

# Fondamenti dell'Architettura Internet e Vulnerabilità Intrinseche dei Protocolli IP/TCP

(Appunti elaborati a partire dalle slide del corso  
Network and System Defense, A.A. 2025/2026)

Leonardo Polidori, Edoardo Marchionni, Chat GPT

17 novembre 2025

## Indice

<b>1</b>	<b>NET_01</b>	<b>11</b>
1.1	Architettura di base e principi di rete . . . . .	11
1.1.1	Definizione e interconnessione . . . . .	11
1.1.2	Indirizzamento e instradamento . . . . .	11
1.1.3	Anatomia dell'indirizzo IP e subnetting . . . . .	11
1.1.3.1	1) Cosa significa “/27” . . . . .	12
1.1.3.2	2) Perché basta guardare l'ultimo otetto . . . . .	12
1.1.3.3	3) Trova il blocco in cui cade 99 . . . . .	12
1.1.3.4	4) Verifica con l'AND (ultimo otetto) . . . . .	13
1.1.3.5	5) Broadcast: tutti i bit host a 1 . . . . .	13
1.1.3.6	6) Intervallo host e conteggio . . . . .	13
1.1.4	Sistemi autonomi (AS) e routing globale . . . . .	13
1.1.5	La routing table e l'algoritmo di lookup . . . . .	13
1.2	II. Il viaggio del pacchetto: esempio di richiesta DNS . . . . .	13
1.2.1	Topologia e hop . . . . .	13
1.2.2	Stack protocollare e incapsulamento . . . . .	14
1.3	III. Vulnerabilità intrinseche di IP e TCP . . . . .	14
1.3.1	Identificazione, spoofing e non-ripudio . . . . .	15
1.3.2	Confidenzialità . . . . .	15
1.3.3	Integrità dei dati . . . . .	15
1.3.4	Packet replication e anti-replay . . . . .	15
1.3.5	Insicurezza delle mappature dinamiche . . . . .	15
1.4	Laboratorio 1: MiTM e DNS spoofing . . . . .	16
1.4.1	Obiettivo e scenario . . . . .	16
1.4.2	Fasi dell'attacco . . . . .	16
1.4.3	STEP 1: ARP spoofing (MiTM) . . . . .	16
1.4.4	STEP 2 & 3: intercettazione e DNS spoofing . . . . .	17
1.4.5	STEP 4: impersonificazione del sito web . . . . .	17
<b>2</b>	<b>NET_02</b>	<b>18</b>
2.1	Sicurezza delle Reti Ethernet LAN (Livello 2) . . . . .	18
2.2	Perché la LAN Ethernet è fragile per natura . . . . .	18
2.2.1	Frame, indirizzamento e forwarding . . . . .	18
2.2.1.1	Multiport Repeaters (Hub) . . . . .	18
2.2.1.2	Bridge/Switches . . . . .	18

