/file con il bit SUID settato. Questo perché andiamo a richiedere i tutti file aventi i bit SGID o SUID (la stringa indica quello).

Lezioni del 04/03/2022 – Sicurezza e vulnerabilità nelle applicazioni web

Com’è fatta un’applicazione web?



Le vulnearbilità, in questo scenario, si possono trovare:

- Sul **lato del client**: potremmo avere
  - errori di definizione del perimetro di interazione con il server
  - Esecuzione di codice maligno sul client
  - Manipolazione del DOM per ingannare l’utente.
- Sul **lato del protocollo**:
  - potremmo viaggiare su un canale non sicuro, per esempio nel caso non sia stato implementato un TLS. [NOTA: HTTP, in sé, non possiede alcuna vulnerabilità intrinseca, ed è anche dovuto dal fatto che il protocollo è molto semplice]
  - Potremmo avere problemi causati da cookie o dalla serializzazione, che però potremmo attribuire a un trattamento errato di questi sull’endpoint (client o server che sia).
- Dal **lato server**, possiamo avere (il gran numero di possibili vulnerabilità deriva anche dal fatto che le applicazioni sul server sono decisamente più complesse):
  - Errori di gestione di richieste HTTP (smuggling)
  - Errori di controllo dell’accesso (IDOR, FD, CSRF)
  - Errori di interpretazione dei dati (XSS, SSRF, XXE, injection, deserialization)
  - Carenza di aggiornamento ed errata configurazione software

OWASP è una delle associazioni più autorevoli sulla sicurezza delle applicazioni web.
Ogni 4 anni rilasciano una Top Ten sulle vulnerabilità di maggiori impatto che si potrebbero trovare all’interno di un applicazione web. Quello che vedremo nella lezione di oggi è essenzialmente ogni singola vulnearbilità della Top Ten del 2020.

#1 – Broken Access Control

Questa vulnerabilità raggruppa varie cause d’accesso non correttamente mediato alle risorse, ovvero che nel mio protocollo d’accesso degli utenti possono accedere ad oggetti/pagine che non dovrebbero vedere.
Dei problemi di accesso sulle web app si potrebbero avere, per esempio, se vengono esposti degli identificativi di oggetti nell’url. Normalmente questo avviene perché si pensa che l’id di un oggetto sia troppo lungo per un utente da indovinare (facendo in sostanza security through obscurity). Gli attacchi di forza bruta funzionano molto bene con questo tipo di cose.
La cosa è ancora è più grave quando queste funzioni sono accessibili dall’utente cambiando un qualche parametro nel link.

Il nome tecnico di ciò che abbiamo appena descritto è **IDOR** (Insecure Direct Object Reference), secondo cui un oggetto viene erogato semplicemente perché si sa come si chiama.

Per mitigare queste cazzate che fanno gli scemi, è necessario:

- non esporre mai dati direttamente dal server web
- eventualmente, per alleggerire, creare mappature effimere e con id non prevedibili (hash)
- usare funzioni che implementino AAA ad ogni richiesta.

```
https://www.example.com/login.php
  – login come user, redirect a
https://www.example.com/userapp.php
  – riscrittura a mano
https://www.example.com/adminapp.php
  – funzionalità di amministrazione!
```

```
https://www.example.com/fileop.php?
f=a.txt&action=backup

https://www.example.com/fileop.php?
f=a.txt&action=delete
```

Esempi di IDOR

Un altro esempio di Broken access control è il **File Disclosure (FD)**.
Questo si trova essenzialmente a metà strada tra Injection e Broken access control. Possiamo vederlo come un caso particolare di IDOR in cui l'oggetto è un elemento del filesystem; caso classico: ***path traversal***.
L'esempio classico è quello della lezione di ieri, in cui essenzialmente se ho dei caratteri/operazioni usate per risalire la gerarchia nel FS:

```
doc = ../../../../etc/passwd
                ⇓
cat /home/maybe/deep/username/../../../../etc/passwd
```

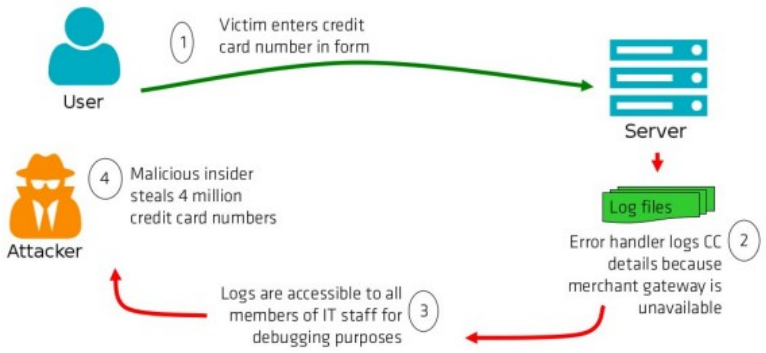
Essenzialmente, raggiungo il documento che voglio trovare attraversando il path.

#2 - Cryptographic failures (chiamato Sensitive data exposure in precedenza)

Le web app manipolano grandi quantità e varietà di dati sensibili. Questi dati vanno protetti sia quando sono stabili su una memoria (*at rest*), sia quando sono trasmessi a un servizio (*in transit*).
Sorgono problemi quando:

- un dato non è riconosciuto come sensibile (se progettiamo un sistema che tratta/invia dei dati sensibili come dati non sensibili, allora abbiamo commesso un peccato originale)
- non si individuano tutte le copie del dato da proteggere (per esempio i DRM, Netflix impedisce al browser di salvare il video. Oppure quando c'è caching).
- non si proteggono tutte le fasi di vita di un dato sensibile (es. database o filesystem con cifratura trasparente = protezione *at rest*, che viene automaticamente rimossa prima di mandare il dato in rete)
- non si utilizzano metodi di protezione adeguatamente robusti.

Es. canale correttamente cifrato e salvataggio in database non previsto, ma dato copiato "incidentalmente"su altro file non adeguatamente protetto



Un utente manda una copia della sua carta di credito, e il canale è prottetto in modo corretto e il database è cifrato. Tuttavia, il web server della webapp preserva i log di errore nel database in maniera chiara.
Potrebbe succedere che magari c'è un errore tra il server e il gateway della banca, e nel log vengono salvate le informazioni del form.

Un dipende che può accedere a questi file di log, allora, può accedere a tutte queste carte di credito.

Oltre a ciò, un attaccante potebbe fare un DoS al gateway server, pigliandosi le carte di chi vuole.

#3 – Injection

Con questo attacco, invio input non fidati a un interprete. Questi input provocano l'esecuzione di comandi, e a loro volta delle di uno dei pilastri della sicurezza (C/I/A).
Uno scenario possibile potrebbe essere questo:

- il server utilizza applicazioni esterne e le alimenta con parametri ricevuti via HTTP (normalmente inseriti dall'utente in form HTML)

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

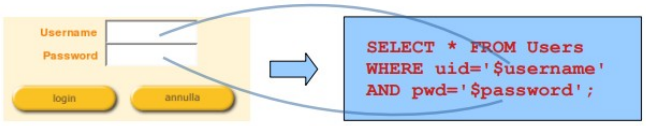
- il server prepara il codice da far eseguire all'applicazione esterna componendo gli elementi statici coi parametri

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

- l'interprete esterno esegue il comando ricevuto

```
name = ; cat /etc/passwd
ls -l /home/; cat /etc/passwd
```

Un altro esempio molto diffuso di iniezione è la **SQL Injection**, che al posto di usare comandi bash fa uso di comandi SQL.
Anche SQL è un linguaggio interpretato, e come tale, presenta delle keyword/simboli speciali che hanno un significato diverso da quello letterale. Inoltre, il carattere non viene identificato come speciale a priori, ma solamente una volta che questa "frase" viene interpretata. Dunque lo sviluppatore non saprà che cosa l'interprete si troverà a processare.




```
user: *        password: ' OR 'a'='a
⇒ query:       SELECT * FROM Users
               WHERE uid=* AND pwd='' OR 'a'='a';

user:          password: '; DROP DATABASE WebApp; --
⇒ query:       SELECT * FROM Users WHERE uid='' AND pwd='';
               DROP DATABASE WebApp; --';
```

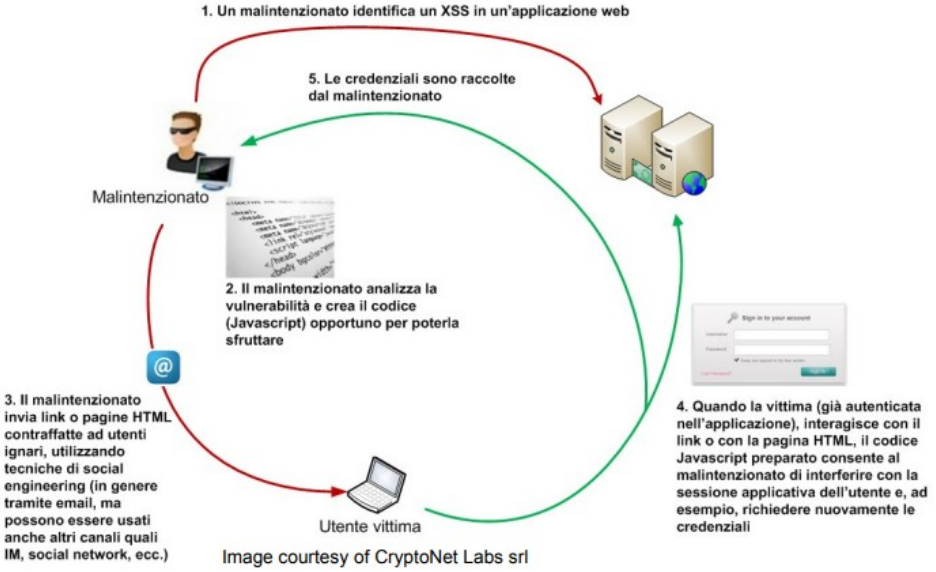
nota: i due trattini in SQL indicano un commento

Ho confuso così il mio interprete, facendogli interpretare come dei comandi dei campi che erano riservati solo ai dati.

Un altro tipo di injection è la **XSS** (**Cross-Site Scripting**). È un iniezione di codice lato browser, e può assumere una molteplicità di varianti.
Nel caso in cui l'XSS sia reso possibile da un'applicazione vulnerabile su di un sito di cui l'utente si fida, si parla di Stored XSS e Reflected XSS.
Si parla invece di DOM XSS quando un'applicazione vulnerabile è in esecuzione sul browser.

In entrambi i casi però, la vulnearbilità sta da "un'altra parte", ovvero o sul server o sull'applicazione che è caricata dal browser che (che però proviene da un server), e nonostante ciò la vittima è l'utente.

Ecco un esempio di reflected XSS:



Il malintenzionato scrive un malvìware che sfrutta una vulnerabilità XSS. Quando la vittima (che è già stata autenticata nell'applicazione) interagisce col link o con la pagina HTML, programma JS scritto dal malintenzionato permette di interferire con la sessione applicativa dell'utente, e ad esempio richiedere nuovamente le credenziali. Queste poi vengono così raccolte dal malintenzionato.

Ecco un esempio pratico della cosa, completo di codice e tutto:

- Una URL con uso apparentemente lecito di un parametro per personalizzare una pagina di benvenuto

```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=John
```

- Il codice vulnerabile lato server (non sanifica il parametro)

```
<?php
    echo 'Welcome to our site ' . stripslashes($_GET['name']);
?>
```

- L'URL preparata dall'attaccante, su cui l'utente viene portato a cliccare

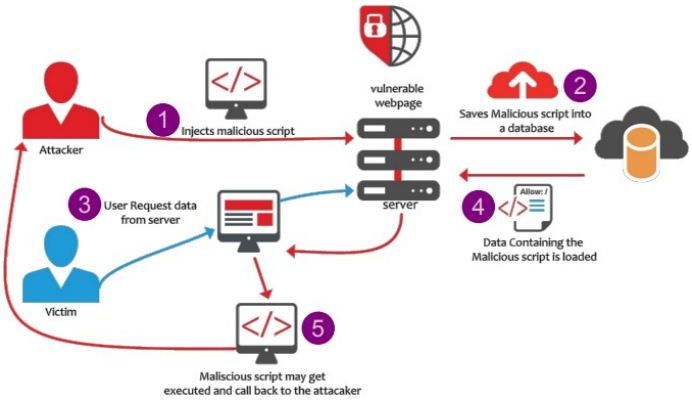
```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=
<script language=javascript>alert('Hijacked!');</script>
```

- Il codice HTML ricevuto dal browser

```
Welcome to our site
<script language=javascript>alert('Hijacked!');</script>
```

Un **DOM XSS** è molto simile al reflected, ma in questo caso il codice che processa dati in modo non sicuro non è sul server, ma già nella pagina, e il codice ha accesso a una grande quantità di informazioni prelevate dal Document Object Model.

Nel **XSS Stored**, l'attaccante inietta uno script direttamente sull'applicazione web conservata nel Server, e la vulnearbilità della pagina sta nel fatto che del contenuto che non dovrebbe essere salvato viene salvato sul database. Quando la vittima fa una query al server, i dati che contengono anche uno script malevolo vengono mandati alla vittima.



Come fare pentesting per controllare la presenza di XSS?
In quel caso potremmo trovarci in due situazioni:

- White Box: si dispone del codice e lo si analizza staticamente. Questo è anche il motivo per cui molte applicazioni open source sono diventate molto solide dal lato della sicurezza negli anni.
- Black Box: si stimola l'applicazione con input ragionati (***fuzzing***)
  - caso semplice: output = dati
  - caso complesso: output = generica segnalazione di successo o di errore (**blind injection**).

Nel caso del Blind injection, per estrarre le informazioni possiamo procedere in vari modi:

- si tenta di iniettare un'operazione lenta (es. sleep) per capire dal tempo di risposta se è stata eseguita
- si osservano le differenze nelle pagine di errore
- si tenta di iniettare un'operazione di pingback.

Gli attacchi di injection possono essere mitigati o con la sanificazione dell'input, oppure con la precompilazione SQL. Altre raccomandazioni sono il binding delle variabili e dare il minimo privilegio agli interpreti.

#4 – Insecure Design

Bisogna fare molta leva sul fatto di sviluppare applicazioni che fin da subito prendono in considerazione la sicurezza. Non possiamo pensare, infatti, di costruire delle applicazioni che sono prodotte fin dall'inizio senza la sicurezza in mente: non possiamo aggiungere la sicurezza DOPO aver sviluppato il prodotto.

Per fare ciò, possiamo fare uso di :

- threat modeling (schematizzare minacce e potenziali vulnerabilità)
- secure design patterns
- reference architectures

In tutti gli ambienti software (e non solo quelli di sicurezza) ad oggi viene introdotto un concetto che prende il nome di paradigma **shift left**.
Con shift left si intende che, se noi prendiamo in considerazione la pipeline SDLC (Software Development Lifecycle) identiamo delle fasi che sono andate da sinistra verso destra. La sicurezza veniva aggiunta solo dopo, come un ripensamento alla fine di aver distribuito il proprio prodotto. Adesso, invece, l'aspetto di sicurezza viene spostato sempre più "a sinistra" (ovvero all'inizio dello sviluppo dell'applicazione).

#5 - Security misconfiguration

Questo aspetto rappresenta una categoria molto ampia che raccoglie degli errori di configurazione a tutti i livelli dello stack. Un esempio sono è l'uso di credenziali di default per DB e server, oppure il non minimo privilegio. Questo vale per server, dispositivi hardware oltre che alle webapp.

Un altro esempio è a livello del protocollo: si potrebbero scegliere dei cifrari deboli per TLS o obsoleti.

Dal lato del client, invece, dobbiamo impostare gli header dal lato server per garantire il corretto comportamento del client. La compatibilità con i diversi browser è molto variegata.
Alcuni di questi header sono:

- `X-Frame-Options`:
  - Controlla l'embedding di risorse con `<frame>`, `<iframe>`, `<object>`
  - `DENY, SAMEORIGIN, ALLOW-FROM <url>`
- `X-XSS-Protection`:
  - filtra i tentativi di reflected Cross Site Scripting (v. A3)
  - `1 ; mode=block`
- `Content-Security-Policy:`
  - previene il caricamento di risorse (es. js) da origini esterne
  - `default-src 'self'`

Nel World wide web, abbiamo una collocazione distribuita di risorse condivise da siti. Il loro riutilizzo è utile e sensato, ma fonte di opportunità di attacco, dalla quale dobbiamo difenderci.
Una soluzione drastica di difesa sta nella **Same Origin Policy (SOP)**: pagine e script caricati da un'origine possono unicamente accedere a risorse provenienti dalla stessa origine. Ad es. uno script iniettato dal dominio A non può leggere i cookie settati dall'interazione col dominio B.
Normalmente, però, si usa il **Cross-Origin Resource Sharing (CORS)**, ovvero un meccanismo per rilassare in modo controllato i vincoli SOP e definire in modo granulare le eccezioni.

Security misconfiguration – XXE (prima a sé stante come XML External Entities)

XML permette di fare riferimento a dati esterni: i riferimenti vengono de-referenziati su indicazione del DTD.
Inviare un XML malevolo a un parser "disattento" può consentire la lettura di file dal server, attacchi Server Side Request Forgery, invio di dati a sistemi controllati dall'attaccante, attacchi blind che esfiltrano dati dai messaggi d'errore e altro che onestamente non ce ne frega più di tanto.

Ecco un esempio:

- **Applicazione per richiedere il numero di pezzi disponibili**

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

- **XML predisposto per attacco XXE**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

- **Risposta:**

```
Invalid product ID: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
```

Essenzialmente questa è una iniezione dovuta a una configurazione fatta male del server.

Gli attachi dovuti a XXE possono essere:

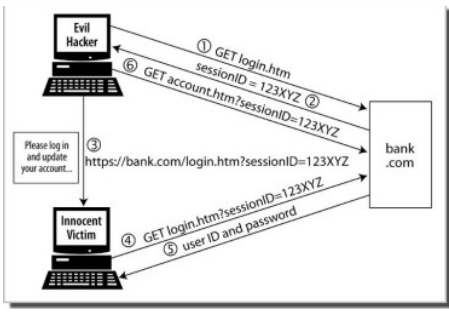
- SYSTEM permette di includere qualsiasi URI
  - richieste verso l'esterno: SSRF
  - richieste verso servizi interni normalmente non accessibili: RCE
- DoS
  - espansione esponenziale di entità: **Billion Laughs Attack** (si scrive un comando XML "ricorsivo" per cui l'esponsione viene espansa in un altra richiesta di espandere il documento)

Possiamo mitigare questo tipo di attacchi usando formati più semplici (come JSON), oppure usare parser aggiornati, disattivare il supporto alle external entities (almeno per i DTD) e infine, ovviamente, sanitizzare l'input consentendo solo gli elementi necessari. Buona pratica è anche installare nei server solo ciò che server.

#7 – Identification and Authentication Failures
Fortunatamente, questo è in forte calo grazie all'uso più diffuso di framework, i casi residui sono principalmente dovuti a **identificazione**.
Le applicazioni web fanno largo uso di sessioni (nonostante HTTP sia un protocollo stateless): dopo l'autenticazione, assegnazione di un segreto (token) che identifica la sessione, e poi vi è uno scambio trasparente all'utente del 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 in diversi metodi (es. attraverso l'anticipazione dell'attribuzione del token, oppure perché l'identificativo prevedibile).

Un esempio è il **session fixation**: l'attaccante avvia la sessione in modo da farsi consegnare un token (a questo punto la sessione non è ancora autenticata), e convince la vittima a proseguire la sessione con l'autenticazione.

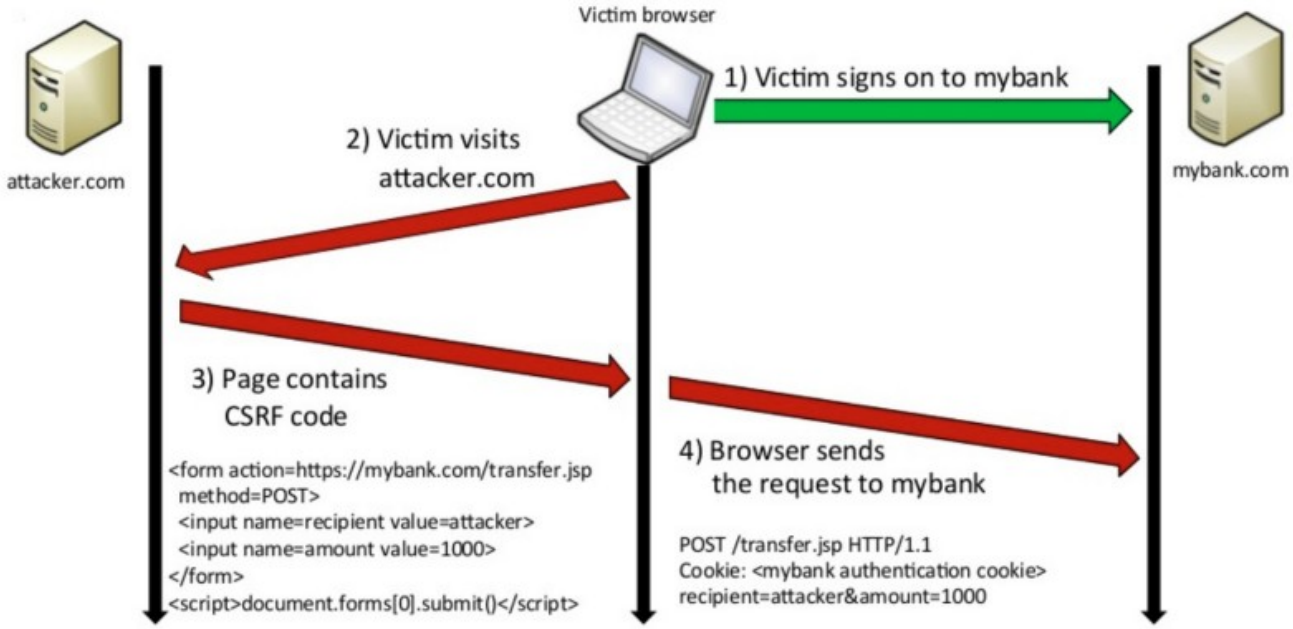


In generale ci possono essere problemi legati all'autenticazione quando:

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

- l’id non viene invalidato al termine della sessione

Un altro esempio di errore di validazione è il Cross-Site Request Forgery.
Noi sappiamo che il browser include automaticamente tutti i token identificativi di una sessione in ogni richiesta inviata al server.
Dunque, se l’host della vittima entra prima in mybank.com e riceve il token di accesso/sessione dal sito.
Dopodiché, entra nel sito dell’attaccante che contiene del codice per fare un attacco CSRF. Per esempio, potrebbe contenere un form nascosto che viene attivato subito al momento dell’accesso al browser, e quindi questo codice gira sul browser della vittima e sfrutta l’access token per fare la richiesta maligna a mybank.



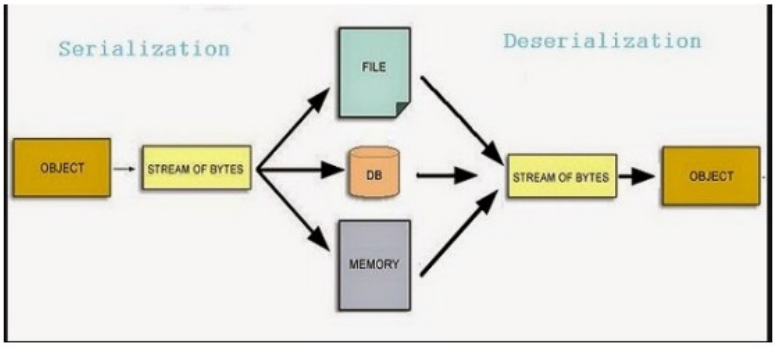
Possibili mitigazioni per Cross-Site Request Forgery sono:
- Il server legittimo dovrebbe includere un segreto in ogni interazione sensibile che sia univoco, non falsificabile e non facilmente intercettabile (URL) (es. in ogni form si include qualcosa come `<input type=hidden value=23a3af01b>`, il server dovrebbe validare i Referer o richiedere l’utilizzo di header HTTP custom).

#8 - Software and Data Integrity Failures

Sono delle vulnerabilità risultanti da insufficiente tutela dell’integrità dei dati, del codice e dell’infrastruttura. Derivano per esempio da:
- uso di plugin da fonti esterne non verificate
- uso di reti di raccolta/distribuzione dati con politiche di tutela dell’affidabilità e dell’autenticità non all’altezza di quelle richieste dal sistema
- uso di processi di sviluppo e dispiego automatizzati che consentono più facilmente di mascherare iniezioni di codice malevolo e configurazioni alterate

Questa vulnerabilità è molto comune: livelli di autenticazione ridotti in fase di aggiornamento rispetto all’installazione.

La **serializzazione** è un 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 utilizzare codifiche testuali (ad esempio XML).
Alla ricezione si può ricreare l'oggetto di partenza (deserializzazione).



Nel caso della serializzazione, l'applicazione potrebbe essere vulnerabile se:

- l'oggetto serializzato viene passato in chiaro via HTTP o memorizzato senza un corretto controllo dell'accesso.
- il de-serializzatore non controlla l'integrità dello stream ricevuto.

Questa vulnerabilità può portare al passaggio di oggetti Java o .NET, e quindi esecuzione di codice arbitrario, oppure al passaggio di cookie con ruolo dell'utente autenticato, che possono portare a privilege escalation.
Possiamo mitigare questa vulnerabilità:

- non deserializzare dati non fidati… se fosse sempre possibile
- usare solo librerie con controllo stretto dei tipi e non lasciare mai che il tipo dell'oggetto venga determinato analizzando lo stream stesso
- monitoraggio (rilevazione fallimenti da brute force)
- sandboxing

[NOTA: queste ultime due le ha fatte un po' alla cazzo]

#9 – Security Logging and Monitoring Failures

Questa non una vulnerabilità di per sé, ma un errore che facilita il lavoro dell'attaccante.
Alcuni di questi errori comuni sono:

- non tracciare accessi falliti, transazioni terminate con errori, invocazioni di 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

E di conseguenza alcuni degli effetti comuni sono:

- non si rilevano tentativi di forza bruta
- non si riesce a ricostruire la dinamica di un attacco per mitigare la vulnerabilità utilizzata
- non si riesce a riparare il danno subito, in parte o del tutto

#10 - Server-Side Request Forgery (new)

Un'applicazione web lato server riceve dall'utente una URL e non esegue verifiche adeguate.
Questo URL viene usato 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.

Lezioni del 09/03/2022 – Esercitazione su WebApp security

Esercitazione pentesting

Abbiamo scaricato una repo che contiene un file .sh che ci permette di creare dei container docker contenenti delle applicazioni vulnerabili. Ovviamente, quando lanciamo una di queste applicazioni, dobbiamo assicurarci essenzialmente che non ci sia un'altra applicazione che sta girando sulla porta 80 allo stesso momento, per impedire di creare conflitti.

Possiamo usare il comando `ss -tulpn` per vedere se c'è un'applicazione in esecuzione che sta eseguendo codice in quella porta. Potremmo trovare per esempio il server nginx che sta girando nella nostra macchina, e dovremo stopparlo con `sudo service stop nginx`.

A questo punto possiamo iniziare ad eseguire una delle nostre app. Usiamo il comando `./pentestlab.sh start dvwa` per lanciare l'applicazione `dvwa` avente la vulnearbilità. L'applicazione aggiunge anche un'entry nel file /etc/hosts in modo che possiamo accedere a localhost travmite il sito `http://dvwa`.

A questo punto, possiamo iniziare a guardare l'applicativo ed ad enumerarlo. La prima cosa che possiamo fare è scannerizzare le porte dell'host. In questo modo possiamo effettivamente capire se la nostra app giri effettivamente sulla porta 80 (e non su un'altra porta) e se per esempio abbiamo un database esposto. Come sappiamo, possiamo usare il comando `nmap` per capire esattamente questo.

Web Content Scanner

Un'altra cosa molto interessante che possiamo fare è questa: fare **web content scanner**. Infatti, una volta che abbiamo scoperto l'indirizzo IP della nostra macchina target e quale servizio espone, dietro quale porta, è possibile provare a fare uno scan dei contenuti. Magari un utente scemo ha messo un file secret.txt nella home del sito o altre cose sceme...
Alcuni dei file che potrebbero interessare sono:

- `robots.txt`, che come sappiamo è interessante.
- `wp-config.php`, se il nostro sito lavora su wordpress
- .git

Possiamo usare dei programmi fatti apposta per controllare la presenza di queste informazioni sensibili. Il tool che usiamo si chiama `gobuster`. Questo tool fa uso di una wordlist per confrontare e controllare la presenza di questi punti di accesso potenzialmente interessanti. Nella nostra VM, possiamo trovarla su `SecLists/Discovery/Web-Content` .

Non ci saranno esercizi all'esame su brute force!

Uno strumento molto utile è anche Burp.
Normalmente, con i binari, quello che vogliamo fare per capire se è un app è vulnerabile è capire dove vanno a finire le nostre richieste. In particolare, vogiamo vedere come cambia la memoria dopo che abbiamo fatto le nostre richieste.
Con una WebApp, la cosa è simile, ma ci basiamo su delle richieste HTTP (che è ciò su cui si basa l'applicazione). Quello che ci serve è quindi la possibilità di interecettare la richiesta e capire effettivamente cosa viene mandato al server. Questo è il concetto di web proxy, che si interpone fra client e server. **`Burp` fa esattamente questo**.

File Inclusion (FI)

La file inclusion è un'altro tipo di vulnerabilità che permette di includere nella richiesta che

facciamo ad una particolare webapp un'altra risorsa. Questa risorsa, ovviamente, non dovrebbe essere invocabile.
Un esempio potrebbe essere il fatto che abbiamo una webapp che permette di fare una richiesta per un file (es. php). In questo modo possiamo dirgli che non vogliamo solo il file che ci vuole inviare, ma anche /etc/passwd tipo.
Al contrario, posso inserire io un file esterno all'interno del server con questa vulnearbilità. In sostanza, la vulnerabilità ci permette di accedere o inserire dei file alterando la logica della webapp.
Ci sono due tipi di file inclusion (che sono gli stessi che abbiamo visto):

- local: prendo file locali dal server.
- remote: inserisco dei file sul server che non dovrebbero essere inseriti

Possiamo vedere come funziona il FI se esaminiamo la nostra dvwa. Infatti, possiamo vedere cosa succede quando richiediamo un file esaminando la source della pagina usando il comando **View Source**. Nel caso di dvwa, fa una semplice GET al server con php per prendere quel file, senza fare controlli per backslash o altro.
Dunque, possiamo facilmente prendere il file /etc/passwd in modo semplice e veloce. Lol.

Se mettiamo il livello di sicurezza a medium, allora la nostra webapp adesso ha attivato un filtro per l'url. In questo modo non è più facile come prima, ma possiamo benissimo provare lo stesso a prendere la risorsa. Tuttavia, possiamo benissimo non scalare la gerarchia delle cartelle e prendere direttamente il file passando per la root del FS.
L'approccio a filtri è sempre sbagliato, infatti potevo benissimo usare //etc/passwd se /etc era filtrato. Il filtro http invece poteva essere bypassato usando HTTP (ovvero le lettere maiuscole).

Questa tenica di scalare le cartelle per prendere il file è un esempio di **Local File Inclusion** (LFI).

Command Injection

Questa vulnerabilità consiste nel fatto che essenzialmente viene eseguito un comando dato in input ad un'applicazione web sul server dell'applicazione in cui gira l'applicazione web.

Se andiamo nella sezione command injection di dwva, possiamo notare che ciò che esegue lo script php è in sostanza una exec del comando ping + testo dato in input, *senza filtri*.
Quindi, possiamo usare il **;** per eseguire un qualsiasi altro comando oltre al ping, oppure i && per eseguirli entrambi. L'app adesso è letteralmente totalmente nostra.

Adesso, mettiamo la sicurezza dell'app a medium, possiamo usare un sacco di altri metodi per eseguire il nostro comando. Ad esempio, possiamo usare i simboli ||, |, & e molto altro...

Mettendo la difficoltà su hard, notiamo che ci sono molti filtri. Ma comuqnue, possiamo usare la pipe ('|') scrivendo il nostro comando in modo "attaccato".

SQL Injection

Si inserise una query SQL tra i dati di input che vengono consegnati all'applicazione.
Quello che succede è che ci sono dei sync (che sono delle chiamate tipo system, exec etc) e prima di questi input c'è un sistema di input che ci permette di sanificare l'input ed evitare un iniezione di codice.
L'SQL Injection avviene perché abbiamo un input che viene passato direttamente all'SQL Engine del database, senza che venga sanificato.

Ci sono SQL injections che sono semplici, e che ci permettono di alterare la logica della query e basta per accedere a dei dati ai quali normalmente non potremmo accedere (Ad esempio se esiste un utente con una data password, oppure se esiste un utente con particolari permessi etc..).

Inoltre, con un SQL Injection, posso cambiare totalmente la logica dell'applicazione, e fare in modo che questa mi ritorni true a prescidere da ciò che succede.

Se noi mettiamo in un campo di input una stringa formata da un numero/carattere a caso e un apice, e otteniamo un messaggio di errore di SQL, ql 99% abbiamo la possibilità di SQL injection.

Union Based SQL Injection

Il concetto delle union based SQL è il fatto che possiamo usare la keyword UNION per avere delle informazioni che si possono recepire con un'altra query SQL. (es. SELECT * FROM USERS).

Tuttavia non è così semplice: in SQL le UNION richiedono che il numero delle query della prima query combaci con il numero delle colonne della seconda query.
Per farlo, possiamo usare una combinazione di SELECT NULL, mettendo tanti NULL quanti pensiamo che ce ne sia nel nostro schema del DB. Dopodiché, usiamo il comando # per dire dare una select unica, tipo
SELECT NULL, NULL, NULL #
in questo modo non dobbiamo inserire FROM.

Il cancelletto serve per commentare il resto della query

Dobbiamo però capire su quale tabella dobbiamo fare questa select, e per capirlo noi dobbiamo usare delle primitive SQL che ci ritornano il nome e la struttura di alcune tabella nel DB.
Ci sono implementazioni di SQL che contengono il nome dello schema come colonna del DB sotto il campo `schema_name`.
La query che dovremo usare in questo caso sarà fatta all'`information_schema`, altro non sono che tabelle che contengono i nomi di tutta la struttura del database, facendo le query su queste tabelle scopriamo tutte le info necessarie.
Query finale:

```
'union select null,schema_name from information_schema.schemata #
```

Ma questo potrebbe non bastarci. Infatti, può darsi che le informazioni che stiamo cercando si trovino invece in un'altra tabella. Dunque, dobbiamo usare un'altra query diversa:

```
' union select null,table_name from information_schema.tables #
```

In questo modo possiamo enumerare le tabelle.

Se vogliamo sapere le colonne delle nostre tabelle, possiamo ancora ricorrere all'information schema, e in questo modo la query diventa:

```
' union select null,column_name from information_schema.columns
where table_name ='users' #
```

Se mettiamo la difficoltà a medium, notiamo che adesso non abbiamo più un input, bensì abbiamo una select. Dunque, possiamo usare Burp per modificare la richiesta e inviare comunque una query alla nostra webapp. [DISCLAIMER: In questo caso, siccome si usa l'encoding http, dobbiamo usare al posto degli spazi l'espressione %20, che è il carattere ``` nell'encoding http.]
Alternativamente, se al posto di Burp usiamo lo strumento ispeziona di Firefox, possiamo riprodurre la vulnerabilità senza preoccuparci degli encoding.

XSS (Cross Site Scripting)

Ultima classe di vulenrabilità è client-side. Ovvero, il codice malevolo viene eseguito lato client sulla macchina della vittima, e non come quelli di prima che invece erano mirati al server (e che si basavano sul codice eseguito sul server).

Questo tipo di attacchi sono mirati a rubare dati sensibili della vittima, lato client. In particolare, siccome le web app eseguono codice JS lato client, sono nate un gran numero di vulnerabilità.

La vulnerabilità di maggiore importanza è il **XSS** (Cross Site Scripting).
Questa vulnerabilità permette agli attaccanti di inserire il proprio lato client codice (normalmente Javascript) in siti web o applicazioni web.

Una volta ricevuto questo codice dannoso insieme alle pagine Web originali visualizzati nel client Web, agli aggressori ottengono delle informazioni o eseguire tutto ciò che è in scope da javascript sul browser della vittima.

In questo modo, possiamo, per esempio:

- Rubare cookie
- Rubare informazioni specifiche sensibili
- Far compiere delle azioni per conto dell'user della webapp
- aggiungere banner pubblicitari.

Possiamo testare un metodo di XSS andando sulla nostra webapp (dopo aver settato il livello di vulenrabilità a basso) e aggiungendo il tag <script>alert("XSS")</script> nell'input.

Per difenderci da XSS, dobbiamo essere molto più raffinati rispetto alle SQL Injection, siccome ogni browser interpreta Javascript in modo diverso. Ci sono davvero tante cose fighe:

Esempi:

- https://github.com/payloadbox/xss-payload-list
- https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

Se mettiamo il livello medio dvwa, possiamo usare il tag script in questi modi per permettere XSS:

- <SCRIPT>
- <script >
- <scriPt >
- <script src="">
- una delle tecniche di payload viste nei link sopra...

Lezioni del 11/03/2022 – Binary Exploits

Attacchi applicativi – exploit binari

Gli attacchi applicativi sfruttano le vulnerabilità del Sistema Operativo, dell'Hardware sottostante e del software in esecuzione locale. Non sono quindi coinvolti livelli dello stack di rete che si trovano al di sotto del livello applicativo, e in generale non è di nostro interesse il router e l'infrastruttura di rete per questa parte. Si ha un focus nel sistema operativo Linux e sull'architettura intel IA32 (ovvero x86 a 32 bit).

Lo scopo di un exploit è di far eseguire delle oeprazioni ad un processo per cui non era stat pensato. Se pensiamo l'exploit come un attacco, allora questo può avere 3 obiettivi:

- Fermare il processo (Denial Of Service – DOS) – che potrebbe essere il primo step per un attacco più complesso. Dunque...
- … lo si potrebbe anche fare per dirottare il flusso di esecuzione (Esecuzione di codice maligno)
- Ottenere i privilegi di altri utenti (e privilege escalation)

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

La comprensione dei meccanismi alla base delle tecniche di Exploit necessita di una conoscenza basilare di:

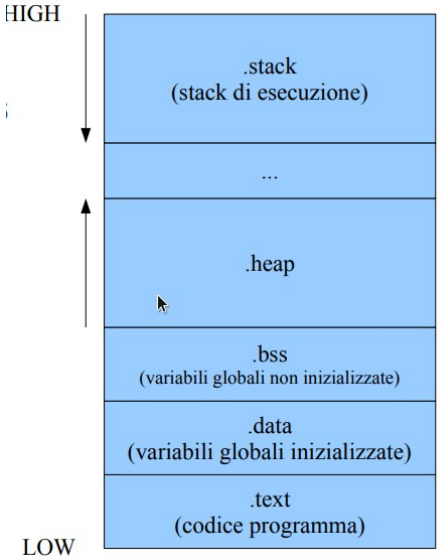
- Disposizione in memoria di un processo
- Funzionamento dell'architettura del processore su cui lavora il sistema vittima (nel nostro caso IA32)

Si noti che la maggior parte delle vulnerabilità sono relative a codice scritto in C/C++ e linguaggi derivati. Infatti tali linguaggi, più vicini all'hardware, non realizzano controlli (in automatico) sui dati. Linguaggi di più alto livello (come ADA, ad es.) in fase di compilazione aggiungono controlli ad ogni istruzione/gruppi di istruzioni.

Processi in Memoria

Nel sistema operativo Linux, lo spazio di indirizzamento di un processo in memoria è ***suddiviso in segmenti***:

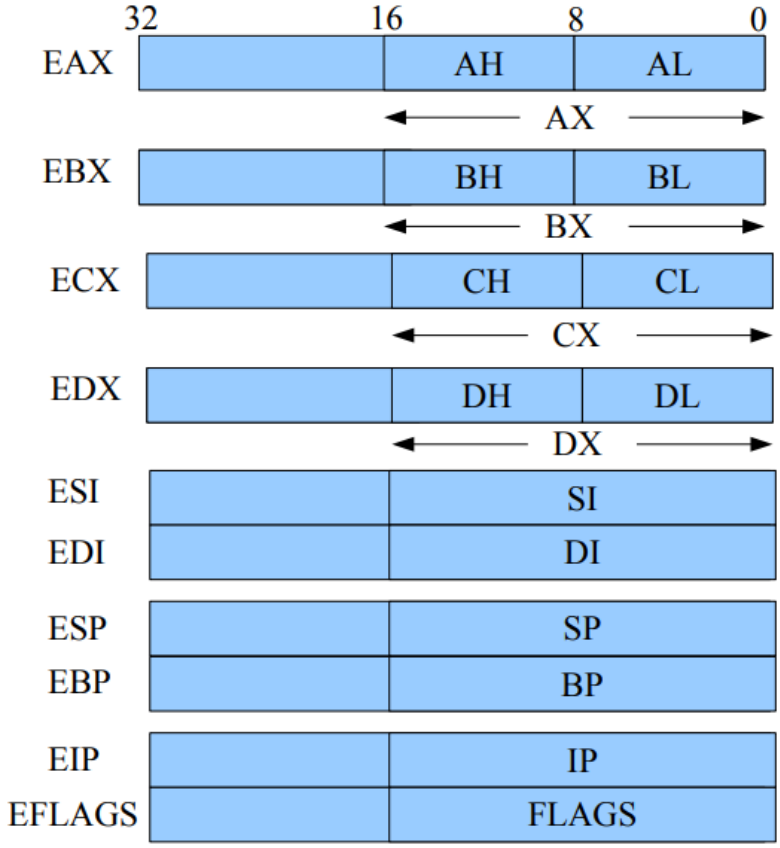
- Segmento `.text`, che contiene il codice eseguibile
- Segmento `.data`, **contenente i dati inizializzati** (variabili statiche inizializzate)
- Segmento `.bss`, contenente le variabili non inizializzate
- Stack d'esecuzione, con i record d'attivazione del processo e le variabili locali
- Heap, segmento di memoria contenete le variabili dinamiche (può crescere dinamicamente)
- Altri segmenti necessari al funzionamento (.got, .ctor, .dtor)



Quando viene scritto un programma, il programmatore può vedere solo gli indirizza virtuali. Questi sono poi tradotti dall'OS (+HW, come MMU) in indirizzi fisici. Questi segmenti potrebbero anche essere non necessariamente contigui.
Mentre lo stack cresce verso il basso, l'heap cresce verso l'alto.

Cenni sull’architettura IA32

L'architettura IA32 è dotata di 4 registri 32 bit “general purpose”: EAX, EBX, ECX, EDX.



Due registri sono usati per le operazioni di copia dati in memoria:

- ESI: source
- **EDI**: destination (INDIRIZZO DI RITORNO)

Due registri hanno ruoli speciali per il controllo di flusso:

- EIP: Instruction Ptr
- EFLAGS: Status register

Due registri hanno ruoli importanti per la gestione dello stack:

- ESP: stack pointer – punta all’ultima cella occupata dello stack.
  Ci sono poi delle operazioni 2 operazioni che possiamo eseguire sullo stack:
  - PUSH decrementa ESP di 4 (byte) e scrive il valore sulla cella puntata
  - POP recupera il valore dalla cella puntata da ESP e poi incrementa ESP di 4

- EBP: base pointer – punta all’inizio dello stack locale; tutte le variabili locali sono referenziate relativamente a EBP

Convenzioni di chiamata C IA32

[DISCLAIMER: noi analizziamo il linugaggio C siccome ha un controll odiretto della memoria contrariamente a qualcosa come JAVA. Se infatti abbiamo un sistema basato su JAVA, dobbiamo andare a controllare se la vulnerabilità sia effettivamente presente nell’interprete JAVA].
La traduzione da C ad Assembly adotta alcune convenzioni standard.

Una di queste, è la convenzione di chiamata di default: **`_cdecl`**.
Questa convezione esegue le seguente oeprazioni:

- Inserimento sullo stack di tutti i paramtri attuali di chiamata in ordine inverso rispetto alla signature del metodo.
- Chiamata tramite “CALL” salvando l’indirizzo di ritorno sullo stack.
- 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.
- È compito del chiamante rimuovere i parametri attuali dallo stack

Altre convenzioni sono (che però non ci interessano):

- `__stdcall`
  - L'unica differenza con la modalità precedente è che è responsabilità del chiamato eliminare i parametri dallo stack.

- `__fastcall`
  - La differenza con la __cdecl è che i parametri attuali non sono passati via stack bensì tramite i registri generali (da EAX a EDX e ESI e EDI, quindi con un limite di 6 parametri di 32 bit).

Esempio di chiamata

Codice **chiamante** (convenzione __cdecl)

```
…
int result;
…
result = sum(4,5);
…
```

Compilazione

```
0x80401000 result: DW 1
…
0x8040200A PUSH    0x5
0x8040200F PUSH    0x4
0x80402015 CALL    _sum
0x8040201A ADD     ESP, 0x8
0x8040201F MOV     result, EAX
…
```

Indirizzo di ritorno

Il chiamante immette i parametri della funzione in cima allo stack in ordine inverso

Il chiamante invoca la funzione tramite l'operazione CALL = PUSH dell'indirizzo di ritorno + caricamento indirizzo funzione in EIP

Di ritorno dalla funzione, il chiamante rimuove dallo stack i parametri passati

Il chiamante prende il risultato dal registro EAX

Si fa l'ADD per sottrarre siccome lo stack cresce verso il basso. Inoltre sottraiamo siccome dobbiamo togliere i paramtri appena inseriti dallo stack.

Codice **chiamato** (convenzione __cdecl)

```
int __cdecl sum(int a, int b) {
    int c;
    c = a+b;
    return c;
}
```

Compilazione

```
PUSH    EBP
MOV     EBP, ESP
SUB     ESP, 0X4
MOV     [EBP-4], [EBP+8]
ADD     [EBP-4], [EBP+12]
MOV     EAX, [EBP-4]
MOV     ESP, EBP
POP     EBP
RET
```

Salva il contenuto di EBP, e memorizza in EBP il puntatore al frame pointer (contenuto di ESP)

Riserva sullo stack lo spazio per la variabile locale c

Effettua le operazioni usando come Riferimento (in base a cui muoversi sullo stack) il registro EBP

Salva il risultato in EAX

Ripristino di EBP e ESP

Ritorno al chiamante = POP dell'indirizzo in EIP

Do a EBP il valore di ESP, in questo modo dico che lo stack comincia da quel punto.

Possiamo vedere una animazione molto interessante ed esplicativa della condizione della memoria durante queste operazioni a questo link (slide 13–27).

Stack Overflow (o buffer overflow)

Oltre ad essere un sito tra usato, lo **stack overflow** è uno degli exploit più utilizzato nel campo dei binary exploit.
Abbiamo bisogno di alcuni prerequisiti per la sua attuazione:

- Presenza di un *buffer locale* ad una funzione (tale buffer si troverà quindi sullo stack).
- Possibilità di alimentare il buffer con input esterni (ad es. da tastiera, da file, da 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 dalla chiamata (sovrascrivendo quello preesistente).
- Risultato: Il flusso di controllo non procede con il ritorno al chiamante "lecito" bensì al "nuovo" indirizzo di ritorno posto dall'attaccante.

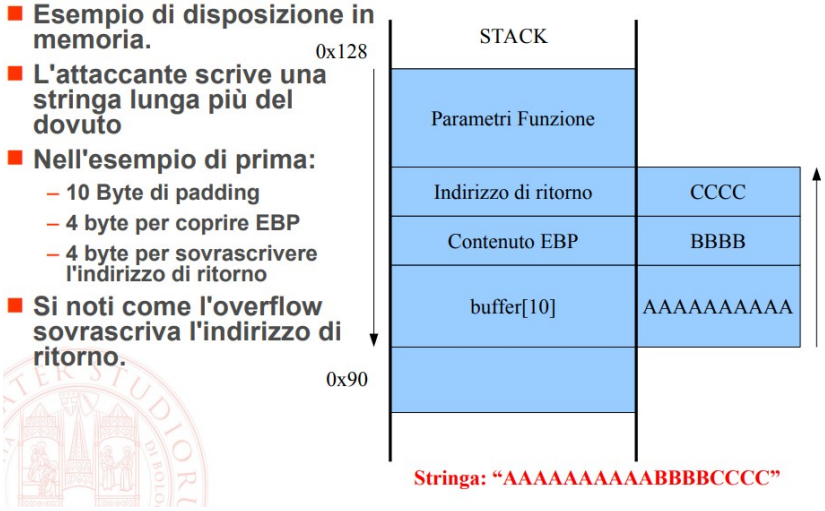
Questa vulnerabilità nasce dal fatto che il linguaggio C non controlla che i dati con cui alimentare i buffer/array abbiano una lunghezza congrua (tale controllo è a carico del programmatore). I buffer invece hanno già di loro un blocco di memoria riservao nello stack, e quindi una grandezza massima definita.
Inoltre, si ricordi che:

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

Un classico esempio di funzioni C che sono vulnerabili a buffer overflow sono `gets()` di C, che prende in un input una stringa da stdin senza controllare la lunghezza di essa, e la scrive su un array di char.

Vediamo un esempio:

- **Esempio di disposizione in memoria.**
- **L'attaccante scrive una stringa lunga più del dovuto**
- **Nell'esempio di prima:**
  - **10 Byte di padding**
  - **4 byte per coprire EBP**
  - **4 byte per sovrascrivere l'indirizzo di ritorno**
- **Si noti come l'overflow sovrascriva l'indirizzo di ritorno.**

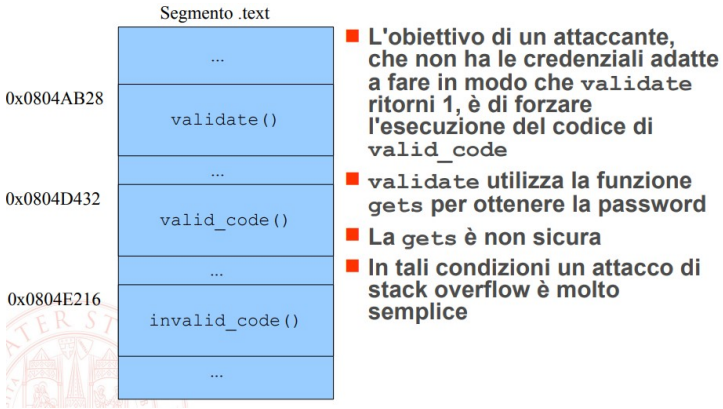


Notare come la frase in input venga distribuita "verso l'alto"

Le conseguenze dell'attacco possono essere:

- **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. Si hanno due alternative:
  - **Shell Coding**: si "inietta" (come parte della stringa) un pezzo di codice maligno preparato in precedenza. In questo modo, è possibile eseguire codice con i **privilegi** del processo vittima.
  - **Return to LibC**: la stringa iniettata per overflow scrive sullo stack i dati necessari per effettuare una invocazione di una funzione di libreria C.

Supponiamo che ci sia un utente che voglia accedere ad una funzione che gli permette di entrare nel PC a seguito dell'inserimento di una password corretta.
Il nostro utente è un hacker, e sa che la memoria dell'applicazione è organizzata in questo modo:



Il nostro utente allora scrive una stringa composta in questo modo:

- 10 byte di padding
- 4 byte per "scavalcare" l'EBP
- 4 byte per scrivere l'indirizzo di valid_code: 0x0804D432

In questo modo, quando la funzione `validate()` eseguirà il RET, allora invece che tornare al chiamante salterà a valid_code().
La stringa inserità avrà questa forma:

`AAAAAAAAAABBBB\x32\xD4\x04\x08`

Siccome l'architettura del PC è in formato **little endian**, dobbiamo invertire l'ordine dei byte.

Inoltre, è necessario tradurre l'indirizzo di valid_code nella sua rappresentazione ASCII. Il risultato comprende anche caratteri non stampabili. Esistono varie tecniche per passare al processo tali caratteri (ad es. printf di bash).

L'attacco Stack Overflow può essere svolto anche con architetture ad indirizzi crescenti. Semplicemente, si va a sovrascrivere l'indirizzo delle funzioni vulnerabili.

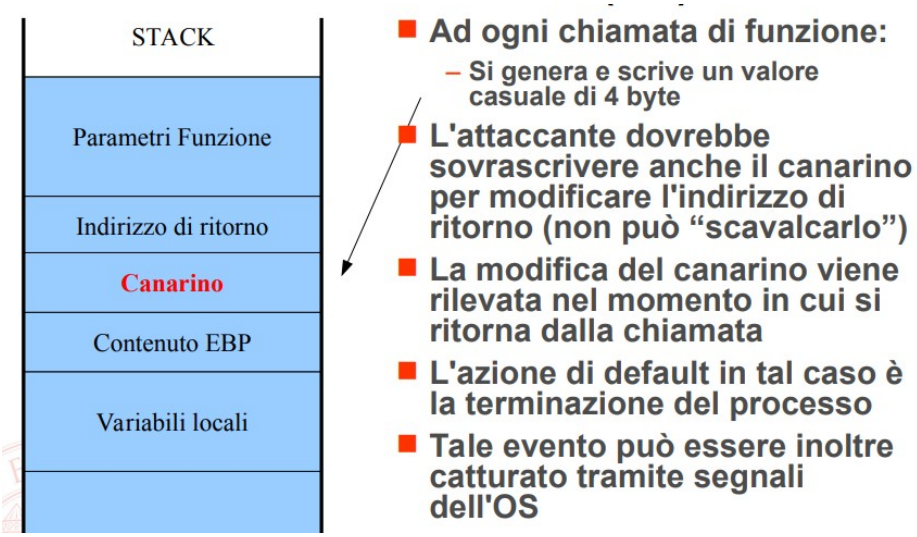
Canarini per lo stack overflow

È una contromisura per lo stack overflow. Questo consiste nel porre sullo stack, prima di un buffer, un dato di riferimento. Il processo è in grado di rilevare un tentativo di attacco verificando l'integrità di tale dato. Infatti, in caso di attacco con overflow il dato viene sovrascritto e la verifica da esito negativo. Tale protezione è **realizzata tramite la collaborazione congiunta di compilatore e libreria standard**, non è un meccanismo fornito dall'OS o dall'HW. Ovviamente, la generazione del dato e la verifica causano overhead.
Di default questa protezione non è abilitata, ma può essere abilitata in GCC attraverso le direttive:

- `--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 del buffer oltre la quale la protezione viene attivata. Questo evita che la protezione venga attivata per tutte le chiamate a funzione, riducendo l'overhead.

Esempio di Canarino

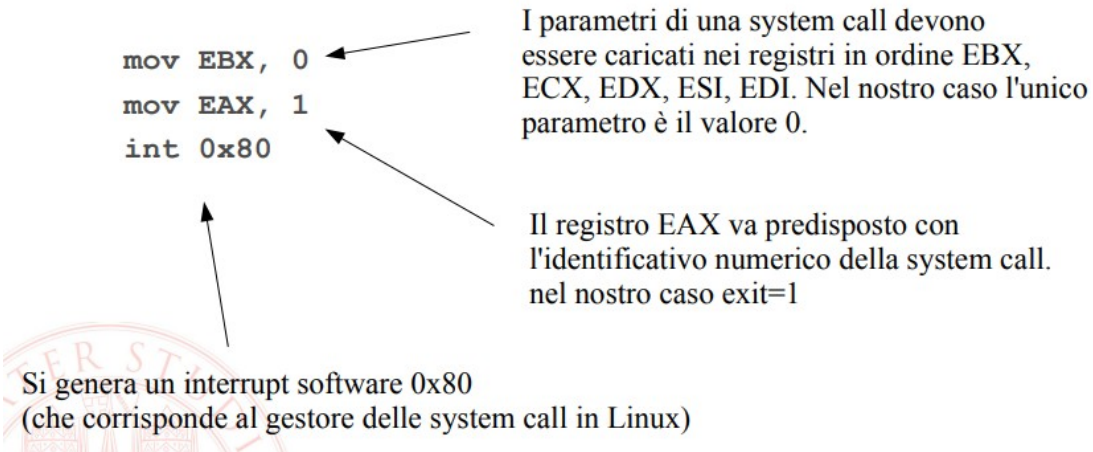


Il canarino è comunque attaccabile per forza bruta, ovvero possiamo provare ad indovinare il valore del canarino.

Shellcoding

Lo Shellcoding non è una forma di attacco alternativa, bensì è un tecnica per sfruttare lo stack overflow. Consiste nell'iniettare (tramite stack overflow) **sia il codice maligno che l'indirizzo di ritorno che punti al codice che abbiamo appena iniettato**.
Viene chiamato shellcoding siccome l'obiettivo tipico di tale approccio è l'apertura di una shell (da cui il nome), con i privilegi del processo attaccato (tipicamente root o con il bit setuid attivato). Il principio di questa tecnica è utilizzabile anche con altri tipi di attacco alternativi allo stack overflow.

Ecco un esempio di codice maligno che realizza l'invocazione della syscall exit con il valore 0, terminando il processo attuale:



Il codice sopra però ha un problema fondamentale: c'è una lunga serie di byte uguali a 0. Dunque, il primo di questi byte verrà interpretato come **terminatore di stringa**: il resto del codice verrebbe buttato via.

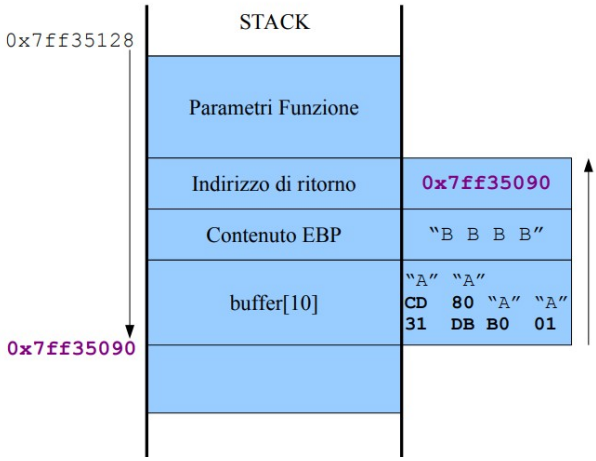
Dunque, bisognerà applicare la tecnica degli **Zeroes Cut Off**. Ovvero, inventarsi una stringa di

operazioni che hanno la stessa semantica di quelle precedenti, ma che vengono codificate in binario in modo diverso (ad esempio, senza byte 0).

```
xor EBX, EBX        31 DB
mov AL, 1     ⟹    B0 01
int 0x80            CD 80
```

NB: AL è l'insieme di 8 bit meno significativi del registro EAX

A questo punto, abbiamo una situazione in memoria del genere:



Il problema però sta nel fatto che lo stack è un'area fortemente dinamica: mentre è facilmente prevedibile dove sono collocate le funzioni (ovvero nel .text segment, infatti gli indirizzi di locazione [virtuali] sono sempre gli stessi stabiliti dal compilatore), lo stack invece dipende moltissimo da come è stato usato lo stack finora, quidni l'indirizzo di ritorno (ovvero 0x7ff35090) non è sempre facile da determinare (si può comunque calcolare e intuire però).

Dunque, spesso si fa uso di una **NOP Sled**. Essenzialmente, si usano dei byte 90 (che corrispondo all'operazione NOP in binario, ovvero operazioni che non fanno nulla) in modo che il mio codice in qualche modo cada nella zona del codice da eseguire.
In poche parole, da un margine di errore al salto.

Come facciamo a difenderci da questo tipo di iniezione? (**NX Stacks**)

Possiamo, con l'aiuto dell'hardware, impostare un flag in modo che alcuni frammenti di **memoria vengano contrassegnati come non eseguibili**. Infatti, se ci pensiamo, è un po' contronatura che gli stack siano eseguibili, siccome dovrebbero contenere solo dei dati. Per utilizzare questa funzionalità, però, deve essere presente il supporto del sistema operativo.

Questa feauture non presente nei processori INTEL/AMD a 32 bit. Una evoluzione della tecnologia degli NX Stacks è data dai **W^X**. Invece che marcare una singola pagina di memoria come non eseguibile, tutte le pagine vengono marcate o come non eseguibili o come non scrivibili.

Come aggirare NX Stacks?

Ci sono molti modi, ma solo due di essi è particolarmente importante:

- **RET2LIBC / RET2SYSCALL**
  - Tramite Stack Overflow si dirotta il flusso di esecuzione, piuttosto che verso codice sullo stack protetto da NX, verso una (o più) funzioni della onnipresente libreria C, oppure verso una system-call del s.o. . Se abbiamo accesso alla libreria standard C, possiamo fare veramente tutto, ed è molto facile trovarle siccome sicuraemente una le librerie dinamiche avranno almeno una chiamata ad una libreria standard C.

- Format Strings
  - Tramite la stringa di formato passata a printf si inietta codice e si forza il salto che lo esegue.
- Heap Overflow
  - Si sfruttano vulnerabilità specifiche dei metadati inseriti dalle librerie C per l'allocazione dinamica di memoria
- **Return Oriented Programming (ROP)**
  - Si assembla il codice da eseguire come sequenza di “gadget” (brevi sequenze di linguaggio macchina prese dai binari già in memoria)

RET2LIBC / RET2SYSCALL

Idea di base: non iniettare codice ma usare codice già caricato per altri scopi, come funzioni dell’onnipresente standard C library e system calls.
Usare stack overflow per iniettare puri dati

- riempire lo stack con indirizzi e parametri necessari a eseguire il codice scelto in modo che compia l’azione malevola
- saltare alla funzione di libreria o attivare la syscall

Esempio: si vuole eseguire `system(“/bin/sh”)`.
Per fare ciò, seguiamo questi passaggi:

- Trovare l’indirizzo della funzione di libreria system
- Trovare un modo di passare sullo stack la stringa “/bin/sh”
- Comporre lo stack in modo che alla ret, ESP punti alla cella che contiene l’indirizzo di system e che questa trovi sullo stack l’indirizzo del parametro atteso (la stringa)
- Si possono collocare strategicamente più indirizzi in modo che il ritorno da una library call ne scateni un altro (es. funzione `exit` se si vuole garantire una terminazione pulita del processo per mostrare un comportamento non anomalo)

In questo caso, trovare l’indirizzo della syscall non è troppo difficile:

- codice compilato staticamente → librerie incluse in .text
- codice linkato dinamicamente → entry point inclusi in .text come stub che caricano e chiamano la funzione a tempo di esecuzione
- disassemblare il binario è lo strumento principale

ASLR

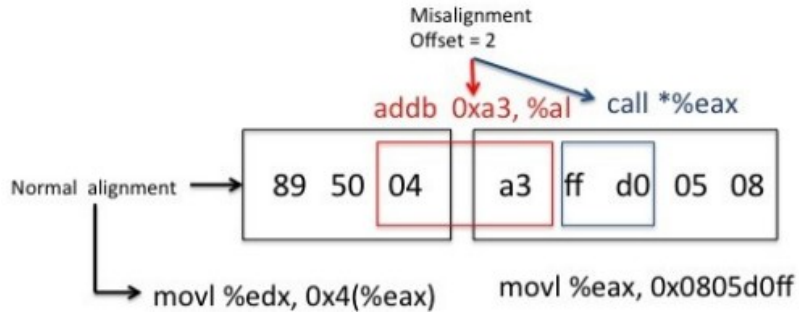
Una prima contromisura contro l'iniezione di codice maligno è l'**ASLR: Address Space Layout Randomization**.
E' una protezione offerta dall'OS. Si tratta di rendere casuale **l'indirizzo di partenza dei segmenti** che compongono un processo (non tutti, ma solo: indirizzo delle librerie, base dello stack, 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.

Non modifica però il punto delle syscall, che sono comunque eseguibili. E nemmeno .text. Motivo per cui è facile far girare la vulnearbilità RET2SYSCALL.

ROP (Return-Oriented Programming)

Generalizza il meccanismo di prendere delle funzioni in uso nel programma (e che quindi sono presenti in .text). In poche parole, setacciamo il file .text in modo da trovare un qualasiasi *gadget*

(ovvero una qualsiasi sequenza di codice che termini con RET).
Dunque, vado a cercare nel .text tutte le occorrenze del singolo byte che codifica RET (o un salto).
Essenzialmente deve succedere una cosa del genere:



Nel nostro codice ho 89 50 04… Ma io volendo potrei trovare il modo di far eseguire al processo solo 04 a3 (che viene interpretata coma addb) e ff d0 (che viene interpretata come call *%eax)

Iniettiamo 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 → salta al gadget successivo

Contromisure

In questo caso, NX-Stack e W^X quasi inutili siccome i vari attacchi fanno uso "legittimo" dello stack.

L'ASLR inoltre non si applica a .text, e si può aggirare facendo brute forcing degli indirizzi (difficile nelle arch. 64bit) oppure utilizzando gli stessi puntatori alle funzioni del codice lecito.
Una estensione dell'ASLR che lo rende più sicuro è PIE (Position Independent Executable).
- .text randomizzato
- richiede doppia indirezione dei puntatori a funzione (tabelle PLT e GOT)
  - sovrascrivibili per dirottare le invocazioni!
  - hardening: RelRO

Possiamo sennò usare i canarini. Questi sono validi, ma:
- rallentano
- esistono alcuni modi di aggirarli
  - es. bruteforcing di processi che forkano figli: il canarino verrà copiato identico nello stack del figlio
  - provo ad attaccare il figlio, se sbaglio crash
  - faccio generare un altro figlio, che avrà canarino identico
- generalizzazione: **CFI (Control Flow Integrity)**

CFI (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.

Lezioni del 16/03/2022 – Laboratorio di Binary Exploitation

Introduzione al Binary Exploitation

Quando stavamo facendo privilege escalation, stavamo sfruttndo una vulnerabilità del binario, che era nota e che aveva delle funzionalità importanti.
Ciò significa che conoscevamo il comportamento del binario, e in generale la vulnerabilità consisteva nella sua cattiva configurazione (ad esempio, quando abbiamo usato sed, questo era stato configurato in modo da avere la privilege escalation).

Ma cosa succede se invece un binario che consideriamo sicuro sia stato programmato in modo da contenere una vulnerabilità? A questo punto, io posso lavorare sull'input del programma in questione.

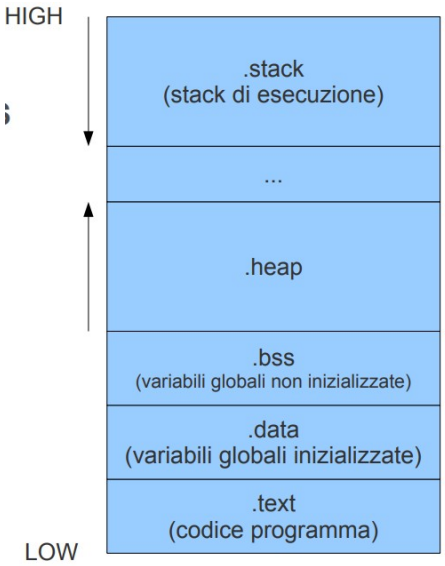
Little tip:
prima di fare queste cose, bisogna fare in modo che la randomizzazione della memoria sia settata a 0.
`echo 0 > /proc/sys/kernel/randomize_va_space` per farlo.

Quando abbiamo scaricato i programmi, lo dobbiamo compilare usando i flag:

- `-fno-stack-protector`, disabilita i canarini
- `-m32`, compila per architettura 32 bit
- `-z execstack`, rende lo stack eseguibile

Ripassino di quello che ha detto il prof nella lezione precedente

Ogni processo ha il suo spazio di memoria dedicato, che contiene molti segmenti di esecuzione. Tra questi, abbiamo l'`.heap` (cresce verso l'alto) e lo `.stack` (cresce verso il basso). Noi lavoreremo soprattutto con lo stack.



Teniamo a mente, comunque, che il programmatore (e il processo) non vedono gli indirizzi fisici, ma solo quelli virtuali (che poi vengono tradotti in fisici dall'MMU + OS).

L'architettura di cui ci interessa nel nostro caso è l'IA32, che è a 32 bit. In questa architettura, ci sono due registri importanti che hanno la funzione di gestire lo stack. Questi sono:

- lo **ESP,** che fa da stack pointer e punta all'ultima cella occupata dallo stack.
- Lo **EBP**, che fa da base pointer, e punta all'inizio dello stack locale. Tutte le variabili locali sono referenziate in modo relativo a EBP.

Teoria dello stack

Siccome nello stack c'è tutto che è in esecuzione sul programma (per esempio, anche le stringhe di input), possiamo sfruttare le operazioni in cui una stringa viene inserita e non validata, in modo da sovrascrivere parti dello stack che non erano state pensate per essere scritte dall'input, causando un attacco di Stack Overflow (o Buffer Overflow).

GDB

Il tool principale che useremo in questa lezione è **GDB**, che è un debugger ed è uno degli strumenti più utili nel reverse engineering. Questo programma serve in sostanza per testare altri programmi, e il compito principale del debugger è quello di mostrare il frammento di codice macchina che genera un determinato problema.

Esiste anche una GUI di GDB, che è più user friendly rispetto alla versione da terminale.

Possiamo eseguire gdb sul nostro programma compilto prima usando **`gdb [nome del programma]`**.
Ora che siamo sulla console di gdb, possiamo usare **`run ciao`** per dare in input al programma la stringa "ciao".
Con il comando **`b *main`**, mettiamo un breakpoint all'inizio della funzione main.
Ora che rifacciamo run stringa, il nostro programma si fermerà al breakpoint. Dopo averlo raggiunto, possiamo usare il comando `next` per andare all'azione successiva, oppure il comando `step` permette di andare alla linea successiva dell'assembly.

Possiamo poi usare il comando `x/200xw $esp` per dire a gdb di stampare i 200 byte successivi dello stack partendo da esp.
L'output che ci comparirà sarà una serie di valori esadecimali, che inizialmente potremmo non comprendere.

Altri comandi utili sono:

- `disas [nome funzione]` (es. `disas main`), che mostra l'assembly associato ad una funzione.
- `info functions`, che mostra gli indirizzi di tutte le funzioni caricate in memoria in quel momento. Se abbiamo incluso delle file .h, ci verranno mostrate un sacco di funzioni, anche per esempio nel nostro programma ne abbiamo create solo due.
- `info registers`, che mostra lo stato attuale dei nostri registri.

Buffer overflow

Nel caso del nostro esercizio, noi dobbiamo fare in modo che ci stampi il flag richiesto, e ce lo stampa solo se il valore della variabile control è di un determinato valore. Dunque, per provare, possiamo provare a scrivere una stringa molto lunga e vedere cosa succede.
Ad esempio, `./es $(perl –e 'print "A"x100')` che ci stampa AAA* con 100 A.
Notiamo che non cambia nulla, l'output dell'applicazione rimane lo stesso. Dunque, proviamo ad aumentare finché l'output cambia.

A questo punto, dopo situazioni di prova ed errore, finalmente possiamo riuscire a scrivere la stringa come la vuole il programma.

Siccome l'architettura del nostro computer è **Little Endian**, ci tocca scrivere la parte finale del nostro input (che è quella che sovrascriverrà *control*) al contrario.

Funzione segreta

Noi dobbiamo invocare una funzione di cui conosciamo l'indirizzo, ma che non viene mai invoxata nel codice vero.

Innanzitutto, dobbiamo trovare il modo di far crashare il nostro programma. Per fare ciò, procediamo per prova ed errore, ed iniziamo con un payload piccolo.
GDB ci mostra che il nostro programma va in segmentation fault, e ci mostra l'ultimo indirizzo di ritorno che ha causato il crash.
Usiamo il comando `run $(perl –e 'print "A"x20')` per vedere che succede.
L'indirizzo di ritorno, nel nostro caso però, è pari al codice ASCII delle A inserite, e quindi effettivamente è saltato nella zona interessata.
Possiamo sempre la nostra tecnica di prova ed errore per eseguire il nostro codice e capire esattamente il numero di byte necessari per scrivere correttamente l'indirizzo di ritorno.
A sto punto, troviamo l'indirizzo della funzione secret (che è qurlla che contiene il flag),

riscriviamo l'indirizzo della nostra funzione al posto delle BB nel payload (tenendo a mente che i byte dell'indirizzo vanno sempre scritti al contrario a causa dell'endianess).

La nostra funzione darà comunque segmentation fault (siccome sovrascriverà anche altri dati necessari al ritorno) però a noi non ci interessa, siccome la funzione secret viene eseguita lo stesso e ci stampa il flag.

Shellcode dopo buffer overflow

Adesso, la cosa che ci rimane da fare è sfruttare la vulnerabilità per eseguire dello shellcode malevolo.

Creiamo l'ambiente per fare ciò rendendo root il proprietario del file, e poi rendiamo il file a noi eseguibile (con chmod). In questo modo, ogni volta ceh eseguiamo il programma, lo eseguiremo comportandoci come root.

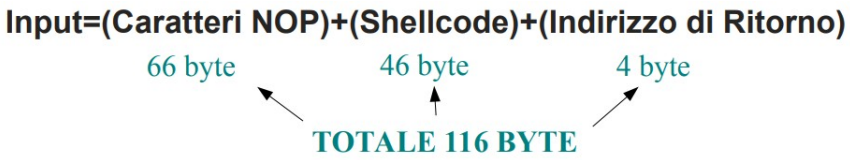
Prima di tutto, ci comportiamo come prima (nel caso di buffer overflow) e cerchiamo di trovare il numero di byte necessari per riscreve l'indirizzo di ritorno (che chiamiamo **offset**).

A sto punto, se facciamo `x/300xw $esp` dentro gdb possiamo vedere la situazione in memoria nello stack del nostro programma. E noteremo che possiamo vedere la nostra di A e B (sottoforma di codice ASCII).

Se io al codice dell'indirizzo di ritorno metto l'indirizzo di del codice shellcode, il nostro programam lo esegue. In particolare, con Shellcode si intende del codice esadecimale che ci permette di eseguire una shell. Possiamo facilmente generare uno shellcode usando dei tool appositi (un esempio è msfvenom) siccome la scrittura di shellcode apposito è piuttosto complessa.

Se però abbiamo un offset di 112 caratteri, in questi 112 caratteri noi dobbiamo scrivere il nostro shellcode, e dovremo riempire gli spazi "inutili" con del codice che comunque sia eseguibile (le A quindi non vanno bene). Possiamo allora usare dei caratteri NOOP (ovvero \x90) che si incontrati non fanno niente e èassano al carattere successivo.

Nel nostro caso, l'input dev'essere di 116 byte, mentre il nostro shellcode è di 46 byte. Quindi, dobbiamo fare:



A questo punto, dobbiamo trovare l'indirizzo di ritorno.
Per farlo, prendiamo uno degli indirizzi che punta al NOOP. Ed è fatta.
Adesso, se scriviamo la stringa che funziona e ci fa eseguire una shell dentro al programma, e avviamo il comando partendo dalla root. In questo modo, potremo finalmente eseguire la nostra shell da root.

Return to LibC

Se il nostro stack **non è eseguibile**, per come è compilato il programma C (ovvero senza il falg che abbiamo inserito prima), allora cosa possiamo fare? Una delle tecniche che **possiamo usare è il return to LibC**.

Ovvero, per si sfrutta ciò che si ha già in memoria insieme al processo, per eseguire qualcosa di malevolo. La **libc** è la libreria di sistema base che viene caricare con qualsiasi programma in linux. Nella libc c'è la **system** che, insieme ad altre informazioni caricate in memoria, possiamo usare per

eseguire una shell.
Al posto dell'indirizzo di ritorno mettiamo quindi una system.

Per trovare le varie cose che ci servono, usiamo i comandi:
- gdb) p system // per trovare l'indirizzo della system
- gdb) p exit // per trovare l'indirizzo della exit
- gdb) x/500s $esp // per guardare tutto ciò che è caricato come variabile d'ambiente, tra cui la variabile SHELL che contiene esattamente il valore "/bin/sh"

Siccome la variabile della shell inizia con SHELL=, dovremo aggiungere un ulteriore offset di 6 byte.

La stringa finale sarà data da OVERFLOW + indirizzo system + indirizzo exit + Variabile SHELL.

Lezioni del 18/03/2022 – Intro alla sicurezza delle reti

Soluzione agli esercizi vecchi

Consiglio la visione delle soluzioni. Sono abbastanza utili.

Inizio Sicurezza delle reti

Internet è una rete di reti, e la componente elementare di internet è la network IP, che è una sorta di isola. L'isola tipicamente contiene dei calcolatori che fungono da nodi terminali della rete, detti host. Inoltre, ogni isola è collegata fra loro attraverso particolari apparati ceh svolgono la funzione di "ponte" fra queste isole, e vengono chiamati gateway router.

Ogni dispositivo connesso alla rete possiede:
- Un indirizzo globale:
  - È valido per tutta la rete, e deve essere univoco per evitare le ambiguità. Va assegnato, seguendo una procedura di gestione globale che assicura la non replicazione.
  - MAC Address
- Un indirizzo locale:
  - È valido limitatamente ad una certa sottoporzione della rete.
  - Può non essere univoco.
  - Possono essere assegnati a livello locale.

Nella terminologia di Internet si definisce:
- **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

La rete fisica normalmente ha capacità di instradamento e può avere indirizzi locali (es. indirizzi MAC). Siccome Internet possiede un'architettura a strati, gli indirizzi fisici vengono nascosti e le applicazioni possono così lavorare solo con indirizzi IP.

[poco importante]

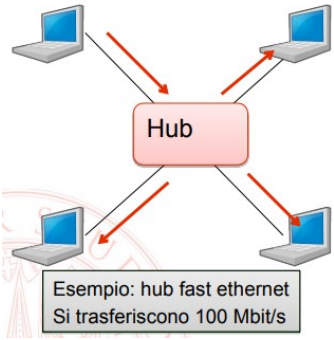
Esistono molti tipi di reti locale (o network IP):
- Wi-Fi : Network realizzata con tecnologia wireless in area locale
- ADSL e xDSL: Network realizzata con tecnologia a media distanza via cavo tramite infrastruttura di uno specifico fornitore di servizio pubblico
- Ethernet: Network realizzata con tecnologia a breve distanza via cavo privata in area locale
- GPRS/EDGE/LTE: Network realizzata con tecnologia radio a media distanza tramite infrastruttura di uno specifico fornitore di servizio pubblico

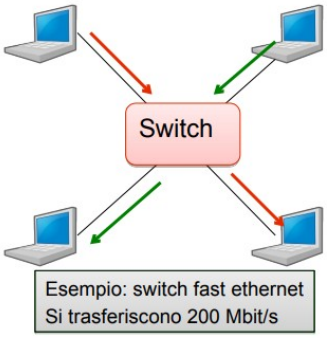
[fine parte poco importante]

Per connettere più reti locali fra di loro, possiamo usare dei bridge, che connettono fra loro due diverse LAN. Un bridge che interconnette fra di loro più di 2 reti LAN viene detto HUB. Un solo switch Ethernet svolge una funzione simile all'hub, ma garantendo migliori prestazioni. È in grado di trasferire contemporaneamente trame da più porte di ingresso a più porte di uscita. Opera una funzione di commutazione a livello 2 basata sull'indirizzo MAC.

La differenza principale fra Hub e Switch sta nel fatto che:

- Hub
  - bus collassato = mezzo condiviso, trasmissione broadcast delle trame
  - Capacità aggregata = capacità della singola porta
- Switch
  - Sistema di commutazione = ri-trasmissione selettiva delle trame
  - Capacità aggregata superiore a quella della singola porta





Lo Switch costruisce una "Tabella di inoltro". Associa il Mac Address delle interfacce alla porta dello switch su cui si trova collegata l'interfaccia stessa. Se alla porta è connesso un altro switch, da essa si raggiungono molteplici MAC di destinazione.

Abbiamo inoltre un tipo di switch, detto learning switch, che permette di creare la tabella di inoltro in base alla comunicazione vera e propria della rete (ovvero, crea le tabelle di inoltro nel tempo). Tuttavia, questo modello non scala, siccome gli switch impara gli indirizzi uno per uno e se abbiamo milioni di connessioni abbiamo un problema.

La network IP ha la stessa struttura della rete globale, e tutti gli host che appartengono alla stessa network IP sono in grado di parlare fra loro grazie alla tecnologia con cui questa viene implementata. Dunque, 2 host che sanno di appartenere alla stessa subnet, sanno di non aver bisogno di nessun intermediario.

Interconnessione tramite switch: tutti gli host usano la stessa subnet mask. Quindi, tutti gli host appartengono alla stessa sottorete.

Interconnesione tramite router: ci sono più sottoreti connesse dallo stesso router. Dunque, due host che stanno queste due sottoreti avranno bisogno di un router (ovvero di un intermediario).

Il protocollo ARP (Address Resolution Protocol) è un **protocollo di tipo broadcast**. Lo spiego in modo veloce.
Un router prima di inviare un pacchetto, chiede quale indirizzo MAC è associato all'IP a cui vuole inviare il pacchetto. L'host glielo comunica e riceve così il pacchetto. La risposta che da l'host è unicast.

ARP inoltre è un modo per far rispondere SEMPRE ai computer, anche sono occulti (cosa che invece non succede coi ping)!

Lezioni del 25/03/2022 – continuazione fondamenti di rete, attacchi sulle reti

Continuazione parte teorica sulle reti

Per far parlare tra loro le isole (network IP) è necessario che:

- Vi siano dei collegamenti fra le isole stesse, spesso realizzati con tecnologie diverse da quelle dell'isola
- Vi siano degli apparati che permettono di usare questi collegamenti nel modo opportuno
- Sia possibile scegliere il giusto collegamento verso l'isola che si vuole raggiungere

I router inoltrano i pacchetti, e ragionano in molto diverso degli apparati fisici della rete locale (come ad esempio gli switch e gli hub non fanno altro che propagare il traffico facendo in modo che raggiungesse *tutti* i dispositivi sulla stessa LAN). Il router invece deve analizzare il pacchetto, e in base alle informazioni del pacchetto deve decidere come e dove inoltrarlo. In poche parole, deve trovare il modo migliore per prendere un pacchetto e portarlo a destinazione.

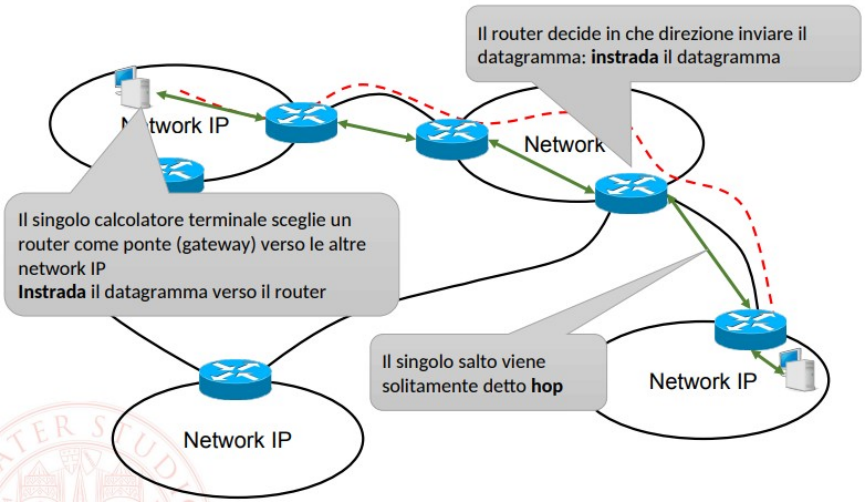
Cosa fa IP?

La tecnologia IP è agnostica rispetto alla tecnologia con cui sono realizzate le network . Il protocollo IP è concepito per lavorare indifferentemente su tecnologie diverse. **L'obiettivo di IP è quello di rendere possibile il dialogo fra network a prescindere dalla loro implementazione e localizzazione**.

Quando un pacchetto arriva a un router, questo si pone una domanda cruciale e deve scegliere se trasmettere il pacchetto sulla sua rete locale oppure mandarlo ad un gateway. Per rispondere a questa domanda, il router usa una base di dati di destinazioni possibili **(ovvero la tabella di instradamento)**, e prende la decisione in base ai dati contenuti in essa. La tecnologia della propria rete può dunque essere usata per:

- raggiungere la destinazione finale.
- raggiungere il primo ponte da attraversare.

Ogni salto viene detto ***hop***.



Instradamento diretto e indiretto

- **Routing** : scelta del percorso su cui inviare i dati. I router formano struttura interconnessa e cooperante: i datagrammi passano dall'uno all'altro finché raggiungono quello che può consegnarli direttamente al destinatario.
- **Direct delivery**: IP sorgente e IP destinatario sono sulla stessa rete fisica. L'host sorgente spedisce il datagramma direttamente al destinatario.
- **Indirect delivery**: IP sorgente e IP destinatario non sono sulla stessa rete fisica. L'host sorgente invia il datagramma ad un router intermedio.

[PS: per compilare le tabelle di routing, i route annunciano alla rete che sono i router che servono per raggiungere una certa sottorete]

Alcuni esempi di sicurezza nei vari layer:
- livello fisico: IEEE 802.1X, WPA2, Layer-2 VPN
- livello rete: IPSec, Layer-3 VPNs, router authentication
- livello trasporto: SSL, TLS
- livello applicazione: authentication, crittografia, application-layer VPNs.

Attacchi passivi

Si parla di **attacchi passivi** quando gli attacchi non modificano i dati in transito.
Tra questi, abbiamo:

- la scansione (**scanning**) è uno dei primi passi della ricognizione
- lo **sniffing** può compromettere la riservatezza dei dati
- il recupero di una **chiave** consente di impersonare la vittima

Scanning - esempi

- Scansione di una rete: indirizzi raggiungibili
- Scansione di un host: porte TCP / UDP aperte, consente di dedurre le versioni del sistema operativo e dei servizi in esecuzione
- "Loudness": per scopi VA, la scansione può essere aggressiva. Gli strumenti implementano molti modelli di scansione silenziosa per eludere il rilevamento

Sniffing

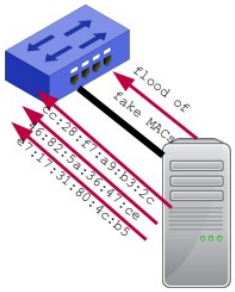
Lo sniffing richiede l'accesso fisico ai dati in transito (Per esempio: stando già sulla rete locale, oppure a seguito di un attacco di dirottamento).
Su reti locali:

- **wireless**: tutto dovrebbe essere criptato, ma molti protocolli sono difettosi!
- **cablate**: la crittografia esiste (802.1x per l'autenticazione delle porte, 802.1AE per la cifratura del traffico) ma non la usa quasi nessuno

MAC flooding

Gli switch offrono una protezione limitata: idealmente, il traffico viene inviato solo sulla porta del destinatario. Se lo switch **non trova un MAC nella CAM** (content addressable memory, aka la tabella) manda i pacchetti in broadcast. Es. switch con CAM di dimensione 6 righe

Porta	MAC raggiungibile
1	eb:a6:99:de:1c:b0
1	2c:65:1e:b1:9f:44
3	0c:2e:22:b0:8e:16
4	5b:06:72:1b:3c:03
4	e4:b0:56:d5:2d:0f
4	92:ff:9e:6c:b0:8e



Porta	MAC raggiungibile
1	46:82:5a:36:47:ce
1	cc:28:f7:a9:b3:2c
1	e7:17:31:80:4c:b5
1	...
1	...other fake MACs
1	...

Il MAC flooding costringe lo switch a comportarsi come un hub.

Wireless key recovery

Ci sono quattro principali generazioni di protezione delle reti WiFi: WEP, WPA, WPA2, WPA3.

- WEP (Wired Equivalent Privacy)
  - chiave simmetrica precondivisa
  - stream cipher RC4, falla di progettazione: è possibile recuperare la chiave se viene raccolto sufficiente testo cifrato prodotto dalla stessa chiave
  - la chiave è "randomizzata" da XOR con un IV piccolo (24 bit)
  - generate abbastanza traffico e l'IV si ripeterà (oppure facendo cadere e poi rialzare la connesione)
- WPA (WiFi Protected Access)
  - una patch intermedia durante il lancio di WPA2 • sostituisce IV con TKIP (128 bit) – modalità personale con chiave precondivisa • nessuna segretezza in avanti: qualsiasi utente che conosce la chiave potrebbe decrittografare tutti i pacchetti – modalità aziendale con autenticazione utente su canale protetto Wireless key recovery
- WPA2
  - a lungo considerato essenzialmente sicuro
    - grave vuln scoperta nel 2017: attacchi di reinstallazione chiave (KRACK) • Android e Linux possono essere indotti a (ri) installare una chiave di crittografia completamente zero
    - In altri dispositivi è comunque possibile decrittografare un gran numero di pacchetti
    - I pacchetti possono contenere credenziali utente con validità a livello aziendale!
  - Pre-shared key (PSK) corta se si usa WPS
- WPA3
  - vari miglioramenti a garanzia della scelta di cifrari robusti
  - sostituisce PSK con Simultaneous Authentication of Equals (SAE)
    - usa un sistema di handshake detto Dragonfly
  - vulnerabile ad attacchi Dragonblood
    - tipo 1: sfrutta la retrocompatibilità con WPA2 – attacco MITM per forzare downgrade
    - tipo 2: sfrutta implementazione non corretta di alcuni passaggi crittografici – consente password partitioning
  - dispositivi aggiornabili

Attacchi attivi

Gli **attacchi attivi** minacciano l'integrità, l'autenticità o la disponibilità di reti e sistemi.
Spoofing e hijacking sono spesso un passaggio preliminare per un attacco più impattante, ad es:

- rubare una rete per originare spam e scomparire
- fingere un'identità di rete per rubare le credenziali
- dirottare il traffico per fare sniffing

Denial of Service (DoS) rende inaccessibile un servizio.

Attacchi nel Link layer (livello data-link)

- **Spoofing MAC**: si assume l'identità di un dispositivo a livello di indirizzo fisico. È molto efficace sia per bypassare ACL (es. se ho uno switch con dei MAC address scritti a mano che vengono usati per usare la mia rete), che per ottenere tutto il traffico destinato alla vittima. **È però limitato alla LAN**, ma tecnicamente facile da mitigare: 802.1x (ma organizzativamente complesso → raro che lo si faccia!).
- **ARP Poisoning**: convincere un host (specialmente il gateway) che l'IP di una vittima è associato al MAC dell'attaccante. In questo modo si avvelena anche la cache degli altri utenti.

Attacchi del livello rete

- **IP spoofing**
  - assumere l'indirizzo IP di una vittima
  - efficace per dirottare il traffico solo su LAN. Su Internet, il routing invierà le risposte agli indirizzi mimati (siccome non ha effetto sul routing dei pacchetti).
    - molto facile da fare su LAN, difficile su rete globale.
  - l'attaccante non può ottenerli
  - attacchi di backscatter! → →
  - utile per gli attacchi riflessi (replay attack), oppure per DOS facendo richieste al nome della vittima che si vuole bloccare.
- **IP hijacking** – i router si scambiano informazioni su come raggiungere le destinazioni – posso inviare pacchetti BGP per convincere i router di routing che le tabelle di routing sono state modificate, siccome **BGP non è autenticato**!
  - Estende la portata dello spoofing IP su scala globale
- Entrambi utili per bypassare ACL e 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 privo di connessione: molto facile
- TCP perderà la connessione se l'attaccante non utilizza i numeri di sequenza corretti per la finestra scorrevole (sliding window). Ma è comunque utilizzabile per DoS.

Spesso i protocolli a livello di applicazione utilizzano identificatori di sessione come i cookie HTTP.

In entrambi i casi l'attaccante ha due opzioni:

- indovinare (forza bruta, spesso molto difficile)
- sfruttando lo sniffing (se già sul percorso dei dati)

(D)Dos

- Qualsiasi attacco dirottamento può causare un errore mirato. Livello di trasporto: l'invio di un SN errato o un reset esplicito su connessioni TCP li 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)**. Botnet IoT: Mirai, Bashlite, ...
- un controllo degli accessi impreciso sull'infrastruttura può peggiorare le cose:
  - in termini di effetto
  - nascondendo l'origine
  - per esempio: attacchi di amplificazione DNS

Un esempio di esfiltrazione: DNS Tunnelling

Query e risposte DNS possono contenere *dati.* Utilizzabile per esfiltrare dati da un computer infettato o per mettere in contatto un bot con il C&C

Protocolli ausiliari: DNS hijacking

Un server DNS malevolo può fornire in risposta l'IP dell'attaccante quando viene richiesto dalla vittima di risolvere un nome legittimo.

Il DNS ha vari problemi:

- 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 pharming
Ci spiega come poter far in modo di convincere gli altri host a fare richieste a me rispetto che al DNS lecito, in modo da rispondere alla richiesta DNS dando gli IP che vogliamo noi?
Ad esempio, cliccando su un link all'interno di una mail, e dentro la pagina HTML ci potrebbe essere un exploit javascript.

L'exploit agisce sul router della rete locale [siccome un tempo l'80% dei router domestici erano settati con credenziali di default], comunicando con esso infettando uno degli host amministrativi che ha accesso al router. In questo modo, l'attaccante usa il router stesso come server DNS, e ogni volta che accede a un sito, la vittima verrà dirottata sul sito dell'attaccante.

Lezioni del 30/03/2022 – Arp, DHCP,

bho
bho

Lezioni del 1/04/2022 – Cifratura

Cos'è la crittografia?
La crittografia è una tecnica che permette di garantire la sicurezza delle informazioni.
Le caratteristiche della sicurezza delle informazioni sono:

- **Riservatezza** (le informazioni non dover poter messe a conoscenza di enti non autorizzate a visonarle)
- **Integrità** (poter riconoscere con certezza se un dato se un dato è stato modificato o meno) e **autenticità** (garantire che l'informazione derivi dall'ente che l'ha effettivamente mandato)
- **Disponibilità** (non può essere difesa crittograficamente)

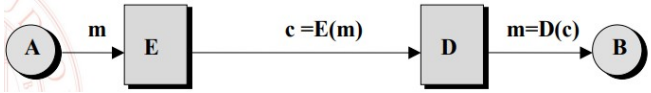
La crittografia consiste nell'elaborazione matematica e algoritmica della codifica delle informazioni.
Permette di :

- Prevenire la violazione della riservatezza (una rilevazione 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 (non può essere prevenuta!): aggiungere al codice elementi che permettano la verifica delle informazioni ricevute.

Steganografia
L'arte e la scienza del comunicare senza che altri se ne accorgano. Esempi storici: ● La tavoletta cerata di Demerato
Le tecniche moderne equivalenti sono, ad esempio: modifica dei bit meno significativi di dati multimediali.

Operazioni principali nella crittografia
Abbiamo due operazioni principali nell'ambito della crittografia:

- Cifratura converte il testo in chiaro in testo cifrato
- Decifrazione converte il testo cifrato in testo in chiaro



Algoritmi segreti

Rappresentano una possibilità di rendere irrealizzabile D, quindi di dare una scatola nera per cui sappiamo che se ci inseriamo dentro un messaggio cifrato *m'* otteniamo un messaggio in chiaro m.

- Benefici apparenti: difficoltà di studiare come invertire la cifratura
- Problemi:
  - mancanza di revisione della qualità
  - difficoltà di diffusione delle procedure
  - difficoltà di sostituzione delle procedure

Ovviamente, se nessuno sa com'è fatta la funzione D, comunque trovarne le debolezze diventa più complesso.

Principi di Kerckhoffs

Questi sono i principi base di un buon algoritmo di cifratura

1. Deve avere **sicurezza computazionale o assoluta**
2. L'algoritmo non deve essere segreto, ma la chiave dell'algoritmo deve essere segreta.
3. Cifratura = ricordare un segreto semplice per poter scambiare molti segreti arbitrari

Metodi della crittoanalisi

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

- **Forza bruta**: si tira ad indovinare D (o il suo particolare segreto) e si decifra il testo intercettato: se non ha alcun senso si ripete il procedimento
- **Solo testo cifrato**: si eseguono analisi statistiche su una grande quantità di materiale cifrato e se ne usano le indicazioni per individuare quale *p* probabilmente corrisponde ad un dato *c*
- **Testo in chiaro noto**: ci si procura in qualche modo sia dei testi cifrati, sia i corrispondenti testi in chiaro e si cerca di dedurre D analizzando le varie coppie
- **Testo scelto**: si può scegliere testo da cifrare o da far decifrare per ottimizzare il procedimento di deduzione della chiave
- **Rubber hose**: si minaccia, ricatta o tortura qualcuno finché non cede la chiave.

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

- Analizzare le proprietà statistiche del testo. Ci possiamo difendere attraverso:
  - *robustezza* = capacità dell'algoritmo di occultare le proprietà del testo in chiaro
- Cercare la chiave tra tutte quelle possibili. Ci possiamo difendere attraverso:
  - *sicurezza assoluta* = rendere totalmente indistinguibile la chiave giusta dalle altre
  - *sicurezza computazionale* = rendere troppo oneroso il processo di ricerca della chiave

Mattoni della robustezza – confusione

La proprietà di **confusione** misura il grado in cui la *struttura della chiave* viene resa irriconoscibile nel testo cifrato.
Una modifica di un singolo elemento della *chiave* dovrebbe riflettersi sul 50% del testo cifrato. In questo modo, l'analisi del testo cifrato non restituisce indicazioni utili sul valore della chiave.

Mattoni della robustezza – diffusione

La proprietà di **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 del *testo in chiaro* dovrebbe alterare il 50% del testo cifrato.
In questo modo, l'analisi del testo cifrato non restituisce indicazioni utili sul testo in chiaro.

Sostituzione monoalfabetica

La ***sostituzione*** è il modo più semplice di introdurre **confusione**.
Ci sono mille enigmi: Cifrario di Cesare, Agony Columns del Times, parole crociate crittografate della Settimana Enigmistica…

La sostituzione si attacca considerando la probabilità statistica dell'uso di certe lettere.
La sostituzione è molto più efficace se applicato sui bit, mentre è facilmente attaccabile se applicato per esempio al linguaggio naturale (soprattutto se si considerano le statistiche di frequenza dei caratteri).

Trasposizione

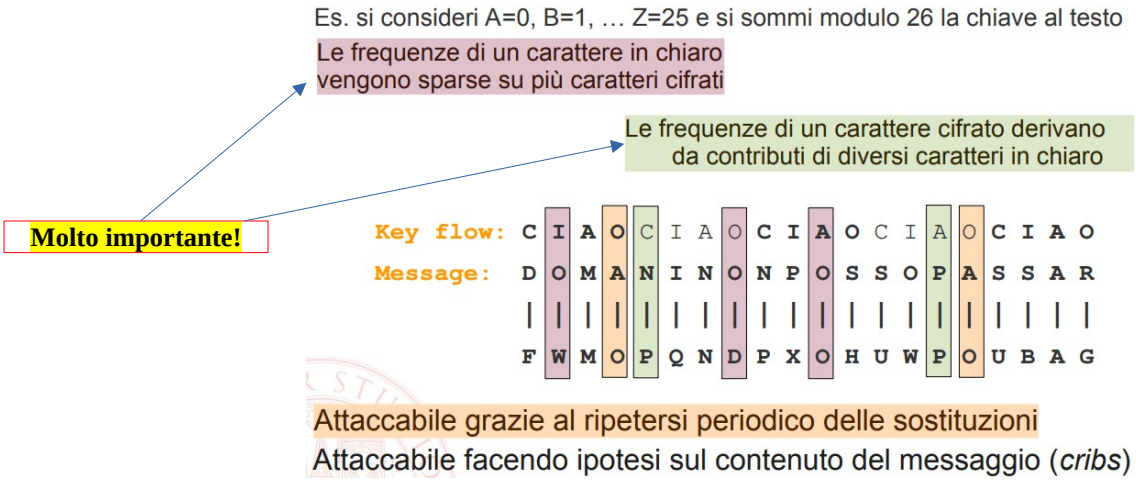
La ***trasposizione*** è il modo più semplice di introdurre **diffusione**.
Un esempio è la scitala degli Spartani, che si basa in modo algoritmico su una tabella scritta per colonne e letta per righe.

A	P	I	L	N		E	A	
L	R	M	E	I	P	N	N	I
L	O	E	Z		R	T	C	O
E	S		I	M	E	O	H	
	S	E	O	I	S		E	

La trasposizione si attacca considerando non solo la probabilità statistica delle singole lettere, ma quella delle coppie di lettere.
In particolare, guardo se i digrammi più popolari (per esempio TH o IN, che sono molto popolari nella lingua inglese) si presentano sempre a distanze costanti. A questo ipotizziamo la grandezza della e otteniamo il testo trasposto.

Questo tipo di attacco però si può fare solo se la trasposizione è applica una sola volta. Se si usa più di una volta, allora diventa molto difficile da attaccare.

Sostituzione polialfabetica



Test crittoanalitici – Kasiski

Si usa per la sostiuzione polialfabetica soprattutto.
Consiste nella:

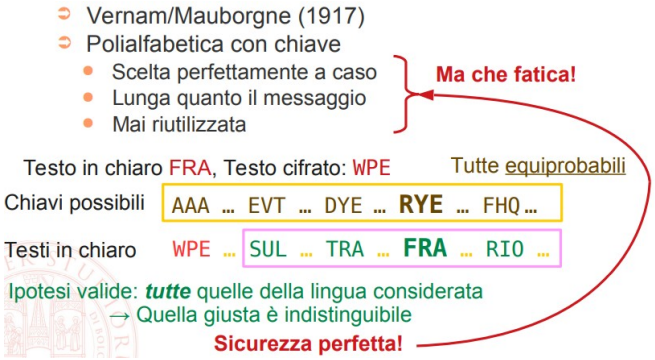
- ricerca nel cifrato di sequenze identiche
- annotazione delle distanze
- **fattorizzazione e scelta delle distanze con un fattore comune**
- lunghezza della chiave = MCD (massimo comun divisore)

Test crittoanalitici – Indice di coincidenza

Formalmente, **consiste nella probabilità che due lettere scelte a caso in un testo siano uguali**.
Usato per misurare le variazioni di frequenza delle lettere nel testo cifrato. Alta variazione è indice di basso "spargimento" → sostituzioni semplici.

One-time pad

La debolezza della sostituzione alfabetica sta nel fatto che la chiave si ripete. Ma se la chiave non si ripete, siamo a posto e viaaaaa.



Il one-time pad è un esempio di sicurezza perfetta, siccome tutte le chiavi sono equiprobabili (siccome sono generate in modo casuale) e quindi anche tutte le altre stringhe in chiaro possibili sono equiprobabili.

Viola la terza proprietà di Kerchoffs, siccome la chiave è lunga quanto il testo/cifrario stesso, e quindi non ha molto senso.

Lezioni del 6/04/2022 – Cifrari moderni: AES, DES, RSA

Esistono due tipi di cifrari: simmetrici e asimmetrici. A loro, **i cifrari simmetrici** (es. DES) sono suddivisi ind ue sotto-categorie: a blocchi e a flusso.

Cifrari a blocchi

Riprendendo i concetti sui cifrari classici, abbiamo alcune osservazioni da fare:

- Riguardo all'input: sappiamo che è possibile ridurre a priori la riconoscibilità statistica dei *simboli dell'alfabeto* in molti modi, tra cui:
  - Aumentandone il numero (frequenza media = 1/N) → facile se prendo come "lettera" un blocco di 8 bit, ma anche 16, 32, 64...)
  - Rendendoli equiprobabili (compressione).
- Riguardo all'algoritmo: ad ogni operazione di sostituzione e trasposizione aumenta la confusione e la diffusione.

Nei cifrari moderni simmetrici, abbiamo i cosiddetti **cifrari composti**, in cui abbiamo dei round di operazioni. In un round, si eseguono delle operazioni di trasposizione e sostituzione. Tanti round incrementano l'effetto (ma anche la complessità) di una cifratura. L'ideale sarebbe quindi rendere un singolo round molto efficace.

Cifrari di Feistel

Molti algoritmi di cifratura si basano in qualche modo su questa struttura, in cui:

- Si divide il plaintext in due parti. Una destra e una sinistra.
- La parte sinistra diventerà la parte destra del passo successivo, mentre la parte destra viene presa e sommata con la chiave.

La chiave è a 64 bit, ma di questi solamente 56 bit sono chiave vera, gli altri sono di checksum.

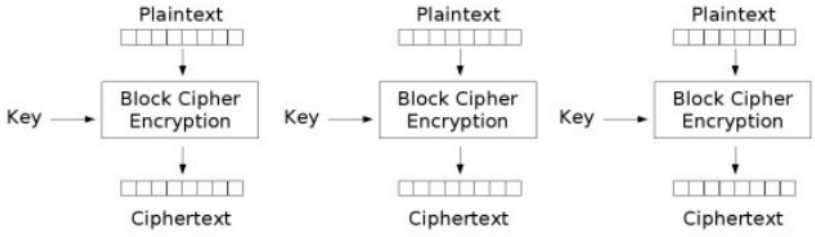
AES – Chiave simmetrica

AES è un algoritmo a chiave simmetrica, con peliculiari caratteristiche:

- Non usa la struttura di Feistel ma l'aritmetica dei campi finiti
- Fa uso di blocchi di 128 bit
- Può utilizzare chiavi di lunghezza diversa: 128 bit, 192 bit, 256 bit

Modi di operazione della cifratura

Ci sono molti modi in cui possiamo applicare la cifratura.
Il metodo peggiore consiste nell'applicare la cifratura blocco per blocco, avendo così una stringa singola formata da blocchi cifrati singolarmente. È male siccome un attaccante può capire subito se due stringhe sono uguali in questo modo.
Questa tecnica si chiaam **Electronic Codebook (ECB)**:



Electronic Codebook (ECB) mode encryption

In generale, l'idea migliore è quella di concatenare fra di loro i blocchi, in particolare usare una qualche caratteristica del blocco precedente per influenza la cifratura del blocco n+1.
Tra queste tecniche, abbiamo:

- **Cipher Block Chaining (CBC)**
- **Cipher Feedback (CFB)**
- **Output Feedback (OFB)**

Oppure, si realizzare l'equivalente a blocchi di un cifrario a flusso (es. Counter (CTR)).

Cifrari a flusso

Un esempio di cifrario a flusso è la sostituzione polialfabetica e la One-time pad, in cui si genera un flusso di chiave (ovvero una sequenza di bit o caratteri casuali) totalmente random e imprevedibile.
La chiave condivisa viene chiamata **seme**, dalla quale viene generato il flusso di chiave.

Funzioni Hash

Grazie alle funzioni hash, possiamo ottenere dei **fingerprint** (ovvero delle impronte digitali) delle informazioni che sono compatte e di dimenisione arbitraria. Un fingerprint ha una dimensione fissa, e la funzione è pubblica e conosciuta, senza che sia necessario l'uso di una chiave.

Le funzioni hash crittografiche sono robuste se:

- Non si può trovare un documento che abbia un fingerprint prefissato (proprietà di unidirezionalità, o one-way). Ovvero, non possiamo arrivare dal documento tramite il fingerprint.
- Non si può trovare una coppia di documenti che abbiano lo stesso fingerprint. (**collision-free**)

Sono due proprietà diverse, siccome una più facile dell'altra da attaccare.

Le funzioni hash ci permettono di difendere l'integrità del messaggio (ovvero, ci permettono di dire se il messaggio è lo stesso che è stato inviato, e che non è stato modificato durante il suo tragitto).

Non ci permettono, invece, di difenderci dagli attacchi man-in-the-middle. L'attaccante potrebbe benissimo prendere il nostro messaggio, e inoltrarlo/ripeterlo, oppure se il messaggio è in chiario può benissimo ricalcolare l'hash.
Ci basterebbe unicamente che l'hash sia mandato in un canale sicuro.

Attacchi agli algoritmi di hashing – **proprietà di one-way**

Con gli algortitmi di hashing, difficilmente riusciamo a trovare dei difetti nell'algoritmo in sé dal lato della proprietà di one-way.
Come sempre, possiamo usare un attacco di forza bruta, generando un sacco di documenti e vedendo se possiede la fingerprint che cerchiamo, ma ovviamente questo sistema ha complessità esponenziale.

Alcuni degli algoritmi di hash più diffusi sono:

- MD5, MD6, RIPEMD, SHA, SHA-3

Difficilmente, quindi, riusciremmo a ricavare da un hash il documento d'origine.

Attacchi agli algoritmi di hashing – **proprietà di collision-free**

La situazione è diversa invece con la proprietà di collision free. Infatti, **sia MD5 che SHA-1 hanno dei difetti algoritmici da questo lato**. Inoltre, se vogliamo attaccare questa proprietà, possiamo usare il paradosso del compleanno, che tramite una serie di calcoli del cazzo, ci dice che per trovare una coppia di documenti con lo stesso hash ci bastano $\mathbf{2^{m/2}}$ tentativi.

In questo modo, possiamo convincere ad una vittima che il messaggio inviato sia un invece altro. Es. posso far firmare a qualcuno una lettera di insulti, che invece io vario leggermente per renderla una lettera di lodi.

Altri attacchi: length extension

- Noto $H(m_1)$ e la lunghezza di m_1
- Senza conoscere m_1
- Scelto un m_2 dall'attaccante
- È possibile calcolare $H(m_1 \mathbin{@} m_2)$

Molto importante siccome in questo modo posso prendere un messaggio, attaccarci un'altra stringa alla fine, ricalcolare l'hash facilmente l'hash e convincere la vittima che quello era il messaggio originale.

Inizio dei sistemi asimmetrici – problemi difficili

I sistemi asimmetrici si basano su quelli che vengono chiamati problemi difficili. Tra questi abbiamo:

- Funzioni pseudo-unidirezionali
  - Operazioni facili in un verso e (speriamo) computazionalmente infattibili nell'altro
  - A meno di conoscere un segreto
- Fattorizzazione di grandi numeri
- Molte operazioni in aritmetica modulare
  - Numeri interi
  - Come risultato di un'operazione si prende il resto della divisione per un modulo fisso

RSA: algoritmo di cifratura asimmetrica

La generazione delle chiavi in RSA si basa con questi 3 passi:
Generazione delle chiavi:

1. si scelgono due numeri primi p e q
2. il modulo viene calcolato come n = p·q
3. si sceglie a caso un numero d e si calcola un numero ***e*** tale che ***e·d*** mod (p-1)(q-1) = 1. Questa operazione è facile solo conoscendo p e q, che vengono poi dimenticati

In questo modo, **la chiave pubblica è (e, n)**, **la chiave privata (d, n)**
Per cifrare: $c = m^e \bmod n$, per decifrare: $m = c^d \bmod n$

Robustezza

Non ci sono modi efficienti noti di invertire l'esponenziale modulare (la sua complessità è assimilabile alla forza bruta).
Ci sono algoritmi "quasi efficienti" per fattorizzare il modulo (es. General Number Field Sieve, sub-esponenziale). Per risolvere, usiamo dei moduli grandi (oltre 2048 bit).

Trappole:

- Non è dimostrabile che non esistano algoritmi classici efficienti (ma nessuno ha idea di come trovarli)
- Quantum computing
- Implementazioni troppo efficienti:
  - Spesso si sceglie e con pochi "1" (es. 3, 17, 65537)
  - Se troppo piccolo, m^e non "trabocca" da n!

Vantaggi della c. asimmetrica

- Per la riservatezza
  - È un cifrario a blocchi, di sostituzione, con dimensioni enormi
    - No forza bruta
    - No analisi statistica dell'"alfabeto" (salvo casi particolari)
  - Le chiavi usate per cifrare e decifrare sono diverse e dalla chiave pubblica non è derivabile la chiave privata
    - La chiave pubblica può essere distribuita
    - Chiunque può usarla per cifrare
    - La chiave privata corrispondente è l'unica che può decifrare
- La chiave privata è specifica di un solo utente quindi utile anche per autenticare

Poi vabbe, per la riservatezza delle informazioni, chi invia il messaggio lo cifra con la chiave pubblicca del destinatario, così questo potrà decifrarlo con la sua chiave privata, ma questo lo si sapeva già.

Posso comunque cifrare con la chiave privata, e lo uso per cifrare l'hash in modo da "**firmare**" il messaggio.

Vantaggi di RSA

Grandi vantaggi:

- distribuzione delle chiavi
- utilità per tutte le proprietà di sicurezza

Punti deboli:

- Prestazioni (5-10 volte più lento di AES)

- Sistemi ibridi
- alcuni attacchi specifici (known plaintext)

Aggiungere prestazioni e flessibilità
Per farlo, uso la crittazione ibrida, che fa uso di chiave simmetrica.

Ha fatto laboratorio di GPG (PGP non proprietario). Possiamo direttamente vedere la lezione dalle slides, siccome ci dicono esattamente le stesse cose in questo caso.

Lezioni del 8/04/2022 – Gestione delle chiavi

È necessario avere un metodo per la:
- Generazione
- Memorizzazione
- Distribuzione

delle chiavi.
Questi servizi ci vengono garantiti dalle PKI, ovvero le public key infrastructure.

Generazione delle chiavi

- Per le chiavi simmetriche, i nonce, i vettori di inizializzazione, i padding, … ci basta un buon generatore di numeri casuali
- Per le chiavi asimmetriche, ci servono dei *numeri primi*. Per generarli, si parte da numeri random, e si applica un test di primalità.

Generazione di numeri random

Un buon generatore di numeri random deve avere due caratteristiche principali:
- Casualità
- Non prevedibilità

- **True Random Number Generation**:
  - Si fa uso di sorgenti fisiche di entropia, che possono essere elementi ad hoc come rumore termico oppure eventi "imprevedibili" nel calcolatore, es. intervalli di arrivo degli interrupt dai dispositivi
- **Pseudo Random Number Generation**:
  - Sono deterministici
  - Se il risultato supera i test statistici, allora è accettabile come PRNG
  - Tipicamente hanno come input un seme (seed) **prodotto da TRNG**.
  - Noto il seme → nota la sequenza generata!

Il miglior attacco a RSA non è la forza bruta ma la ricerca dei fattori del modulo.

Memorizzazione

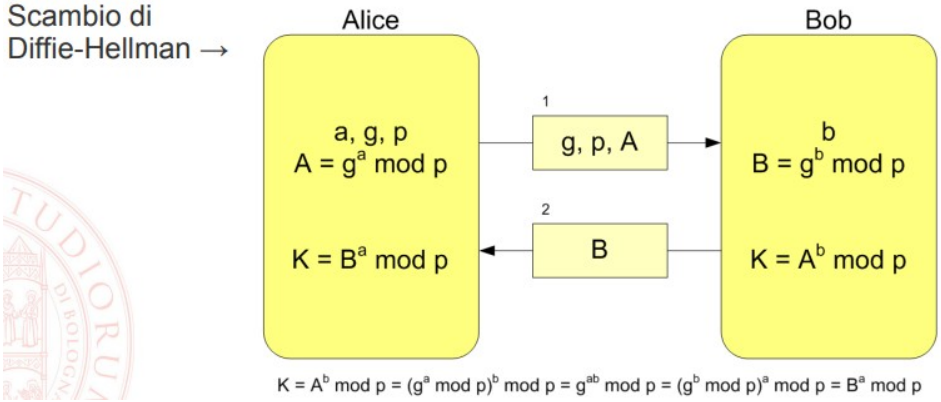
- Chiave di decifrazione: requisiti contrastanti
  - perdita devastante → backup (va comunque memorizzata da qualche parte, altrimenti come decifriamo?)
  - segretezza fondamentale → non diffusione
- Come fare quindi per memorizzare in segretezza?
  - Cifratura con passphrase
  - Hardware Security Module
  - Key escrow
  - Secret sharing

- Chiave di firma (si applica praticamente solo a chiave asimmetrica)
  - se compromessa si sostituisce, nessuno ha bisogno di recuperarla in assenza del titolare → nessun backup!
  - non deve essere usata contro la volontà del titolare → cifratura con passphrase, HSM

Distribuzione

Per le chiavi simmetriche, sappiamo che non devono mai essere esposte in chiaro. Ma come, dobbiamo lo stesso distribuirle per usarle. Per farlo, possiamo:
- Usare lo scambio manuale

- Usare un KDC (Key Distribution Center):
  - Ogni utente condivide una chiave con un centro fidato (Key Distribution Center)
  - Per stabilire una connessione, due utenti negoziano una chiave attraverso connessioni cifrate col KDC
- Si può usare lo **scambio** di Diffe-Hellman (NON È UNA TECNICA DI CIFRATURA, MA UNA TECNICA DI SCAMBIO!!!!!!)



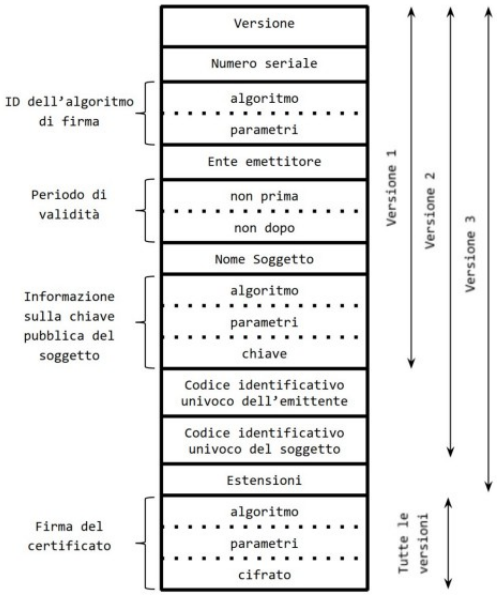
Un attaccante passivo non apprende nulla dalla visione di una chiave pubblica o dall'intercettazione dei parametri di DH, un attaccante attivo può invece sostituire i valori inviati da una parte all'altra coi propri. Ad esempio:

- RSA: l'attaccante ha una propria coppia di chiavi PRIVI e PUBi , e quando due utenti cercano uno la chiave pubblica dell'altro, ricevono invece PUBi
  - quando il mittente cifra i messaggi, l'attaccante li può decifrare, e con la chiave pubblica del destinatario legittimo, per ri-cifrarli per non insospettirlo (magari alterati)
  - l'attaccante può firmare messaggi con PRIVI e il destinatario, verificandoli correttamente con PRIVI si convincerà che siano del mittente legittimo
- DH: l'attaccante stabilisce due chiavi separate con A e B e continua a fare da "passacarte" senza insospettirli
- **Il problema quindi per i sistemi asimmetrici non è la riservatezza, ma l'autenticità dei dati pubblici ricevuti**

Certificazione delle chiavi pubbliche

Serve un modo per associare con certezza una chiave pubblica al suo legittimo titolare (nonché unico possessore della corrispondente chiave privata)

- Modello web of trust:
  - L'autenticità di una chiave pubblica è testimoniata da altri utenti
  - L'utente che riceve una chiave da uno sconosciuto può decidere di accettarla per autentica se è firmata da qualcuno fidato
  - Vantaggio: nessuna entità "super partes" di cui doversi fidare
  - Svantaggio: pessima scalabilità
- Modello infrastrutturale:
  - esiste una terza parte fidata che documenta l'associazione Certificati



Poi vabbe, il resto dei certificati lo so.

Lezioni del 13/04/2022 – Difesa delle reti: SSL, IPSec, OpenVPN

Contromisure contro gli attacchi sulle reti

HTTPS è un protocollo che combina essenzialmente due strati, che sono il livello applicaztivo e lo strato che si trova tra il livello applicazione e il livello di trasporto. Con HTTPS si intende HTTP over SSL.
Questo metodo permette di bloccare gli attacchi (in particolare quelli man-in-the-middle), ma esistono modi per:

- Evitare che venga visualizzato l'URL effettivamente visitato.
- Per far accettare al browser qualsiasi certificato.

Abbiamo poi altre contromisure che possiamo usare. Tra queste:

- Protezione a livello di rete: IPSec
- Protezione del data link layer
- Strumenti di uso comune: TOR e SSH

SSL (la nuova versione TLS – Transport Layer Security)

SSL è stato progettato come uno strato di protocolli indipendente. Si colloca logicamente fra strato di trasporto e applicazioni. Grazie a questa architettura non richiede una modifica dei protocolli di rete.

L'implementazione si presenta come una serie di funzioni di libreria. Per rendere una applicazione capace di suare SSL, è sufficiente inserire nel codice le chiamate a tali funzioni di libreria.

Il nostro sistema operativo di base non supporta SSL, ma appunto deve essere implementato a livello applicazione.
Dunque, se vogliamo rendere la comunicazione sicura, dobbiamo rendere le nostre applicazioni compatibili con SSL (non è quindi trasparente all'applicazione).

L'SSL record protocol funziona come un header con delle informazioni utili alla connessione, ma anche dati di servizio.

Sessione e connessione

- Due entità che colloquiano utilizzando SSL devono avere aperto una **sessione**
- Una **singola sessione** può includere **numerose connessioni sicure contemporanee**
- Due entità possono avere attive **numerose sessioni contemporanee**

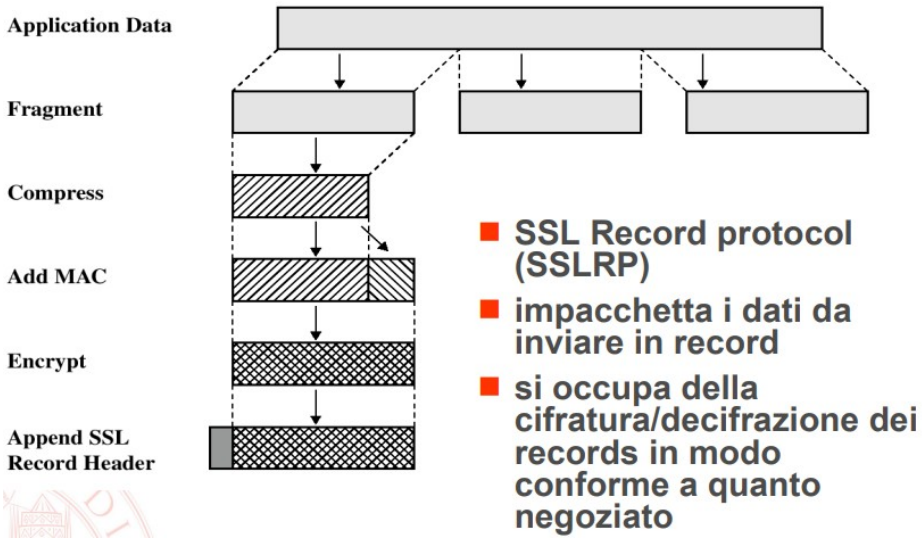
Architettura di SSL – Handshake Protocol

È un protocollo di stretta di mano che ricorda quello di TCP.

- **Consente al server ed al client di autenticarsi reciprocamente**
  - challenge response basato su crittografia asimmetrica e certificati X.509
  - nelle applicazioni web è comune che solo il server provi la sua autenticità al client
- Negozia gli algoritmi e le chiavi per la cifratura ed i controlli di integrità
- Interviene prima che qualsiasi dato sia trasmesso
- Progettato per limitare il carico di elaborazione e di rete
  - Implementa un meccanismo di caching delle sessioni per limitare il numero di aperture e quindi il carico di elaborazione
  - Tenta di ridurre al minimo l'attività sulla rete (scambio di messaggi)
  - E' possibile negoziare un numero di sessione: se la comunicazione si interrompe temporaneamente, vengono poi recuperati i parametri della sessione senza rinegoziarli.

Il client può anche chiedere al server di mandare la sua firma, tramite un meccanismo di challenge & response.

Architettura di SSL – record protocol



I dati dell'applicazione vengono spezzati in pezzi più piccoli, detti frammenti. Ciascun frammento viene poi compresso, si aggiugne un MAC (Message Authetication Code), si critta e infine si aggiungono gli SSL record Header.

SSL → TLS

Transport Layer Security è l'evoluzione di SSL. **Ha lo stesso formato di record.**
Simile a SSLv3, ma differenziato in:

- numero della versione
- codice di autenticazione del messaggio
- funzione pseudocasuale
- codici di avviso
- suite di cifratura
- tipi di certificato client
- certificate_verify e messaggio finito
- calcoli crittografici
- padding

Come fa un utente a sapere quali certificati sono effettivamente leciti (aka effettivamente di Amazon per esempio)? Si tenga a mente che il browser contiene al suo interno una serie di certificati sia di delle CA che sono "fidate".

Attacchi omografici

Si usano degli URL che sembrano esteticamente degli altri URL leciti (come quello di paypal), ma che in realtà usano degli altri caratteri di altre codifiche (es. cirillico).
Per evitare questi attacchi, si usa **punycode**, che praticamente consiste nella conversione dei caratteri internazionali in sequenze di caratteri base.
Il browser così può usare punycode, e far visualizzare l'URL in punycode al posto della codifica nel font internazionale.

Iniezione di CA nel certificate store

Ci potrebbe essere un malware che inietta delle CA nel certificate store, e iniettando una CA che è falsa, che permette di certificare qualsiasi altro sito che lui vuole. Tuttavia, per farlo deve avere accesso alle chiavi private della macchina.

Fake certificates

Il modo più "semplice" di impersonare un sito è falsificare il certificato. Si parla in questo caso di **rogue certificates**, e portano alla compromissione di una CA. Tuttavia è molto complesso da svolgere, e ci si può difendere usando delle CA di cui ci fidiamo molto e basta.

Comunque, nel caso in cui un certificato venisse falsificato, oppure una CA venisse violata, abbiamo lo stesso dei modi per difenderci:

- HTTP Public Key Pinning
  - limita le chiavi associabili a un dominio (root, intermediate, o end-entity)
  - dichiarato nell'header HTTP: Public-Key-Pins, Public-Key-Pins-Report-Only
  - È però deprecato dal team di Google Chrome, siccome era comunque violabile tramite man-in-the-middle.
- Certificate Transparency (CT – RFC 6962)
  - un framework aperto per scrutinare il processo di rilascio dei certificati;
  - È una specie di sistema peer2peer usata per scambiarsi delle informazioni sulla legittimità della certification authority.
  - In questo modo, il proprietario del dominio può vedere il certificato di quel dominio.

HTTP Stripping

Siccome la parte di inizializzazione della sessione HTTPS era la parte più intensiva/costosa della connessione, essenzialmente si faceva in modo che le pagine che non contenevano contenuti sensibili usassero una comunicazione HTTP, mentre quelle invece più sensisbili usassero HTTPS. E un MITM poteva benissimo arrivare, modificare la pagina e poi pigliarsi tutti i dati che voleva.

Per difendersi, si usa:

- HSTS (HTTP Strict Transport Security)
  - policy implementata dai server per forzare i browser a interagire via HTTPS
  - è però inefficace sulla prima richiesta, siccome sta nell'header di risposta.

Tuttavia...

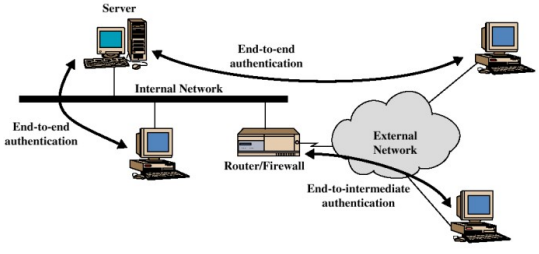
- Attacchi simili per i quali non c'è un equivalente di HSTS: STARTTLS command injection
  - connessioni che partono insicure e chiedono l'upgrade – MITM interferisce o accoda comandi in ordine errato

Heartbleed

Implementazione errata nella libreria OpenSSL della heartbeat extension che causa un vulnerabilità. In pratica, permette di leggere dei pezzi della memoria del serverweb e in questo modo posso rubare dei dati importanti dal server (es. la chiave privata del server).
Questo perché essenzialmente, per vedere s eun client era ancora connesso, si inviava una parola al server dicendo anche di quanti caratteri questa fosse. Se si chiedeva una parola di dimensioni molto minori rispetto a quanto dichiarato, allora venivano inviati anche degli altri dati della memoria.

Virtual Private Network

Reti virtualmente private = metodi di trasporto del traffico.
Essenzialmente, ci si connette fra due reti sopra la rete pubblica, ma si da l'illusione alle due reti di stare comunicando sulla stessa rete privata e sicura.
Garantiscono sicurezza, virtualmente perché il traffico è convogliato attraverso reti insicure



IP Security

Il termine VPN adesso ha una connotazione totalmente diversa. Infatti ora con VPN si intendono delle reti che sono totalmente anonime e sicure. Con VPN adesso si intende adesso essenzialmente l'accesso attraverso un intermediario a internet.

IPSec non è un protocollo singolo:

- set di algoritmi di sicurezza
- framework per la negoziazione degli algoritmi
- specifiche per la gestione delle chiavi

Il vantaggio principale di IPSec è quello di essere essenzialmente totalmente invisibile alla applicazioni, siccome lavora solo sullo strato di rete. Essenzialmente, quindi, richiede uno stack TCP/IP modificato, e quindi il nostro kernel deve supportare appunto l'uso di questo sistema operativo.

Applicazioni di IPSec:

- Interconnessione di sedi remote attraverso Internet
- Accesso di client alla rete aziendale attraverso Internet
- Creazione di reti complesse con criteri di protezione differenziati

Vantaggi di IPSec

- Trasparente alle applicazioni
- Applicabile al traffico infrastrutturale di Internet, come i messaggi che i router si scambiano per aggiornare le tabelle di instradamento

Il pacchetto però deve essere contrassegnato in modo da indicare il supporto a IPSec.
SOLO I ROUTER DEVONO ESSERE SPECIALI E SAPER PARLARE IPSEC.
[Oppure, se un road warrior, allora dovra avere proprio un PC che supporta IPSec per connettersi da qualsiasi punto al router con IPSec anche se sta su una rete insicura per esempio.]

IPSec inoltre, siccome si trova sotto il livello trasporto, non suppone che ci sia sotto di esso una connessione affidabile, e quindi implementa lui stesso i controlli dell'integrità dei messaggi.

Inoltre supporta:

- Controllo dell'accesso
- *Integrità anche senza connessione*
- Autenticazione dell'origine dei dati
- *Rilevazione dei replay*
- Riservatezza dei dati
- Parziale riservatezza dei flussi di traffico

Terminologia di base

- SA (**Security Association**)
  - relazione unidirezionale tra mittente e destinatario, definita da
    - Security Parameter Index (SPI)
    - IP Destination address
    - Security Protocol Identifier
  - due modalità possibili di SA
    - Transport Mode (viaggia sulle rete interna)
    - Tunnel Mode (viaggia su internet)
- Protocolli di sicurezza
  - AH (Authentication Header)

    - ESP (Encapsulating Security Payload)
- Instradamento
    - Le SA vengono attivate da tabelle specializzate del sistema operativo, nel caso di Linux le extended route (eroute)

Authentication Header e Encapsulating Security Payload

	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. Autentica il payload del pacchetto ma non l'header IP	Cifra ed autentica l'intero pacchetto IP interno.

Considerazioni comparative

- SSL/TLS
    - Contro: è specifico di un dominio applicativo
    - Pro: è semplice e realmente standard
- IPSec
    - Pro: è generale e trasparente alle applicazioni
    - Contro: è tipicamente implementato nello stack TCP/IP del sistema operativo, con variazioni che rendono difficile l'interoperabilità
- Soluzioni "ibride"
    - utilizzo di varianti di SSL per il trasporto di pacchetti IP analogo al tunnel mode di IPSec
    - implementazione user space, indipendente dal S.O. (Es: OpenVPN)

OpenVPN

OpenVPN riproduce con software in user space i concetti di transport e tunnel mode di IPSec (li mima).
Serve comunque un piccolo componente kernel space: la generazione di interfacce di rete virtuali, rispettivamente di tipo *tap (transport)* e *tun (tunnel)*

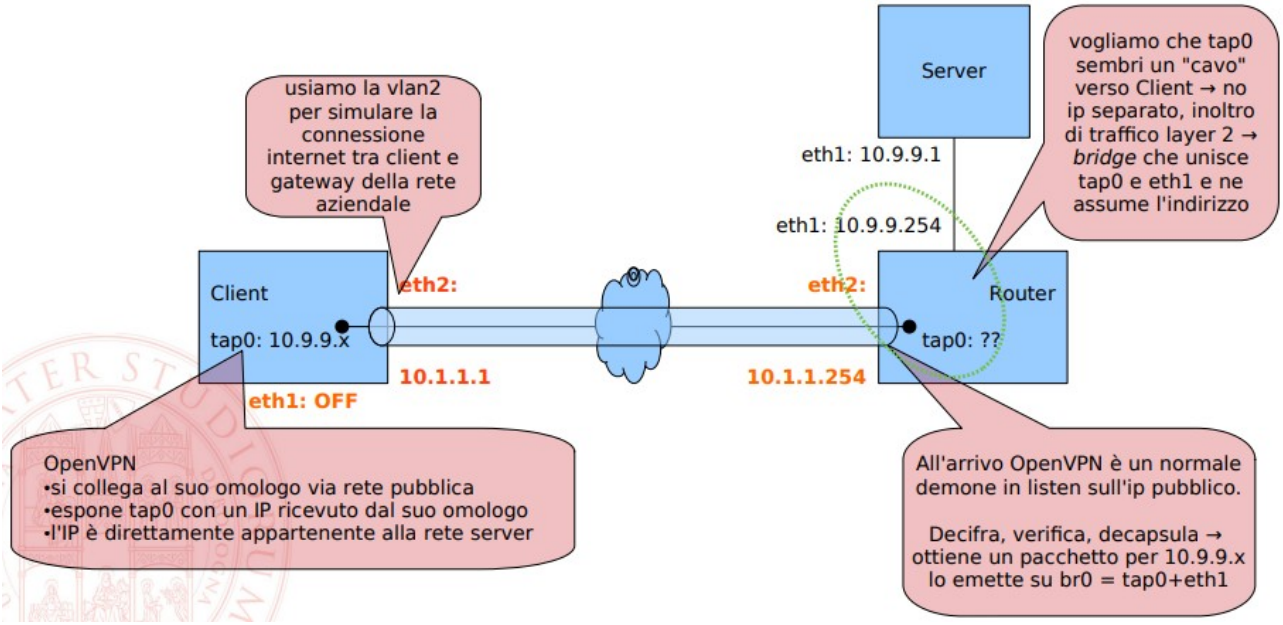
- queste interfacce si usano esattamente come quelle reali
- i pacchetti inviati a un'interfaccia reale sono inviate al device driver della scheda hardware
- i pacchetti inviati a un'interfaccia virtuale sono inviati al processo che le ha create
- l'uso o meno di queste interfacce è determinato da normali entry nella routing table dell'host

Il tunnel mode è molto semplice:

- Il router ha una interfaccia di rete virtuale (che chiamiamo tun0 per esempio) che è strettamente legata al processo di OpenVPN
- Quando il router riceve un pacchetto, arriva al router che riceve il pacchetto tramite l'interfaccia virtuale, e poi applica le varie modifiche (tipo crittazione, autenticazione etc) e lo manda poi all'interfaccia di rete collegata con l'esterno. In questo modo poi arriva al router che supporta OpenVPN, che applica le varie cose e poi lo invia al destinatario vero.

Come si vede, l'interfaccia tun è un puro artificio per creare una connessione punto-punto tra i due gateway mediata da OpenVPN.
Dal punto di vista delle applicazioni, gli indirizzi delle interfacce *tun* sono trasparenti e non appartengono a nessuna delle subnet effettivamente utilizzzate da client e server.
Per rendere una macchina remota virtualmente parte di una rete locale si ricorre al transport mode, tipicamente associato al **bridging**.

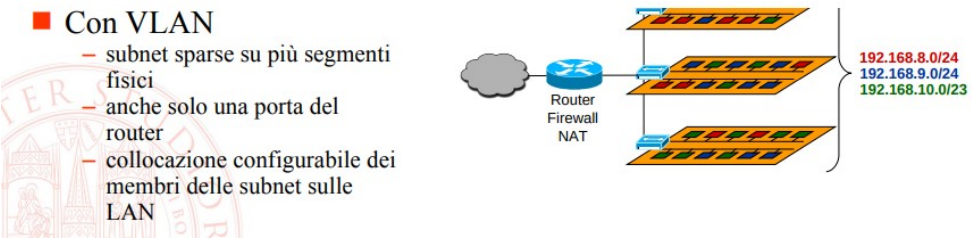
Il transport mode di OpenVPN cambia indirizzi IP (al contrario del transport mode di IPSec), siccome in realtà "simula un tunnel".



Il mio PC con OpenVPN infatti assumerà un IP compatibile con VLAN della mia azienda,

Data Link security

Gli switch possono gestire **Virtual LAN**: segregano il traffico tra differenti subnet in modo che gli host di una non vedano il traffico delle altre anche se fisicamente condividono parte dell'infrastruttura



VLAN – classificazione

- VLAN statica o port-based
  - ogni porta di uno switch appartiene a una o più VLAN
  - un host può appartenere a una o più VLAN, solo in base alla porta dello switch a cui è connesso
  - per spostare un host da una VLAN a un'altra, bisogna riconfigurare la porta dello switch
- VLAN dinamica
  - un host appartiene a una o più VLAN in funzione del proprio MAC o IP, indipendentemente dalla porta dello switch a cui è connesso (MOLTO ATTACCABILE)
  - per spostare un host da una VLAN a un'altra, bisogna riconfigurare la mappatura indirizzo-VLAN

Posso anche aggiungere un **tag** alla VLAN, in modo da poter dare l'accesso a più VLAN a un solo dispositivo. Possiamo inoltre trasportare pacchetti di VLAN diverse in modo riconoscibile fra router in questo modo.

VLAN – modi di funzionamento delle porte

Porte in access mode

- appartengono a una singola VLAN
- tagging dei pacchetti non necessario
- tipico uso per connessione di semplici host

Porte in trunk mode

- appartengono a più VLAN
- tagging necessario per distinguere a che VLAN appartiene ogni pacchetto
- possono essere configurate simultaneamente per gestire pacchetti untagged, che vengono considerati appartenenti a una VLAN nativa + pacchetti tagged di altre VLAN
- tipico uso per connessione a router o tra switch

Attacchi a VLAN

Switched spoofing:

- Prerequisito: vittima che accetta riconfigurazione con **Dynamic Trunking Protocol** – DTP.
- l'attaccante fa credere di essere lui stesso uno switch, e configura la vittima in modo che la porta a cui è connesso venga impostata come trunk su cui far passare tutte le VLAN.

Double Tagging:

- Prerequisito: catena di switch con gestione non attenta di tag annidati, e attaccante legittimamente connesso alla VLAN "V1" nativa del trunk che interconnette gli switch
- l'attaccante invia a un host sulla VLAN "V2" un pacchetto costruito ad arte perché appaia come se fosse annidato: con tag V2 incapsulato in un tag V1
- il primo switch rimuove il tag V1 e inoltra il pacchetto a tutte le porte access o native che appartengono a V1, incluso il trunk
- il secondo switch riceve il pacchetto e lo interpreta come appartenente a V2, quindi lo inoltra all'host vittima
- **NOTA: funziona solo in andata, non c'è modo di forzare la vittima a generare una risposta in modo da riceverla**

SSH

SSH è un protocollo, che usa gli stessi principi TLS, ma senza avere i certificati. Non sappiamo quindi quando la chiave pubblica è effettivamente quella giusta. Non fa uso di un sistema di certificati come SSH, o almeno, ci sono, ma sei tu che devi stabilire se puoi fidarti o no del sistema.

SSH inoltre può essere usato al volo per creare delle VPN (-L).
Permette inoltre di creare una buca per firewall (-R), essenzialmente permetto a una macchina che si trova dietro a un firewall di collegarsi ad un gateway pubblico esterno in modo che questo endpoint pubblico esponga una porta 110.
Permette inoltre di creare delle catene per salti (-J), in modo simile a Tor quasi.

Lezioni del 20/04/2022 – Laboratorio su OpenVPN e SSH

[la prima parte della lezione è stata la configurazione di una rete virtuale, molto interessante, ma non utile per l’esame, quindi non lo scrivo. Still, potrebbe servire per dopo]

[1.30]

Lezioni del 22/04/2022 – Difesa delle reti 2: Firewalls

Firewall

È simile a una cinta muraria di un castello con un solo portone, che divide ciò che avviene "fuori" con ciò che avviene "dentro".

Il firewall è compostod a uno o più componenti hardware o software, ed è un punto di passaggio obbligato (infatti è efficace solo se non ci sono altre strade per accedere alla rete da proteggere).
Passa solo quello che è esplicitamente autorizzato.
La sua robustezza è molto importante. Il sistema dev'essere immune agli attacchi. È quindi un sistema dedicato, in cui sia possibile rinunciare a flessibilità e praticità in favore della riduzione delle vulnerabilità.

Tecniche di controllo

Col firewall possiamo controllare:

- ***Traffico***
  - Esaminare indirizzi, porte, e altri indicatori del tipo di servizio che si vuol rendere accessibile
- ***Direzione***
  - Discriminare a parità di servizio le richieste entranti verso la rete interna da quelle originate da essa (N.B.: il traffico è sempre composto da uno scambio bidirezionale di pacchetti, la direzione logica di una connessione è definita da chi prende l'iniziativa)
- ***Utenti***
  - Differenziare l'accesso ai servizi sulla base di chi lo richiede (**controllo degli accessi**). N.B.: nel protocollo TCP/IP non c'è traccia dell'utente responsabile della generazione di un pacchetto!
- ***Comportamento***
  - Valutare come sono usati i servizi ammessi, per identificare anomalie rispetto a parametri di "normalità"

Tipi di firewall

- Tre tipi fondamentali
  - Packet filter
  - Application-level gateway
  - Circuit-level gateway

- Due collocazioni particolari
  - Bastion host
  - Personal firewall

Tipi di firewall: packet filter (PF)

Esamina unicamente l'header del pacchetto, e svolge le decisioni in base a questo. Ad esempio:

- **Link layer**:
  - Interfaccia fisica di ingresso o uscita
  - MAC address sorgente / destinazione
- **IP layer**:
  - Indirizzi sorgente / destinazione
  - Protocollo trasportato (ICMP, TCP, UDP, AH, ESP, …)
  - Opzioni IP (ECN, TOS, …)
- **Transport layer**:
  - TCP flags (SYN, ACK, FIN, RST, …)
  - Porte sorgente / destinazione

Un PF applica in serie un elenco di regole del tipo “se condizione allora azione”.
Normalmente la prima trovata in cui il pacchetto soddisfa la condizione determina il destino del pacchetto e interrompe la scansione dell’elenco.

Le azioni di base sono scartare o inoltrare il pacchetto. Ce ne sono poi altre comunemente implementate:

- Loggare i dettagli del pacchetto
- Modificare in qualche modo il pacchetto

Se nessuna regola viene attivata, si applica una politica di default (scartare o inoltrare il pacchetto).
Normalmente le regole sono raccolte in più liste separate, corrispondenti a punti di controllo diversi (es. per i pacchetti in ingresso al firewall e quelli in uscita).

Vantaggi

- Semplice e veloce. È implementato tipicamente in tutti i router
- Trasparente agli utenti
  - Se il firewall coincide col default gateway di una subnet, per farlo attraversare non si deve riconfigurare nessun sistema.
  - Nell’implementazione locale a un sistema, può intercettare il traffico locale e reindirizzarlo a componenti user-space arbitrari

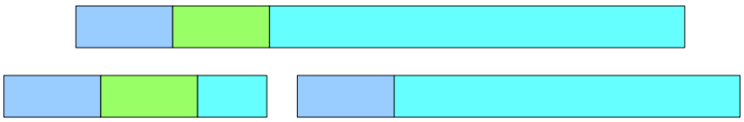
Svantaggi

- Regole di basso livello
  - Comportamenti sofisticati richiedono set di regole molto complessi
- Mancanza di supporto alla gestione utenti
  - Negli header non compaiono elementi identificativi

Vulnerabilità e contromisure (parziali) del PF

→ Frammentazione

- Frammenti successivi al primo non possono attivare condizioni che menzionano parametri dell’header di trasporto (es. porta sorgente/destinazione, TCP flags) → evasione
- Molti altri attacchi basati su vulnerabilità dei riassemblatori
- Soluzione drastica: scartare i pacchetti frammentati
- Soluzione costosa: riassemblare sul firewall (non implementabile su packet filter puro)



L’immagine mostra un pacchetto in cui la parte verde sarebbe l’header di trasporto.
Questo, se il pacchetto viene frammentato, si trova solo sul primo pacchetto per esempio.

→ Spoofing (falsificazione degli indirizzi del mittente)

- Ci possiamo difendere facendo un controllo di coerenza tra subnet e **interfacce**/configurazione.
  - Multicast (224.0.0.0/4) se non utilizzato
  - Provenienti da “fuori” con IP sorgente della rete “dentro” e v.v. (Impossibile su router infrastrutturali)

→ Source routing (instradamento determinato dal mittente). Ormai ignorato da tutti i router.

Limitazioni del PF

Se non si introduce un livello di vera e propria analisi del protocollo applicativo, il filtraggio stateful **non può gestire protocolli che negoziano dinamicamente le connessioni e/o le porte**.
Es. FTP, streaming protocols per multimedia.

Inoltre, il firewall PF, siccome non va a vedere il contenuto del payload, **NON PUÒ PROTEGGERE CONTRO ATTACCHI DATA-DRIVEN** (ovvero dovuti a dati nel payload).

Tipi di firewall: PF

- Formalmente un PF è *stateless*, ovvero non ha memoria del traffico passato. Decide su ogni pacchetto solo sulla base delle regole.

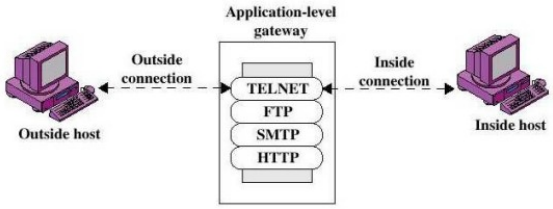
- Evoluzione: **PF stateful.** Ha memoria di qualche aspetto del traffico che vede passare. Può decidere su di un pacchetto riconoscendolo parte di un flusso di traffico già instaurato.
  - Implementazione specifica del tipo di PF
  - Utile soprattutto per protocolli senza connessione

- Evoluzione: **Multilayer protocol inspection firewall**
  - Tiene traccia dell'intera storia della connessione per verificare la coerenza del protocollo
  - In alcuni casi anche oltre il livello di trasporto

Firewall basato su Application-Level Gateway

Anche chiamato ***proxy server***. In questo ruolo può svolgere anche altre funzioni, es. caching.

Un ALG è 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.
Questo si mette in mezzo alla connessione in modo del tutto **NON TRASPARENTE**.



Vantaggi

- Comprende il protocollo applicativo, quindi permette filtraggi avanzati come
  - Permettere/negare specifici comandi
  - Esaminare la correttezza degli scambi protocollari
  - Attivare dinamicamente regole sulla base della negoziazione C/S
- Sono integrabili con processi esterni per l'esame approfondito del payload, es:
  - Antispam/antivirus per la posta, Antimalware/antiphishing per il web
- Permette di tenere log molto dettagliati delle connessioni.

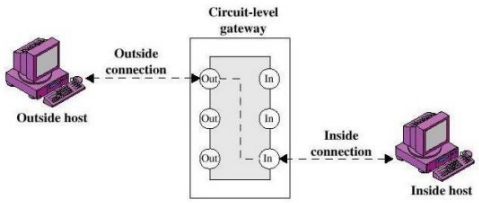
Svantaggi

- Molto più pesante di un PF
- Specifico di un singolo protocollo applicativo (quindi, se vogliamo comunicare con FTP, dobbiamo creare un nuovo proxy server).
- **Non sempre trasparente, può richiedere configurazione del client**

Firewall basati su Circuit-level gateway (CLG)

Spezzano la connessione a livello di trasporto

- Diventano endpoint del traffico, non intermediari
- Inoltrano i payload senza esaminarli



Sono essenzialmente dei proxy general purpose, che non esaminano i payload. Spezzo comunque la connessione però (ovvero, client parla col gateway, gateway parla con server...).

Dev'essere comunque configurato. È una sorta di firewall fatto al contrario, siccome viene impiegato soprattutto dall'interno verso l'esterno.

Vantaggi
- Può essere configurato **trasparentemente** agli utenti per autorizzare le connessioni da determinati host considerati fidati (di norma perl non lo è)
- Può agire da intermediario generico, senza bisogno di predefinire quali protocolli applicativi gestire
- Può essere usato in combinazione con le applicazioni per differenziare le politiche sulla base degli utenti

Svantaggi
- Le regole di filtraggio sono limitate a indirizzi, porte, utenti
  - Si può combianare con un PF per gestire più dettagli di basso livello, con un ALG per gestire più dettagli applicativi
- Richiede la modifica dello stack dei client
  - O la consapevole configurazione delle applicazioni

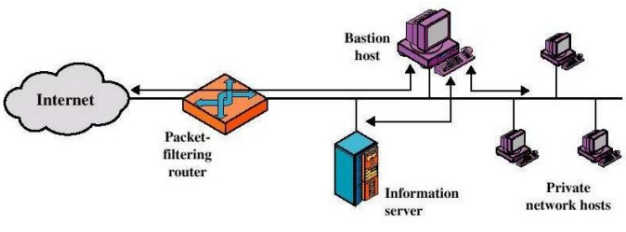
Dove posizionare il firewall?

→ **Bastion Host (BH)**
- Un sistema dedicato a far girare un software firewall, tipicamente per realizzare un ALG o un CLG
- Può servire anche per un PF, ma tipicamente questo è integrato nei router che servono la rete.

→ **Personal Firewall**
- Costituiscono un'eccezione al principio del controllo alla frontiera, essendo installati sulle singole macchine da proteggere.
- Vantaggi
  - Correlazione fra applicazione sorgente/destinazione e pacchetto → altissima precisione nel controllo di cosa è lecito vs. anomalo
- Svantaggi
  - Perdita della centralizzazione della configurazione (o necessità di utilizzare sistemi di deploy piuttosto invasivi)
  - Spesso configurati "learning by doing" → molti alert → ignorati

I firewall non sono adatti a reti in cui siano presenti contemporaneamente client e server.
Infatti i client dovrebbero unicamente generare traffico uscente, e devono essere totalmente schermati dagli attacchi esterni.
I server invece devono ricevere selettivamente traffico dall'esterno, possono essere più facilmente compromessi e non devono poter essere usati per attaccare i client.

Topologie – **screened single-homed BH**
Abbiamo un PF garantisce che solo un BH possa comunicare con l'esterno.
Il BH implementa un ALG (eventualmente con autenticazione).



In questo modo abbiamo un doppio filtraggio:

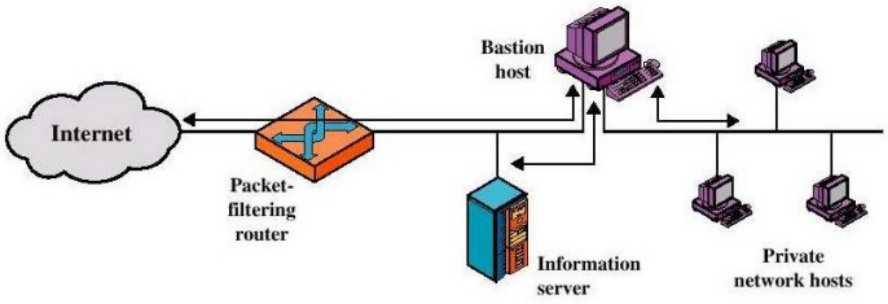
- a livello header (PF)
- a livello applicativo (BH)

Dunque, se un hacker vuole prendere il controllo completo della rete interna, avrà due sistemi da compromettere. (Tuttavia, per un accesso significativo è sufficiente compromettere il PF).

In questo modo è semplice fornire accesso diretto a server totalmente pubblici.

Topologie – **screened dual-homed BH**

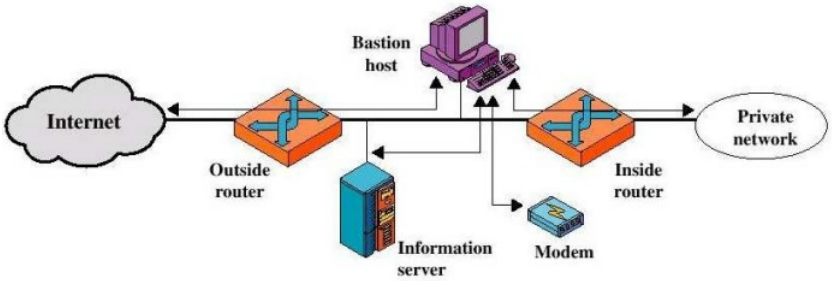
Come prima, ma il BH separa fisicamente due segmenti di rete.
La compromissione del PF non dà accesso alla rete interna. Si crea una zona intermedia detta **"demilitarizzata" (DMZ)** (I server sono collocati qui, siccome se qualcuno prende in controllo del server, comquneu per entrare nella rete privata dovrà compromettere il BH).
Svantaggio: tutto il traffico dai client deve fluire attraverso il BH, anche quello del tutto innocuo.



Topologie – **screened subnet**

L'uso di due PF router

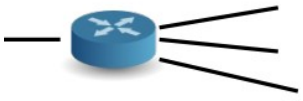
- Rafforza la separazione tra esterno e interno
- Nasconde completamente all'esterno l'esistenza della subnet privata, ostacolando l'enumerazione da parte degli attaccanti
- Nasconde l'esistenza di Internet alla rete privata, ma consente ai router di inoltrare il traffico "banale" senza passare dal BH



Topologie – variazioni sul tema

Altri di topologie sono:

- Sacrificando il doppio livello di protezione, se si dispone di un PF molto affidabile o di poco budget:
  - Si possono unificare le funzioni di R1 e R2 della topologia screened subnet
  - **con >3 interfacce si possono realizzare diverse DMZ**



È però un single point of failure

- Al contrario, se si deve gestire con elevata sicurezza una topologia di rete caratterizzata da molte zone con esigenze di protezione via via più elevate, si possono concatenare in serie DMZ con vari PF. [MOLTO COSTOSO].

IPTables

iptables è il packet filter integrato nel kernel Linux.
Si appoggia sul framework netfilter

- definisce degli hook nello stack di rete del kernel. Un hook è un punto di aggancio generale in cui dico “quando il pacchetto è stato processato dal OS, si invocherà questo hook”.
- ogni pacchetto che attraversa lo stack di rete innesca gli hook
- si possono registrare programmi agli hook in modo da far eseguire controlli e manipolazioni sui pacchetti

Cinque hook in punti strategici dello stack di rete

- `NF_IP_PRE_ROUTING`
  - attivato da un pacchetto appena entra nello stack di rete. Questo hook viene elaborato prima di prendere qualsiasi decisione di instradamento riguardo a dove inviare il pacchetto.
- `NF_IP_LOCAL_IN:`
  - attivato dopo che un pacchetto in arrivo è stato instradato se il pacchetto è destinato al sistema locale.
- `NF_IP_FORWARD:`
  - attivato dopo che un pacchetto in arrivo è stato instradato se il pacchetto deve essere inoltrato a un altro host.
- `NF_IP_LOCAL_OUT:`
  - attivato da qualsiasi pacchetto in uscita creato localmente non appena raggiunge lo stack di rete.
- `NF_IP_POST_ROUTING:`
  - attivato da qualsiasi pacchetto in uscita o inoltrato dopo che l'instradamento ha avuto luogo e appena prima di essere messo in rete.

Più programmi possono registrarsi allo stesso hook, dichiarando un ordine di priorità

- invocati in ordine
- ognuno restituisce una decisione sul destino del pacchetto

iptables connesso a netfilter

iptables gestisce il traffico registrandosi agli hook.
Ci sono dei concetti fondamentali:

- **tabelle**
  - organizzano i controlli a seconda del tipo di decisione da prendere sul pacchetto
- **catene**
  - organizzano i controlli a seconda dell’hook a cui sono agganciate, quindi del momento in cui decidere cosa fare del pacchetto durante il suo ciclo di vita nel sistema
- **regole**
  - sono gli elementi costitutivi delle catene
  - espressioni del tipo “SE il pacchetto rispetta queste condizioni, ALLORA esegui questa azione”

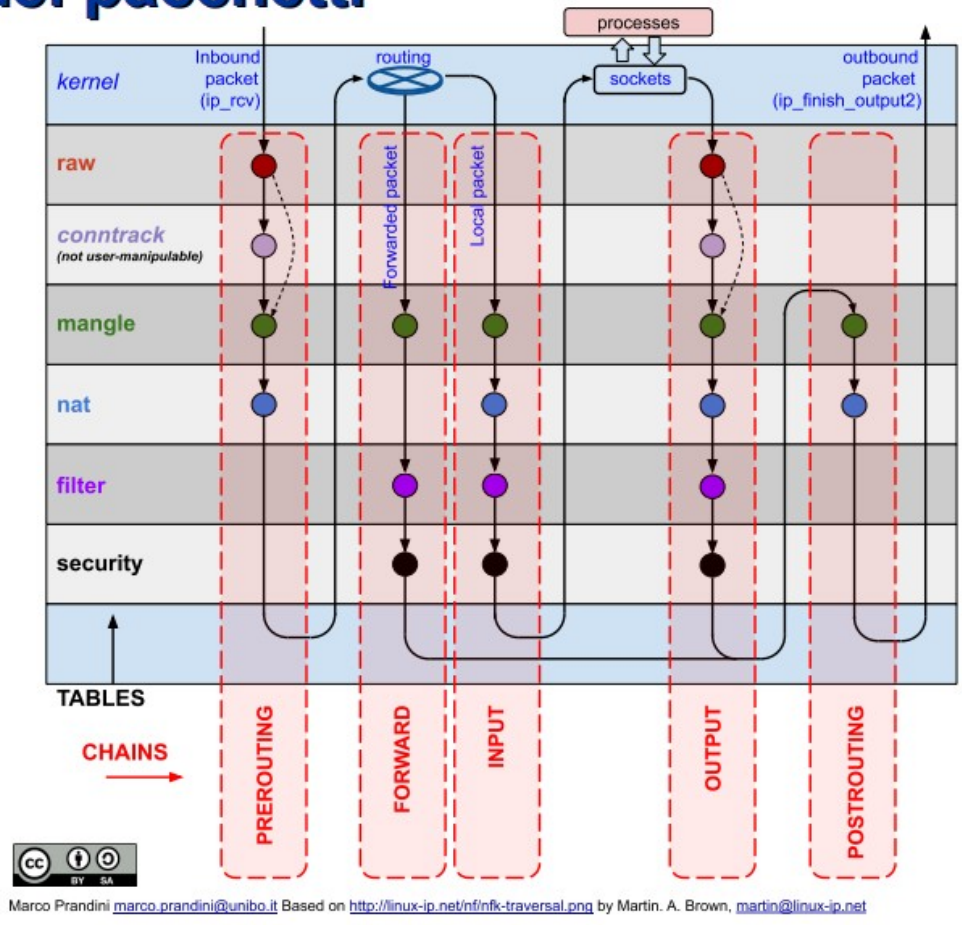
Le catene corrispondono esattamente agli hook di netfilter.

Le tabelle si suddividono in:

- **`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
- **`conntrack`**
  - implementa automaticamente (cioè con una logica non configurabile dall'utente) il riconoscimento delle connessioni e l'attribuzione dei pacchetti alle stesse
- **`filter`**
  - la tabella principale, utilizzata per decidere se lasciare che un pacchetto continui verso la destinazione prevista o bloccarlo.
- **`nat`**
  - utilizzata per implementare le regole di traduzione degli indirizzi di rete, modificando gli indirizzi di origine o di destinazione del pacchetto
- **`mangle`**
  - utilizzata per modificare l'intestazione IP del pacchetto (es. cambiare il valore TTL o qualsiasi altro campo), e può marcare la rappresentazione kernel di un pacchetto per renderlo riconoscibile da altre tabelle e da altri strumenti
- **`security`**
  - utilizzata per impostare i contrassegni di contesto di sicurezza SELinux interni sui pacchetti

Il percorso dei pacchetti

- Ogni pacchetto che entra nello stack di rete viene sottoposto all'esame di varie catene nell'ordine mostrato da questo schema
- Il percorso ha un inizio comune per i pacchetti di origine esterna (tutte le catene PREROUTING delle tabelle che la supportano)
- Il percorso si dirama a seconda che il pacchetto sia destinato a un processo locale (catene INPUT) o a un host remoto (catene FORWARD)
- I pacchetti di origine interna sono processati dalle catene OUTPUT
- I pacchetti di qualunque origine destinati a lasciare il sistema sono infine processati dalle catene POSTROUTING



Lezioni del 27/04/2022 – Laboratorio su IPTables

Piccola premessa

Spesso, in un firewall, siamo interessati a proteggere più o meno tutte le catene, siccome quelle di input potrebbero essere vulnerabili a esfiltrazioni o altro, poi ovviamente dobbiamo occupartci anche del post-routing e forwarding etc…

Nelle macchine che non sono dei firewall o che non si occupano dell'inoltro dei pacchetti, invece, possiamo anche dimenticarci delle catene di forwarding.

Manipolazione delle catene

Si usa il comando iptables

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

Il flag `-L` ci permette di vedere tutte le regole che sono presenti e attive in quel momento. Usando -nL, si ha un output numerico senza che i numeri vengano tradotti in nomi (credo per l'indirizzo IP).

Iptables ci mostrerà solamente 3 catene, siccome suppone che stiamo lavorando solamente nella tabella filter (e nella tabella filter ci sono unicamente quelle tre catene lì).

Se invece usassi `iptables -t nat -nL` vedrei solamente le catene PREROUTING, INPUT, OUTPUT E POSTROUTING. Se lo facessi con mangle, invece, vedrei tutte e 5 le catene.
Oppure volendo, possiamo specificare anche una sola catena, scrivendo il nome di questa. (es. `iptables -nL FORWARD`).

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

Layer su cui possiamo fare filtering - match di base sull'header IPv4
Possiamo fare match mettendo delle regole particolari, basandoci su un layer del pacchetto specifico.

- Caratteristiche "Layer 1"
  - -i <input interface>
  - -o <output interface>
- Caratteristiche "Layer 2" (solo caricando l'estensione con `-m mac`)
  - --mac-source <source mac address>
- Caratteristiche "Layer 3"
  - -s <source address>
  - -d <destination address>
  - -f (fa match coi frammenti dal secondo in poi)
  - Questi due solo caricando l'estensione con `-m iprange`:--src-range from-to e --dst-range from-to
- Caratteristiche "Layer 4"
  - -p <tcp|udp|udplite|icmp|icmpv6|esp|ah|sctp|mh>
  - se il protocollo supporta le porte, abilita l'interpretazione di
    - --dport port[:port]
    - --sport port[:port]
  - se il protocollo è tcp, abilita l'interpretazione di
    - --tcp-flags mask comp
      - i flag sono *SYN ACK FIN RST URG PSH ALL NONE*
      - mask = elenco flag "interessanti" (gli altri flag del pacchetto sono ignorati)

        - comp = elenco flag tra quelli interessanti che devono essere settati per fare match
- se il protocollo è icmp, abilita l'interpretazione di
    - --icmp-type <type>
    - elenco tipi: iptables -p icmp -h

Esempi di popolamento delle catene

La prima cosa che dobbiamo fare quando scriviamo una regola è PENSARE IN QUALE CATENA devo mettere la regola.

Supponiamo che vogliamo fermare i ping provenienti da un certo host: in quel caso, la catena su cui voglio scrivere la regola sarà la catena INPUT.
La regola che scriveremo sarà quindi (dentro R2):

```
iptables -I INPUT -i eth2 -s 10.12.12.10 -p icmp --icmp-type echo-request -j DROP
```

Questa regola aggiunge alla catena input il blocco di tutti i pacchetti che arrivano all'interfaccia eth2, con inp sorgente 10.12.12.10 e protocollo icmp (e tipo di icmp pari a ping).
`-j` indica il target, ovvero l'azione che andiamo ad eseguire se le condizioni si sono verificate (in questo caso, droppiamo il pacchetto).

DROP e REJECT fanno la stessa cosa, ma REJECT manda un messaggio al sender che gli spiega perché il pacchetto è stato droppato.

NOTA: tutto quello che scrivo nella riga è in AND logico! Ovvero, per far si che il TARGET venga eseguito, deve fare match con ogni regola.

Scriviamo ora la regola che dice che nessun pacchetto ICMP debba uscire dalla macchina:

```
iptables -I OUTPUT -p icmp -j DROP
```

Adesso creiamo una regola che permette ad un pacchetto di essere inoltrato da un host ad un altro, seguendo la nostra infrastruttura di rete che avevamo già programmato la settimana scorsa.

In questo caso, dobbiamo usare la tabella nat e inserire nella catena di postrouting un target SNAT che ci permette di cambiare l'indirizzo di 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
```

Per creare un sistema che non fa passare nulla tranne alcuni particolari pacchetti, dobbiamo impostare una "policy di default".
Questo lo possiamo fare usando un comando come:

```
iptables -P INPUT DROP
iptables -I INPUT -p tcp ! --dport 2000 -j ACCEPT
```

Oppure, possiamo usare l'operatore "!"

```
iptables -I INPUT -p tcp ! --dport 2000 -j DROP
```

In questo modo indichiamo che tutti i pacchetti tcp che arrivano alla porta 2000 devono essere buttati via.
Ha senso anche mettere il rispettivo del comando in OUTPUT:

```
iptables -I OUTPUT -p tcp ! --sport 2000 -j DROP
```

stateful filtering

- dal punto di vista del filtraggio, il connection tracking è molto utile perché permette di utilizzare gli stati dei pacchetti per raffinare i match
- per accettare solo pacchetti validi come iniziatori di una connessione (nella direzione lecita da un client verso un server) e seguenti
  `-m state --state NEW,ESTABLISHED`
- per accettare solo pacchetti validi come risposte a una connessione già iniziata (nella direzione lecita da un server verso un client) e seguenti
- `-m state --state ESTABLISHED`

Usare iptables come logger granulare

```
iptables -A INPUT -j LOG --log-prefix “ input_drop ”
```

Essenzialmente, con -A indichiamo che vanno messe in fondo alla catena, e solo così il log funzionerà.

Eliminare delle regole specifiche
In generale, possiamo eliminare delle regole specifiche nella nostra tabella in questo modo.
Prima di tutto, dobbiamo listare tutte le regole che sono presenti nella nostra catena:

```
iptables -vnL --line-numbers
```

Dopo che abbiamo scoperto il numero della regola che vogliamo elimanare, usiamo il comando

```
iptables -D 1 INPUT
```

dove 1 è il numero della regola, e INPUT è la catena in cui vogliamo eliminare la regola.

Possiamo creare anche delle catene costum, ma sono troppo stanco per scrivere cosa fanno.

Lezioni del 29/04/2022 – Autenticazione e Inizio Autorizzazione

La regola AAA

- L'**autenticazione** si occupa di attribuire un'identità certa ad un soggetto che sta utilizzando delle risorse. Questa include una identificazione preliminare.
- L'**autorizzazione** verifica i diritti di un soggetto che sta per compiere una determinata azione su un oggetto. Si ha quindi una decisione esplicita di concessione o negazione di un dato permesso.
- L'**auditing** consiste nel tracciamento affidabile delle decisione di autenticazione e autorizzazione.
  - Permette di verificare l'efficacia delle politiche.
  - È un compromesso difficile fra utilità e usabilità.

Autenticazione

Si basa principalmente su uno di questi fattori, ovvero qualcosa che l'utente:

- **Conosce** (ad es. Password, PIN, risposta segreta)
- **Possiede** (ad es. Carta bancomat, telefono cellulare, hard token, Yubikey)
- **È (fisicamente)** (ad es. Biometrico: iride, impronta digitale, ecc.)
- **È (posizione)** (ad es. GPS, geolcation, Centralizzata Auto, Allarme in casa.)

Usiamo i termini Prover per indicare quello che deve confermare la sua identita e Verifier per indicare colui che la deve verificare.

Autenticazione passiva

- Autenticazione passiva
  - P e V concordano il segreto e lo memorizzano
  - P invia il segreto a V per dimostrare di conoscerlo
  - V lo confronta con la copia in suo possesso per autenticare P

Questo porta a una serie di potenziali problemi… :

- Problemi di comunicazione
  - invio in chiaro → intercettazione da parte di attaccante passivo
  - invio offuscato ma sempre uguale → replay attack
  - servono protocolli di autenticazione più sofisticati – ... oppure un canale cifrato bene
- Problemi di memorizzazione
  - **furto da parte di V**

Memorizzazione delle password

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

  Dunque, non si memorizza la password ma la sua impronta, calcolata con una funzione hash. Questo porta comunque ad una serie di problemi...

- Problemi
  - attacco con dizionario
  - stessa password su macchine diverse
- Mitigazione:
  - **salt** – variazione random inserita alla scelta della password

Esempio di salt (usato in `/etc/shadow`):

```
$6$ViDM2ItuaSNPBxfO$Ip40UoquO.OiFafqIVeMqzZplisBV.CC76j
Ryead9rvidbyWcAr200dH.7N8budnpS3FzJJdDxrKGQjekwTFU0

identificatore dell'algoritmo hash usato
salt
fingerprint calcolato su concatenazione pass||salt
```

A ogni rinnovo o cambio di password, anche il salt cambia. In questo modo, password uguali hanno delle fingerprint diverse.

Rimane comunque inefficace contro attacchi offline, cioè avviati dopo aver sottratto il file delle password . La contromisura essenziale per questi è che la password non sia facile da indovinare.

Inoltre, bisogna sempre usare delle password diverse.

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 intrinsecamente inutile
- Il furto dal canale è inutile per autenticazioni future
- attenzione comunque all'uomo nel mezzo!

S-KEY One-Time Password

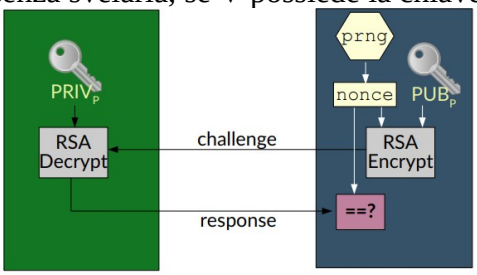
- P conosce il proprio segreto N
- V viene inizializzato col risultato dell'applicazione ripetuta k volte di una funzione hash a N: $h^k(N)$

- Alla prima autenticazione P invia $h^{k-1}(N)$
  - V verifica facilmente che $h(h^{k-1}(N)) = h^k(N)$
  - V scarta $h^k(N)$ e ricorda $h^{k-1}(N)$ come riferimento per la prossima autenticazione

Il sistema però dovrà essere reinizializzato dopo k passi.

Challange & Response

Tipicamente, questo sono utilizzati con crittografia asimmetrica. Ad esempio, P può provare il possesso di una chiave privata senza svelarla, se V possiede la chiave pubblica:



2-Factor Authetication (2FA)

Siccome l'autenticazione a un solo fattore è debole (es. email & password), siccome possono essere facilmente svelate a seguito per esempio di attacchi vari o altro, si può utilizzare l'autenticazione a due fattori, che aggiunge un ulteriore livello di autenticazione che impedisce agli aggressori di accedere a un account anche se ottengono delle credenziali (per esempio).

Essenzialmente, si fa uso di due dei quattro fattori precedenti (es. carta + pin).

È importante che per fare la 2FA i fattori di autenticazione siano DISTINTI.

2SA vs 2FA

Al giorno d'oggi infatti la 2FA viene erroneamente confusa con la **two-steps**, **dove il livello aggiuntivo di autenticazione non è riconosciuto come distinto**.

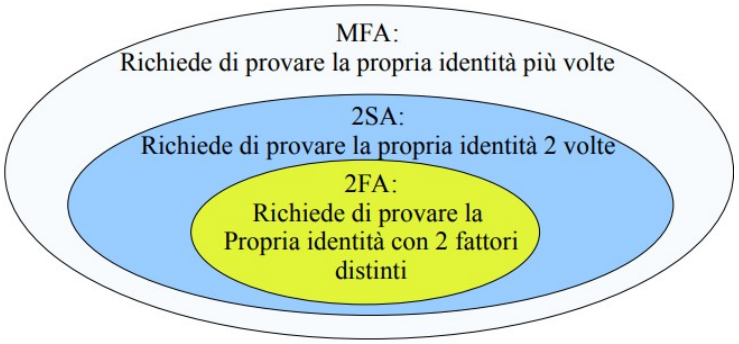
Es: un codice di verifica inviato per SMS o per mail in aggiunta ad una password non è considerato qualcosa che si POSSIEDE (in aggiunta a qualcosa che sa) perché l'SMS è la mail sono "facilmente" oggetto di attacchi MITM.

Un altro esempio, sbloccare un'app con un PIN per avere un token di accesso aggiuntivo è considerato come "conoscenza aggiuntiva" per cui non un fattore di autenticazione aggiuntiva.

Il device per l'autenticazione aggiuntiva deve essere dedicato solo a quello scopo. (Il telefono può essere violato da remoto invalidando il concetto del "possiede").

MFA

Abbiamo poi la Multiple Factor Authentication, che si ha quando si usano più di due fattori, anche distinti, vengano usati per l'autenticazione.



OTP vs TOTP

- La **one-time password** è una password *aggiuntiva*, sottoforma di token è valida solo per un utilizzo.
- La **timed one-time password** è una password aggiuntiva, sottoforma di token è valida solo per un utilizzo ed è limitata nel tempo.

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 su siti web e servizi mobile.
- **UAF (Universal Authentication Framework)**
  - Progettato come sostituto dell'autenticazione di base
  - Tipicamente coinvolge la biometria in cui le informazioni di sicurezza non lasciano mai il dispositivo
- **U2F (Universal Second Factor)**
  - Rafforza e semplifica 2FA utilizzando dispositivi USB, NFC o Bluetooth
  - Ha come scope una forte protezione da phishing, dirottamento di sessioni, attacchi man-in-the-middle e malware
  - Supporto nativo offerto dai principali fornitori e browser

FIDO UAF

FIDO UAF supporta la possibilità di autenticazione senza password.

- È stato rilasciato come standard aperto dall'alleanza FIDO.
- In questo standard, un utente che si autentica su un'applicazione o un servizio sfrutterà uno o più fattori di sicurezza sul proprio dispositivo digitale (solitamente un telefono cellulare) per rilasciare una chiave privata che viene utilizzata per firmare una sfida emessa dal server FIDO UAF.
- Il meccanismo di verifica dell'utente sul dispositivo stesso può essere biometrico, basato sulla conoscenza o sul possesso per sbloccare la chiave privata per le funzioni di firma.

FIDO U2F

- U2F è uno standard di autenticazione aperto che consente agli utenti di accedere in modo sicuro a *qualsiasi servizio online* con una singola chiave di sicurezza, istantaneamente e senza bisogno di driver o software client. FIDO2 è l'ultima generazione del protocollo U2F.
- U2F è stato creato da Google e Yubico, con il supporto di NXP, con l'intento di portare una forte crittografia a chiave pubblica nel mercato di massa. Oggi, le specifiche tecniche sono controllare dal consorzio del settore dell'autenticazione aperta FIDO Alliance.
- U2F è stato implementato con successo da servizi su larga scala, tra cui Facebook, Gmail, Dropbox, GitHub, Salesforce.com, il governo del Regno Unito e molti altri

Esempio pratico di 2FA

U2F per USB la YubiKey – Componente hardware per 2FA attraverso OTP, e crittografia a chiave pubblica con supporto: • Multi-protocol support; FIDO2/WebAuthn, U2F, Smart card, OpenPGP, OTP • USB-A, USB-C, NFC

Autorizzazione

Un soggetto autenticato deve essere autorizzato a svolgere operazioni sulle risorse del sistema. Abbiamo dunque 3 passaggi che dobbiamo definire:

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

Come già detto è molto utile separare politiche e meccanismi

- per confrontare diverse politiche senza essere sommersi dai dettagli di implementazione
- per modellare i componenti al livello di astrazione più appropriato e identificare la serie minima di requisiti che qualsiasi sistema di controllo degli accessi dovrebbe rispettare
- per progettare meccanismi come elementi costitutivi, utilizzabili per diversi tipi di politiche

Caratteristiche delle politiche

- **Principio del privilegio minimo**: qualsiasi accesso deve avvenire concedendo l'insieme di autorizzazioni più ristretto possibile.
- **Consistenza** (coerenza)
  - deve esistere uno schema di risoluzione non ambiguo da impiegare quando è possibile applicare autorizzazioni diverse alla stessa richiesta di accesso

    - nessuna soluzione unica! la regola più / meno specifica vince
    - default allow vs. default deny
    - vince la prima / ultima regola incontrata in ordine spazio / temporale
    - gerarchia degli autori delle regole

- Completezza e correttezza
    - qualsiasi richiesta di accesso deve ricevere risposta entro un limite di tempo predeterminato
    - ci deve essere una regola predefinita da applicare quando non è possibile trovare un'autorizzazione esplicita per una richiesta

Caratteristiche dei meccanismi

- Resistenza alle manomissioni: deve essere impossibile sabotare un meccanismo di controllo degli accessi senza che nessuno se ne accorga
- **Principio di mediazione completa**: ogni accesso alle risorse deve essere sottoposto al controllo e alla decisione del meccanismo
- Piccolo e autonomo: deve essere facile da testare e riparare
- Ragionevolmente economico: il suo costo non deve superare i danni causati da accessi non autorizzati

Modelli di controllo degli accessi

I due paradigmi fondamentali sono:

- **DAC (Discretionary access control)**:
    - Ogni oggetto ha un proprietario
    - Il proprietario decide i permessi
- **MAC (Mandatory access control)**:
    - La proprietà di un oggetto non consente di modificarne i permessi
    - C'è una policy centralizzata decisa da un security manager

Ci sono modelli più complessi:

- RBAC (Role-based access control):
    - I permessi sono assegnati ai ruoli
    - Utile se i soggetti possono assumere dinamicamente ruoli differenti a seconda del contesto (cosa devono fare, dove si trovano, in che tempi operano…) – e varianti…

Generalità sui meccanismi/implementazioni

In prinicipio, si usava una matrice degli accessi per decidere se un oggetto potesse eseguire una specifica operazione su un dato oggetto. Tuttavia, questo metodo era molto costoso e inefficiente.

Implementazioni comuni: ACL e capabilities

- Partizionare la matrice per soggetto: **capability lists**
    - **Una lista associata a ogni soggetto del sistema**
    - Contiene solo gli oggetti su cui il soggetto ha permessi ≠ default

- Partizionare la matrice per oggetto: **access control lists (ACL)**
    - **Una lista associata a ogni oggetto del sistema**
    - Contiene solo i soggetti che hanno permessi ≠ default sull'oggetto
    - Esplicitamente implementata da POSIX e MS Windows
    - Il filesystem Unix tradizionale ha negli inode una ACL "rigida", che elenca sempre e solo tre soggetti:
        - L'utente proprietario (U)
        - Il gruppo proprietario (G)
        - Il gruppo implicito che contiene tutti gli utenti ≠ U e G $\notin$ e i relativi permessi

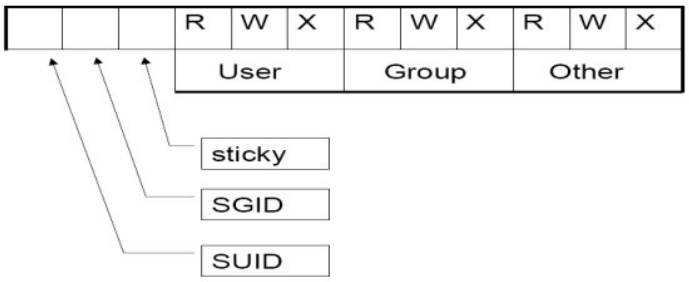
DAC in Linux

In Linux, ogni utente deve appartenere ad almeno un gruppo, ma non c'è un massimo.
Gli account utente possono essere in stato ***locked***, che impedisce di usareli per l'accesso interattivo, ma consente comunque ai processi di usare tale identità.

Con `passwd` possiamo cambiare la password e settare questo lock.

Autorizzazioni su Unix Filesystem

- 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
  - (esattamente un) utente proprietario del file
  - (esattamente un) gruppo proprietario del file
  - Un set di 12 bit che rappresentano permessi standard e speciali



Significato dei bit di autorizzazione

Leggermente diverso tra file e directory, ma in gran parte deducibile ricordando che:

- Una directory è semplicemente un file
- Il contenuto di tale file è un **database** di coppie (nome, i-node)

R = read (lettura del contenuto)
Lettura di un file
Elenco dei file nella directory

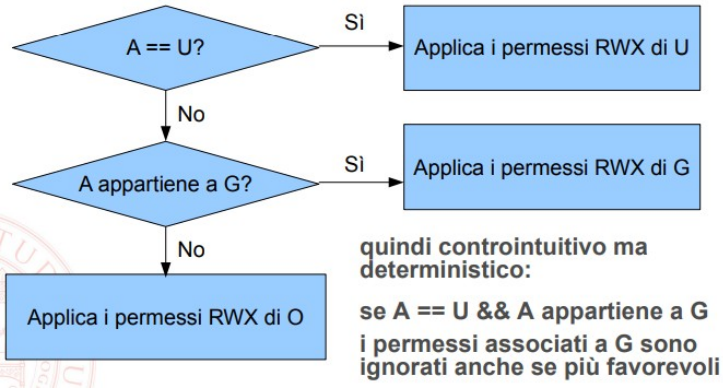
W = write (modifica del contenuto)
Scrittura dentro un file
Aggiunta/cancellazione/rinomina di file in una directory

NOTA che il permesso 'W' in una directory consente a un utente di cancellare file sul contenuto dei quali non ha alcun diritto

X = execute
Esegui il file come programma
Esegui il lookup dell'i-node nella

NOTA: l'accesso a un file richiede il lookup di tutti gli i-node corrispondenti ai nomi delle directory nel path → serve il permesso 'X' per ognuna, mentre 'R' non è necessario

I permessi dei file vengono applicati seguendo questo schema di regole:



Controllo dei permessi predefiniti

Si fa uso di degli automatismi per assegnare i permessi alla creazione:

- Ownership
  - l'utente creatore è assegnato come proprietario del file
  - Il gruppo attivo dell'utente creatore è assegnato come gruppo proprietario
    - Default = gruppo predefinito, da /etc/passwd
    - L'utente lo può cambiare a mano nella sessione con newgrp

!!! umask != chmod!!! umask toglie i permessi inseriti, mentre chmod li aggiunge!

Bit speciali per i file

I tre bit più significativi della dozzina (11, 10, 9) configurano comportamenti speciali legati all'utente proprietario, al gruppo proprietario, e ad altri rispettivamente:

- BIT 11 – SUID (Set User ID)
  - Se settato a 1 su di un programma (file eseguibile) fa sì che al lancio il sistema operativo generi un processo che esegue con l'identità dell'utente proprietario del file, invece che quella dell'utente che lo lancia
- BIT 10 – SGID (Set Group ID)
  - Come SUID, ma agisce sull'identità di gruppo del processo, prendendo quella del gruppo proprietario del file
- BIT 9 – STICKY
  - OBSOLETO, suggerisce al S.O. di tenere in cache una copia del programma

Per trovare i file aventi SUID e/o SGID settati, si usa il comando:

```
find / -type f -perm /6000
```

Bit speciali / per le directory

- Bit 11 per le directory non viene usato
- Bit 10 – SGID
  - Precondizioni
    - un utente appartiene (anche) al gruppo proprietario della directory
    - il bit SGID è impostato sulla directory
  - Effetto:
    - l'utente assume come gruppo attivo il gruppo proprietario della directory
    - I file creati nella directory hanno quello come gruppo proprietario
  - Vantaggi (mantenendo umask 0006)
    - nelle aree collaborative i file sono automaticamente resi leggibili e scrivibili da tutti i membri del gruppo
    - nelle aree personali i file sono comunque privati perché proprietà del gruppo principale dell'utente, che contiene solo l'utente medesimo
- Bit 9 – Temp

    - Questo bit settato a 1 impone che nella directory i file siano cancellabili solo dai rispettivi proprietari.

Attributi
Gli attributi sono primariamente utili per il fs tuning.

POSIX Access Control Lists
Le ACL estendono la flessibilità di autorizzazione.
Vantaggi:

- Specificare una lista arbitraria di utenti e gruppi coi relativi permessi (comunque scelti tra rwx) in aggiunta agli owner
- Ereditare la maschera di creazione dalla directory
- Limitare tutti i permessi simultaneamente (esempio mask sotto).

Possiamo usare `setfacl` per impostare, `getfacl` per visualizzare le ACL.

Capabilities in Linux

(da non confondere con le capability list tipiche dei modelli MAC)

- I poteri di root non sono “monolitici”
- rappresentano autorizzazioni normalmente negate agli utenti standard
- riguardano molteplici aspetti di controllo delle risorse di calcolo e dei processi e dell’accesso alla rete
- una nello specifico è CAP_DAC_OVERRIDE: la possibilità di ignorare i permessi sul filesystem.

È possibile assegnare specifiche capability a processi lanciati da utenti standard, in modo da autorizzare processi a svolgere azioni privilegiate senza accesso a root.
In questo modo si implementa il principio di minimo privilegio.

Lezioni del 04/05/2022 – Continuazione Autorizzazione

DAC nei sistemi Microsoft

Nell'uso più comune, i sistemi Microsoft sono raggruppati in un dominio: un insieme di computer, comunicanti tra loro e che condividono un directory database comune.

Nella directory sono memorizzati vari tipi di oggetti

- I computer che fanno parte del dominio
- Le risorse condivise dai computer (cartelle, stampanti)
- Gli utenti validi sul dominio
- I gruppi di utenti
- I raggruppamenti di altre entità
- …

Utenti

Local user accounts

- ristretti al sistema su cui sono creati
- possono avere moderati permessi amministrativi (che non si estendono alla possibilità di accedere ai dati di altri utenti) --> Power Users Group

Domain user accounts

- appartiene ad un dominio
- profilo memorizzato in AD
- può accedere a risorse non locali, limitatamente ai privilegi che gli sono concessi
  - del proprio dominio
  - dei domini trusted

Proprietà dell'utente

Sono moltissime, accessibili dai tab del wizard qui elencati:

– Member Of	The user's defined group membership
– Dial-in	Remote access and callback options
– General	User's first name, last name, display name description, office location, telephone, e-mail, and Web pages
– Address	User's post office mailing address
– Account	Logon name, domain, logon hours, logon to server name, account options, and account expiration date
– Profile	User profile path, profile script, home directory path and server, and shared document folder location
– Telephones/Notes	Home, pager, and mobile phone numbers and comments on where to contact user
– Organization who	Job title, company, department, manager, and people report to user Environment Applications to run from Terminal server client
– Sessions	Timeouts for Terminal Services
– Remote Control	Permissions for monitoring Terminal Service sessions
– Terminal Service Profile	Location for Terminal Service home directory

Gruppi in windows

Si suddividono in:

- **Distribution Groups**
  - possono essere usati da qualsiasi applicazione abbia bisogno di una lista di utenti (sono letteralmente delle naming lists)
  - il sistema operativo non li utilizza
    - non appesantiscono il logon ticket dell'utente

- **Security Groups**
  - come i DG, ma possono essere soggetti nelle regole che controllano l'accesso alle risorse del sistema.

- In Windows 2003 funzionante in Native Mode è possibile la conversione da un tipo all'altro.

Group scopes

- Sia per i Distribution Group che per i Security Group vale il concetto di **scope** (estensione), che definisce:
  - oggetti di quali domini possono far parte del gruppo
  - in quali domini può essere usato un gruppo per definire regole d'accesso

- I gruppi possono essere:
  - Machine Local (locali ad una singola macchina)
  - oppure validi in un dominio, e li suddividiamo in:
    - Domain Local
    - Global
    - Universal

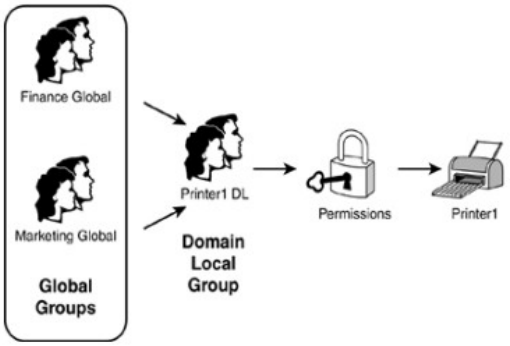
- Nesting: è possibile solo in native mode (sistemi più vecchi) rendere gruppi membri di altri gruppi

	Può contenere	Può essere membro di	Gli possono essere assegnati permessi su
Domain Local Group (DLG)	Utenti, GG, UG, computer di qualsiasi dominio, DLG dello stesso dominio	Altri DLG dello stesso dominio	Risorse dello stesso dominio
Global Group (GG)	Utenti ed altri GG dello stesso dominio	Qualsiasi DLG e UG, GG dello stesso dominio	Risorse di qualsiasi dominio
Universal Group (UG)	Utenti, GG, UG di qualsiasi dominio	DLG e UG di qualsiasi dominio	Risorse di qualsiasi dominio

Utilizzo tipico e consigliato

Sebbene sia possibile assegnare diritti su risorse direttamente a UG e GG, la struttura consigliata è (caso più semplice):

- individuare in ogni dominio utenti con esigenze analoghe e metterli in un GG
- rendere i GG membro degli opportuni DLG (con esigenze analoghe)
- assegnare i permessi d'uso delle risorse ai DLG



Possiamo poi usare gli UG per permettere l'assegnazione di privilegi per realtà ancora più grandi.

Group Policy

- **Group Policy** fornisce un quadro di riferimento per controllare l'ambiente di utenti e computer, cioè per assegnare quel tipo di privilegi o restrizioni che non sono legati a risorse fisiche ovvie quali file, cartelle, ecc.
- Le regole vengono definite in un **Group Policy Object**, che può essere collegato a qualsiasi contenitore di oggetti (una OU, un Site, un Domain) per applicarle a tutti gli oggetti in esso contenuti. [Permette di applicare di una politica molto meno granulare di una ACL, che posso applicare ovunque]
- Ogni GPO contiene due sezioni distinte
  - impostazioni per gli utenti (user settings)
  - impostazioni per i computer (computer settings)

- In ciascuna delle due sezioni le impostazioni sono ulteriormente classificate in:
  - impostazioni software (software settings) (es. aggiornare, installare, rimuovere applicazioni...)
  - impostazioni di Windows (Windows settings) (es. script di startup, sicurezza, shutdown, logon..)
  - modelli per l'amministrazione (administrative templates) (per definire in modo centralizzato le impostazioni del registro di sistema).

ACL in windows

Le ACL sono disponibili su partizioni Windows, in modo da assegnare ad una risorsa una lista di soggetti che hanno dei permessi su quel dato oggetto.
In Windows, per modificare le ACL è necessario:

- o detenere l'ownership
- o che nell'ACL siano assagnate il **privilegio** di "change permission" o "full control"

Ownership ed autorizzazioni

- Ownership
  - L'Owner di files e directories ha il pieno controllo (Full Control)
  - Administrator può sempre prendere l'ownership
  - L'Owner può assegnare le permissions per prendere l'Ownership
  - Nota: gli utenti che creano un file o una directory ne detengono l' Ownership

- Autorizzazioni NTFS predefinite
  - **Ad Everyone viene assegnato automaticamente Full Control**
  - I nuovi file ereditano le autorizzazioni della cartella in cui vengono creati (questo vale anche per i files che vengono copiati in un direttorio)

Accesso ed auditing

- Le ACL per il controllo dell'accesso fanno sì che, ad ogni tentativo di utilizzo di una risorsa, il sistema risponda autorizzando o negando l'operazione
- Ad ogni risorsa è inoltre associata una **SACL** utilizzata per l'auditing, che si presenta come una normale ACL, ma permette di tracciare gli esiti dei tentativi di utilizzo
- Le regole nella SACL possono essere impostate in modo che se l'accesso ad una risorsa da parte di un soggetto viene bloccato o permesso, questo evento sia registrato.

Autorizzazioni standard e speciali

L'obiettivo del sistema di controllo delle autorizzazioni è duplice

- elevata precisione nel controllo dell'accesso
  - in termini di tipo di azioni da concedere/negare
  - in termini di gestione delle complesse relazioni tra utenti e gruppi che possono essere titolari delle autorizzazioni
- facilità d'uso
  - il sistema è Discretionary Access Control (DAC), quindi consente ad ogni utente anche non tecnico di manipolare le autorizzazioni sulle proprie risorse
  - anche l'utente più esperto per il 90% del tempo fa cose semplici

La soluzione ai due problemi è realizzata con un sistema che prevede tre strati di interfaccia utente per “svelare” all'occorrenza i dettagli che servono:

- a basso livello il sistema supporta
  - molte autorizzazioni (autorizzazioni speciali) --> possibilità di controllo fine sui permessi
  - con una logica a tre valori (allow, deny, not set) --> possibilità di definire regole di interazione quando diverse ACL vengono combinate
- le autorizzazioni speciali sono aggregate in un set più ridotto di autorizzazioni standard
- le autorizzazioni standard possono essere visualizzate a due valori (allow, not set) o mostrando esplicitamente i tre valori

Ogni permesso (usando le ACL) può essere impostato in tre modi

- esplicitamente allow
- esplicitamente deny
- o non impostato

Windows segue un modello default deny: ciò che non è esplicitamente consetito è proibito.
Un utente però può appartenere a molti gruppi!

- Ogni gruppo può avere permessi distinti nell’ACL di una risorsa
- il permesso complessivo dell’utente sarà la somma, bit a bit, di tutti i permessi ottenuti in quanto membro dei propri gruppi
- non impostato (sia allow che deny sono bit a zero) presente nei permessi di un gruppo consente che un allow ottenuto da un altro gruppo possa avere effetto: non impostato == deny “debole”/scavalcabile
- un deny esplicito prevarrà sempre su di un eventuale allow ottenuto da un altro gruppo: deny esplicito == deny “forte”/non scavalcabile

Autorizzazioni di accesso alle cartelle e file

Sulle cartelle è possibile impostare una delle seguenti autorizzazioni standard:

- Nessun accesso
- Elenco
- Lettura
- Aggiunta
- Aggiunta e Lettura
- Modifica
- Controllo completo

Nota: I gruppi o gli utenti a cui è stata concessa l'autorizzazione ‘Controllo completo’ su una cartella sono in grado di eliminarne i file, indipendentemente dall'autorizzazione che li protegge.

Sui file è possibile impostare le seguenti autorizzazioni standard:

- Nessun accesso
- Lettura
- Modifica
- Controllo completo

Un breve cenno a MAC – Mandatory Access Control

MANDATORY: le regole di controllo degli accessi sono dettate da un'autorità centrale e i soggetti non possono modificarne alcun dettaglio.
Ad ogni soggetto o risorsa viene assegnata **una classe di accesso** che specifica tipicamente

- un livello di sicurezza all'interno di un insieme ordinato di valori, ad es {TopSecret ► Segreto ► Riservato ► Non classificato}
- una categoria (compartment) all'interno di un insieme non ordinato, ad es. {armi, piani di battaglia, unità di combattimento, …} che riflette le aree funzionali del sistema

Come usare le classi

- le risorse sono etichettate con un livello di sicurezza (sensitivity) che rappresenta la gravità delle conseguenze di una violazione delle policy che le riguarda
- i soggetti sono etichettati con un livello di sicurezza che rappresenta la loro affidabilità: clearance
- le categorie sono utilizzate per raffinare le politiche
- vengono stabilite relazioni di dominanza tra classi; detti
  - S un livello di sicurezza
  - C un insieme di categorie
  - Es. L1 = ‹S1, C1›, L2 = ‹S2, C2› L1 domina L2 se e solo se S1≥ S2 e C2 ⊆ C1

Le relazioni di dominanza vengono usate in modo diverso a seconda della proprietà di sicurezza da proteggere

- **riservatezza → Bell-LaPadula model** (BLP)
- **integrità → Biba model**

L'applicazione simultanea dei due modelli è possibile assegnando due classi di accesso a ogni soggetto e risorsa, una usata per controllare la riservatezza e l'altra per controllare l'integrità

BLP (Bell-LaPadula)

Due regole per proteggere la riservatezza:

- **NO-READ-UP**: un soggetto può leggere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti leggerebbe una risorsa troppo sensibile per il suo livello)
- **NO-WRITE-DOWN**: un soggetto può modificare una risorsa solo se la sua classe di accesso *è dominata* dalla classe di accesso della risorsa (altrimenti potrebbe far trapelare un segreto in luoghi accessibili a soggetti con minor clearance)

Biba

Due regole per proteggere l'integrità:

- **NO-READ-DOWN**: un soggetto può leggere una risorsa solo se la sua classe di accesso è dominata dalla classe di accesso della risorsa (altrimenti utilizzerebbe informazioni meno attendibili al proprio livello di fiducia più elevato)
- **NO-WRITE-UP**: un soggetto può scrivere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti modificherebbe una risorsa troppo sensibile per il suo livello)

Un brevissimo cenno a RBAC

Le autorizzazioni non sono concesse a utenti, ma a ruoli.
Il ruolo ricoperto da un utente può cambiare dinamicamente – nel tempo – secondo il contesto.

Lezioni del 06/05/2022 – IDS

Finalità di un IDS

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

Termini importanti dell'IDS

- **IDS** = Intrusion Detection System
  - è genericamente un sistema in grado di rilevare tentativi di attacco
    - signature based (riconosce attacchi noti)
    - anomaly detection (riconosce deviazioni dall'uso standard)
- **IPS** = Intrusion Prevention System
  - semplificando: 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
- Parametri di qualità del rilevamento degli eventi
  - Falso positivo (FP): segnalazione di attacco errata da evento innocuo
  - Falso negativo (FN): attacco reale che non genera una segnalazione

NIDS, HIDS, EDR

- Due strategie di rilevazione
  - Basate sulla rete (NIDS / Network-based IDS)
  - Basate sull'endpoint
    - HIDS / Host-based IDS
    - EDR / Endpoint Detection and Response

- NIDS usa i dati intercettati sui canali di comunicazione
  - È un sistema dedicato sul perimetro o con sonde, per il traffico di tutti i sistemi.
- HIDS è un processo userland sul sistema da proteggere
  - vede il traffico diretto a un singolo sistema
  - monitora il filesystem, i processi, le attività utente
  - esamina in tempo reale i log file
  - verifica periodicamente contenuti e metadati dei file
- EDR può essere definito come un HIDS fortemente integrato col sistema operativo

NIDS

Vantaggi

- Visibilità di tutto il traffico, entrante e uscente
- Richiede un solo punto di installazione
  - vero solo se rete semplice (dispositivi mobili che lasciano la rete??)
  - con più sonde: possibilità di ragionare su flussi
- Un malfunzionamento non incide sugli endpoint

Svantaggi

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

- < 20-30 Mbps: raspberry PI
- < 200-300 Mbps: pc desktop di un paio d'anni
- < 1 Gbps: server almeno 8 core / 16 thread
- > 10 Gbps: 40+ core e probabilmente serve accelerazione 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 i sistemi già esistenti
  - Non molto impegnativo computazionalmente (distribuito)

Svantaggi

- Punti ciechi
  - Se un evento/pacchetti non lascia tracce sul filesystem è invisibile
  - Non valuta il traffico uscente (egress) – 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 ad 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 run time (system call, fork)
- capacità anti-tampering
- maggiori possibilità di risposta (isolamento di comunicazioni e di processi)

Svantaggi

- richiede interfacciamento stretto con OS (non così ovvio)
  - hooking
  - minifilters
- difficile trovare soluzioni open e non molto costose

Host IDS – integrity check

La rilevazione di intrusioni sull'host è tipicamente svolta per mezzo di un ***integrity checker***.
Principio:

- Si memorizza in un database lo stato del filesystem quando è certamente "pulito"
- Si confronta periodicamente il filesystem col database

Tra i più diffusi:

- Tripwire (commerciale)
- AIDE (fork FOSS di Tripwire)
- AFICK

Integrity checking – homemade

I tool crittografici di base permettono di costruire a mano elenchi con hash dei contenuti dei file essenziali.
In alcuni casi già la distribuzione del sistema aiuta.

AIDE

AIDE è un controllore di integrità configurabile.
Il file `/etc/aide.conf` definisce i tipi di controlli da applicare a file e directory.
Processo:

- Un database di riferimento deve essere costruito su di un sistema *pulito*
- Una scansione periodica confronta i file di sistema con il database in base alla configurazione e riporta le differenze rilevanti

AIDE – qualche esempio

La seguente riga di selezione esaminerà tutto nella directory `/etc`, esaminando in particolare il numero di collegamenti, l'utente che possiede un dato file, il gruppo che possiede un dato file e la dimensione del file:

```
/etc n+u+g+s
```

Gli oggetti possono essere ignorati o saltati utilizzando un punto esclamativo (!), come nell'esempio seguente, che fa sì che AIDE ignori tutto in /var/log: `!/var/log/.*`

I pattern sono sottostringhe ancorate alla radice: attenzione alle esclusioni apparentemente specifiche! `!/var/log/maillog` (quello che si voleva dire, forse era: `!/var/log/maillog$`)

Caratteristiche:

- Compilato, molto veloce
- Integrabile con permessi estesi (acl, selinux)

AIDE – quick start

Configurare:

- le regole di controllo
- il nome del database di riferimento (usato per i controlli)
  - `database=file:/usr/local/aide/aide.db`
- il nome del nuovo database prodotto ad ogni aggiornamento
  - `database_out=file:/usr/local/aide/aide.db.new`
- Inizializzare il database
  - `aide --init`
- Rinominarlo per usarlo come riferimento per i controlli futuri:
  - `mv /usr/local/aide/aide.db.new /usr/local/aide/aide.db`
  - Lanciare un integrity check:
    - `aide -check`

AIDE – uso appropriato

A seconda del caso, AIDE può essere:

- eseguito come strumento forense, solo se si sospetta / si verifica un'irruzione
- 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 / 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, se non se ne dispone, da un supporto di ripristino
  - fermo macchina!

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

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 processi utente
- da processi 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

Soluzioni comuni

- Tipicamente producono file di testo
- Nessuna garanzia di uniformità di formato a parte la marcatura temporale
- In prospettiva integrato in systemd:
  - Journal
  - Attivo dal boot, non dipende dall'avvio di altri servizi
  - Formato binario, visualizzabile con **journalctl**

In-kernel auditing

Linux (≥4.18) supporta il tracciamento di ogni evento legato alle system call. 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)
- per cercare specifici eventi (ausearch)
- per rilanciare le notifiche di eventi ad altre applicazioni invece di scriverle nell'audit log (auditspd)
- per tracciare un processo, analogamente a strace (autrace)

Esempi:

- `auditctl -w /etc/passwd -p rwxa -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
  - 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

- Aggregazione dei dati
  - raccoglie log / eventi da più fonti
  - normalizza e consolida i dati
  - sistema di interrogazione centralizzato
- Correlazione
  - collega eventi con attributi comuni in pacchetti significativi
  - genera automaticamente avvisi in base a condizioni specifiche
- Monitoraggio
  - grafici per visualizzare lo stato corrente
  - grafici relativi alla conformità
- Ritenzione
  - conservazione a lungo termine
  - 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.

OSSEC

Due modalità di funzionamento

- locale, client-server

Modalità client-server

- i client ricevono la configurazione da un server
- i client inviano i log al server su canale cifrato

OSSEC config / decoders e rules

- Parsing dei log file configurabile per mezzo di decoders
  - monitoraggio di file multipli
  - regole di parsing ed estrazione scritte in XML
  - forniscono i campi utili per l'attivazione delle rules
- Analisi dei dati per mezzo di *rules*
  - scritte in XML
  - componibili in gerarchia
  - ruleset pre-configurati per i servizi più diffusi

OSSEC config / alerts

Ci sono un sacco di azioni predefinite per verificare la sicurezza del proprio sistema.
Permette inoltre la creazione di alert personalizzati, che scattano in funzione delle rules e possono eseguire:

- logging dell'evento
- invio di e-mail, sms, …
- esecuzione di uno script – possibilità di esecuzione su host multipli

Network IDS

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

- esaminare tutto il traffico senza rallentarlo
- generando pochissimi falsi allarmi
- senza lasciar sfuggire attacchi reali

Due approcci:

- Signature based: rileva flussi con caratteristiche notoriamente malevole

- Anomaly based: rileva flussi che si discostano dalla “normalità”.

Tra i più diffusi – Snort – **Suricata** – Zeek (ex Bro).

Suricata

- Configurazione:
  - Implementa un linguaggio per la rilevazione di signature
    - Compatibile con SNORT
  - Può essere configurato per rilevare anomalie
  - Può essere esteso con LUA per processing oltre la capacità del linguaggio a regole

- Caratteristiche di funzionamento particolari:
  - Riconosce automaticamente il tipo di traffico e adatta il dettaglio dei log
    - es. salva i certificati X.509 usati nelle connessioni TLS
    - salva l’header a livello applicazione per i protocolli più comuni
      - Deep Packet Inspection

- Output facilmente integrabile con molti strumenti di analisi e visualizzazione
- Molte fonti gratuite di signature.

Suricata può agire da IPS (Intrusion Prevention System). Se interposto tra due reti, può non inoltrare il traffico malevolo.

Lezioni del 11/05/2022 – Laboratorio di HIDS

bho
bho

Lezioni del 13/05/2022 – Host And Cloud security

Messa in sicurezza fisica

Un server è prima di tutto un sistema di calcolo, collocato in un ambiente e connesso a una varietà di dispositivi.

- Normalmente si concentrano le difese sul fronte degli attacchi via rete, a componenti software come applicazioni e sistema operativo
- Le corrispondenti contromisure possono facilmente essere scavalcate da un attaccante con accesso fisico al sistema!
- Le minacce principali sono:
  - Furto dello storage o dell'intero calcolatore
  - Connessione di sistemi di raccolta dati alle interfacce
  - Avvio del sistema con un sistema operativo arbitrario
- La gravità di queste minacce dipende fortemante dallo specifico ambiente
- Molti di questi problemi sono cambiati nello scenario sempre più comune di virtualizzazione sul Cloud, ma altri concettualmente simili sono apparsi, e la logica delle stesse contromisure si può adattare

Se la collocazione è fuori dalla possibilità di controllo diretto, si può considerare di:

- Scegliere un case che possa essere chiuso e fissato al rack – Installare dispositivi di rilevazione delle intrusioni
- Adottare misure di protezione dei dati che rendano inutile il furto
  - L'accesso ai dati va però abilitato manualmente
- Disabilitare le periferiche non utilizzate
  - Salvo poi averne bisogno per esigenze nuove

Attacchi fisici alle risorse logiche

- Per andare a regime il sistema attraversa un processo di boot, che può essere diviso in queste fasi:
  1. BIOS – Individua i dispositivi di possibile caricamento del boot loader e l'ordine per esaminarli
     - Molti BIOS prevedono la possibilità di proteggere con password l'avvio o la modifica della configurazione
  2. Boot Loader – Sceglie il sistema operativo e gli passa eventuali parametri
     - Gestione della "maintenance mode", stesso tipo di protezione con password come descritto per BIOS
  3. Sistema operativo
     - carica i device driver (da non sottostimare) e avvia il processo init
  4. *init* – gestisce i runlevel o i target per coordinare l'inizializzazione del sistema, cioè avviare i servizi nell'ordine corretto

- Ognuna di queste fasi potrebbe essere dirottata da un attaccante con accesso fisico, per far caricare software malevolo

Boot

- Password pro e contro:
  - Se serve una password per l'avvio del sistema, il funzionamento non supervisionato può essere problematico: ad esempio può non ripartire per ore dopo una semplice mancanza di alimentazione
    - In sistemi con gravi esigenze di sicurezza, non sarà comunque l'unica password da fornire, quindi meglio proteggere tutti gli strati
  - Almeno la protezione contro i cambi di configurazione è sempre consigliabile
- MAI affidarsi a un unico strato di protezione
  - Le password del BIOS hanno meccanismi semplici di reset
  - Possono essere indovinate…

Bootloader – configurazione (runtime)

Alcuni esempi di bootloader sono:

- LILO, the Linux Loader – Usato fin dagli albori di Linux
- GRUB, the Grand Unified Bootloader
  - GRUB è più potente e flessibile di LILO, è dotato di una shell che permette di eseguire vari comandi per modificare al volo la procedura di avvio: naturalmente questo permette molti abusi
- Entrambi permettono di passare parametri al kernel, i più importanti ai fini della sicurezza sono
  - single
  - Init=…
- Alcune distribuzioni hanno default rischiosi, ad esempio se si riesce a innescare un maintenance mode aprono una shell di root senza chiedere password

Sicurezza del processo di boot

Problema: come assicurarsi che ogni componente software eseguito da un computer sia autentico, integro e benevolo?

- Anti-malware verificano le applicazioni
- Chi verifica gli anti-malware?? Il S.O. (idealmente rendendo AM inutile)
- Chi verifica il S.O.? Il boot loader potrebbe
- Chi verifica il boot loader? Il BIOS potrebbe, specialmente se assistito da HW speciale, che non possa essere modificato dal S.O., e quindi sia immune da infezioni → **hardware root of (a chain of) trust**

Measured / Trusted / Secure Boot

- **Measured Boot** si riferisce a un processo generale, che tipicamente usa un TPM come hardware root of trust
  - **TPM** = Trusted Platform Module: chip con funzionalità crittografiche
  - Measured Boot non definisce come prevenire un avvio malevolo

- **Trusted Boot** è un processo che usa gli strumenti del Measured Boot e riesce a bloccare il boot non appena individua un componente non fidato.

- **Secure Boot** è il nome specifico dato all'implementazione di trusted boot basata su UEFI
  - UEFI = Unified Extensible Firmware Interface
  - Implementazione Software + chiavi in firmware
  - Serve un BIOS standard per la fase di POST – Può avvalersi del TPM per velocizzare e

migliorare i controlli di integrità

Measured boot

- Basato sul TPM
  - Core Root of Trust for Measurement (**CRTM**)
  - Registers (**PCR**)
- Raccoglie hash di ogni componente caricato, e li inserisce nei PCR che sono fisicamente non modificabili una volta scritti.
- Pospone i controlli fintanto che non dispone
  - Delle crypto keys
  - Di abbastanza memoria per fare i calcoli necessari
- Si può decidere chi fa i controlli e quando – per esempio dall'esterno (sistema fidato) per abilitare funzioni critiche – remote attestation!

UEFI e secure boot

EFI (Intel) nasce come interfaccia più flessibile del BIOS tra S.O. e firmware. UEFI forum standardizza e aggiorna la specifica. UEFI è un "mini OS".

- UEFI verifica ogni componente software prima di passare il controllo a BootLoader/SistemaOperativo
- Richiede la disponibilità di un database di chiavi – Blocca il boot appena rileva una difformità

Le chiavi di UEFI Secure Boot

- UEFI Secure Boot definisce due processi di sicurezza:
  - verifica dell'immagine di boot
  - verifica degli aggiornamenti al database della sicurezza delle immagini
- Per fare questo si avvale di differenti database e set di chiavi

Ci sono molte entità coinvolte nel processo di verifica delle immagini al boot:

- **TP** = trusted platform, procedura di verifica
- **CDI** = UEFI Secure Boot Image Security Database
- **UDI** = qualsiasi firmware di terze parti, inclusi boot loader, PCI option ROMs, o UEFI shell tool.

Al boot, TP verifica l'integrità di UDI utilizzando le policy CDI

- se ok, UDI entra a far parte di CDI e il firmware di terze parti viene eseguito

Il CDI, cioè il database delle politiche di sicurezza da applicare alle immagini software da caricare, è quindi aggiornabile.

- Il fornitore del componente deve firmarlo con la propria chiave privata e rendere disponibile la chiave pubblica.
- la chiave pubblica deve essere iscritta (enrolled) nel firmware del sistema
- normalmente questo **passaggio richiede un reboot in una modalità speciale e l'intervento sulla console**, bloccando quindi l'azione di utenti malevoli ma senza accesso fisico al sistema

Cloud Security - cloud vs. on-premise

- I problemi di sicurezza del cloud riguardano raramente la sicurezza dell'host e della rete
- Impatto "emotivo" della distanza: perdita di controllo
- Osservazione razionale
  - il più delle volte, i fornitori di cloud hanno team di sicurezza di livello mondiale che quasi nessuna azienda può permettersi di assumere per i propri data center
- la due diligence suggerisce di verificare questa affermazione quando si seleziona un fornitore di servizi cloud: Certificazione ISO27001 + ambito di applicazione, risultati di un audit SAS70 di tipo II
- Paradossalmente, alcune minacce reali riguardano la disponibilità
  - Vendor lock-in
  - dipendenza dalla rete
  - cessazione dell'attività

I benefici potenziali del cloud per la sicurezza

Le misure di sicurezza sono più economiche se implementate su scala più ampia:

- Collocazioni multiple = ridondanza di istanze e dati = indipendenza dai guasti dovuti a comportamenti dannosi, opportunità di ripristino
- **Tempestività migliorata**: è possibile utilizzare sistemi su scala più ampia per sviluppare capacità di risposta agli incidenti più efficaci
- Gestione delle minacce: i fornitori di servizi cloud possono coinvolgere specialisti nella gestione di minacce alla sicurezza specifiche

La sicurezza come elemento di differenziazione sul mercato

- Il cliente del servizio cloud (CSC) effettuerà le scelte di acquisto di servizi e risorse sulla base della reputazione di riservatezza, integrità e resilienza del fornitore di servizi cloud (**CSP**), nonché dei servizi di sicurezza offerti da CSP
- Questo è un forte motore per i CSP per migliorare le pratiche di sicurezza

Aggiornamenti più tempestivi ed efficaci:

- Le immagini delle macchina virtuale possono essere pre-rafforzate e aggiornate con le ultime patch e impostazioni di sicurezza in modo centralizzato, riducendo al minimo le vulnerabilità
- **IaaS (Infrastructure as a Service)** può fornire un servizio per consentire l'acquisizione regolare di istantanee dell'infrastruttura virtuale, nonché la rapida implementazione di numerosi aggiornamenti su piattaforme omogenee

Concentrazione delle risorse e rapido ridimensionamento:

- La concentrazione delle risorse consente l'applicazione più semplice ed economica di controlli di sicurezza completi e policy sui processi di manutenzione, gestione degli incidenti e gestione delle patch
- CSP può scalare dinamicamente i meccanismi difensivi su richiesta per il **traffic shaping**, il filtraggio, la crittografia, ecc., al fine di aumentare il supporto per le misure di riparazione durante un attacco
- Quando queste capacità sono combinate con un appropriato schema di ottimizzazione delle risorse, il CSP può limitare l'effetto che alcuni attacchi potrebbero avere sulla disponibilità delle risorse che ospitano i servizi cloud, nonché limitare l'uso delle risorse necessarie per affrontare tali attacchi

Sicurezza come servizio e raccolta di prove

- I CSP di grandi dimensioni possono offrire come servizio interfacce standardizzate per la sicurezza gestita; questo potenzialmente crea un mercato per i servizi di sicurezza, dove i

CSC possono facilmente acquisire controlli di sicurezza preconfigurati con costi di set-up inferiori
- IaaS può offrire supporto per la sicurezza su richiesta, clonare macchine virtuali (VM) e fornire uno storage conveniente per i log, che può essere utilizzato per analisi forensi offline, senza compromettere le prestazioni

Gli **SLA (Service Level Agreement)** impongono una migliore gestione del rischio
- I CSP implementano procedure di audit interno e valutazione del rischio più rigorose per quantificare le sanzioni per gli scenari di rischio negli SLA e definire azioni correttive per ridurre il possibile impatto delle violazioni della sicurezza sulla propria reputazione

I rischi specifici del cloud

Tre categorie principali:
- Rischi Organizzativi
  - Perdita di controllo
  - Lock-in
  - Supply Chain Failure
- Rischi Tecnici
  - Economic denial of service (EDoS)
  - Cedimento dell'isolamento
- Rischi legali
  - Rischi riguardanti la protezione dei dati
  - Rischi riguardanti la giurisdizione

Rischi Organizzativi

Perdita di controllo – forse il rischio considerato più grave:
- CSC cede il controllo a CSP su molti aspetti critici per la sicurezza
- se la SLA di CSP lascia un gap rispetto alle necessità, non c'è modo di chiuderlo
- outsourcing e subcontracting potrebbero mettere in gioco attori non fidati
- CSC non può verificare autonomamente la compliance di CSP

Lock-in:
- molti aspetti proprietari rendono difficile la migrazione
- anche on premise, ma almeno i dati restano in mano al CSC…

Supply Chain Failure
- outsourcing crea catene di fornitura dei servizi: forti quanto l'anello più debole

Interferenze tra politiche di sicurezza CSC-CSP
- CSC potrebbe desiderare controlli in conflitto con l'ambiente CSP, quindi non implementabili
- insicurezze di un CSC potrebbero diventare vulnerabilità di tutta la piattaforma che lo ospita

Rischi Tecnici

Economic denial of service (EdoS)
- Servizi pay-per-use che scalano automaticamente → un attacco di sovraccarico, invece di degradare le prestazioni, aumenta i costi

Vulnerabilità della piattaforma
- Un attacco all'infrastruttura potrebbe consentire l'accesso a tutte le VM
- Molto improbabile, ma devastante

- Più plausibile: compromissione dell'interfaccia di gestione

Cedimento dell'isolamento

- Multi-tenancy teoricamente permette di installare una VM malevola accanto a quelle della vittima
  - Possibili attacchi cross-VM via side channel
  - Possibile forzatura di migrazioni saturando l'host

- Intercettazione dei dati in transito
  - non tanto quelli dell'utente (che deve cautelarsi cifrando end-to-end)
  - operazioni trasparenti del CSP: sincronizzazioni, migrazioni, ecc.
  - raramente CSP fornisce garanzie di sicurezza su questi aspetti

Rischi legali

- Rischi riguardanti la protezione dei dati
  - legislazioni differenti
  - potrebbe essere illegale trasferire alcuni tipi di dati in alcuni paesi
  - CSP potrebbe spostare i dati tra i propri datacenter senza dirlo a CSC
- Rischi riguardanti la giurisdizione
  - CSP potrebbe essere costretto ad azioni dagli organi giudiziari sulla base di leggi del paese della sede legale, del paese dove sono collocati i datacenter, ecc…
  - CSC potrebbe subire interruzioni di servizio o sequestro di dati sulla base di motivi che non sussistono nel proprio paese

Lezioni del 18/05/2022 – Laboratorio su NIDS (Suricata)

Anomaly Based vs. Signature Based IDS

Come già sappiamo:

- Anomaly based
  - Monitora il traffico di rete
  - Tiene traccia dei modelli di traffico e delle informazioni per ottenere dati di riferimento
  - Se viene rilevata una deviazione nel comportamento della rete, l'IDS rileverà un attacco.
  - Elevato rischio di falsi positivi
- Signature based
  - Il database delle firme degli attacchi viene mantenuto localment
  - Confronta il traffico con il database
  - Richiede degli aggiornamenti costanti del db

Suricata

Suricata è un motore di rilevamento delle minacce di rete gratuito e open source, veloce e robusto.
Il motore Suricata è in grado di eseguire rilevamento delle intrusioni in tempo reale (IDS), ***prevenzione*** delle intrusioni (IPS), monitoraggio della sicurezza di rete (NSM) e l'elaborazione dei pcap offline.
Suricata ispeziona il traffico di rete utilizzando regole potenti ed estese e dispone di un potente supporto di scripting Lua per il rilevamento di minacce complesse.

Il file di configurazione è `/etc/suricata/suricata.yaml.`
La configurazione di default è sufficente per iniziare a testarlo e usarlo.
Potete avere più configurazioni in cascata.
Ogni sezione del file di configurazione principale è commentata efficacemente.

Lanciamo un shell di root, e andiamo a modificare il file di configurazione.
La prima cosa che ci interessa è la HOME_NET, che indica essenzialmente la rete che stiamo analizzando attualmente. EXTERNAL_NET invece specifica esattamente il traffico esterno che vogliamo analizzare.

Abbiamo poi varie opzioni di loggin:

- *fast* indica un semplice log unixlike
- *eve-log* invece è più dettaglio e più espressivo

Quello che a noi ci interessa veramente però è il file di ***rules***, che si trova sotto `rules-file`.
Una regola suricata ha una composizione del genere:

```
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;)
```

"Avvisami quando un pacchetto http arriva dalla HOME_NET alla EXTERNAL_NET a prescindere dalla porta (any) e logga con messaggio DetoxCrytpto Ransomware etc…

E il pacchetto contiene una POST, fa richiesta a /generate.php, ha contenuto "publickey=" etc….

Possiamo trovare le regole al path `/etc/suricata/rules/` .

Formato delle regole

Possiamo svolgere varie azioni: drop, alert, pass, reject, log

- Intestazione: protocol address port direction address port
  - Protocol : ip(all/any), tcp, udp, icmp, dns
  - 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

Nella nostra macchina del lab non possiamo usare drop, si può impostare com IPS, ma è complesso.

Comandi utili

Per visualizzare i vari parametri, come sempre, possiamo usare suricata -h, che ci lista i paranìmetri del comando.

In generale, attraverso il comando

```
suricata -c suricata.yaml -s some_rule.rules -i eth0 -vvv
```

I flag indicano:

- -c <yaml configuration file location>
- -i <interface to sniff>
- -s <signatures file> (runs in addition to -c)
- -r <pcap recording file location> (è una configurazione che useremo)
- -l <default log directory location>
- -D }:-)

Possiamo usare anche suricata in modalità "offline", dandogli in pasto un file in formato pcap. In poche parole, invece che lanciare suricata in listening su un'interfaccia di rete, viene lanciato dandogli in input il tracciato di un traffico.

Se usiamo però l'analisi statica, il file di log ci compare totalmente da un'altra parte (nella cartella delle regole in particolare).