2.2.1.3	Address Learning . . . . .	18
2.2.2	Topologie e controllo dei loop: STP . . . . .	19
2.2.3	Adattamento del Livello 3 su Livello 2: DHCP, ARP e NDP . . . . .	20
2.2.3.1	DHCP — Dynamic Host Configuration Protocol (IPv4). . . . .	20
2.2.3.2	ARP — Address Resolution Protocol (IPv4). . . . .	21
2.2.3.3	NDP — Neighbor Discovery Protocol (IPv6). . . . .	21
2.2.4	Vulnerabilità e minacce principali . . . . .	21
2.2.4.1	Accesso alla rete . . . . .	21
2.2.4.2	Compromissione e controllo dello switch. . . . .	22
2.2.4.3	Riservatezza e intercettazione. . . . .	22
2.2.4.4	Integrità del traffico e attacchi Man-in-the-Middle (MITM) . . . . .	23
2.2.4.5	Disponibilità e attacchi di Denial of Service (DoS) . . . . .	23
2.2.5	Contromisure: dal minimo sindacale al robusto . . . . .	24
2.2.5.1	Router-based security (segmentazione L3). . . . .	24
2.2.6	Controllo d'accesso (Access Control) . . . . .	25
2.2.6.1	Obiettivo. . . . .	25
2.2.6.2	802.1X (port-based NAC). . . . .	25
2.2.6.3	Autorizzazione dinamica con 802.1X. . . . .	25
2.2.6.4	Port Security & ACL sugli switch. . . . .	25
2.2.6.5	Segmentazione con VLAN (complemento). . . . .	25
2.3	Laboratorio 2A: ACL L2 con Linux Bridge ed <b>ebtables</b> (binding MAC→porta) .	26
2.3.0.1	1) Preparazione host (MAC/IP dei client). . . . .	26
2.3.0.2	2) Creazione del bridge L2 (lato “switch” Linux). . . . .	26
2.3.0.3	3) ACL L2 con <b>ebtables</b> (NETFILTER). . . . .	27
2.4	Laboratorio 2B: MACsec su Linux (confidenzialità, integrità e anti-replay a L2) .	27
2.4.0.1	Host A ( <b>client1</b> ). . . . .	27
2.4.0.2	Host B ( <b>client2</b> ). . . . .	28
2.4.0.3	Opzioni di sicurezza e test. . . . .	28
2.4.1	Secure Address Resolution (IPv4/IPv6) . . . . .	29
2.4.1.1	IPv4: dal piano di switching ai vincoli per-presa. . . . .	29
2.4.1.2	IPv4: proposte crittografiche. . . . .	29
2.4.1.3	IPv6: NDP sicuro. . . . .	29
2.4.2	Security Monitoring . . . . .	29
2.4.2.1	Firewall & DPI. . . . .	29
2.4.2.2	IDS/IPS e accesso al traffico. . . . .	29
2.4.2.3	Dove il monitoring aiuta davvero. . . . .	29
2.4.2.4	Messaggio chiave (dalle slide). . . . .	29
<b>3</b>	<b>NET_03 — Virtual LANs (VLAN)</b>	<b>30</b>
3.0.1	Definizione e motivazione . . . . .	30
3.0.2	Limiti delle reti fisicamente separate . . . . .	30
3.0.2.1	Benefici principali delle VLAN . . . . .	30
3.1	Assegnazione e membership delle VLAN . . . . .	31
3.1.1	Criteri di assegnazione . . . . .	31
3.1.2	Vista logica . . . . .	31
3.1.2.1	Router “one-armed” . . . . .	32
3.2	Trasporto dei frame: Tagging IEEE 802.1Q . . . . .	33
3.2.1	Porte di tipo Access, Trunk e Hybrid . . . . .	33
3.2.2	Access links . . . . .	33
3.2.3	Access links nelle regioni legacy . . . . .	34
3.2.4	Trunk links . . . . .	34
3.2.5	Hybrid links . . . . .	35

3.2.6	Una stazione può appartenere a più VLAN? . . . . .	36
3.3	Laboratorio 3: Configurazione VLAN e router one-armed . . . . .	38
3.3.0.1	Isolamento del traffico . . . . .	39
3.3.0.2	Routing inter-VLAN . . . . .	39
3.3.0.3	Verifica del comportamento . . . . .	40
3.3.0.4	Osservazione pratica . . . . .	40
3.4	Sicurezza delle VLAN e vulnerabilità di livello 2 . . . . .	41
3.4.1	Minacce principali . . . . .	41
3.4.1.1	1) MAC Flooding (CAM Overflow) . . . . .	41
3.4.1.2	2) ARP Spoofing / Poisoning . . . . .	41
3.4.1.3	3) VLAN Hopping . . . . .	42
3.4.1.4	4) Attacchi ai protocolli di controllo . . . . .	42
3.5	Laboratorio 4: VLAN Hopping e Double Tagging . . . . .	44
3.5.1	Scenario . . . . .	44
3.5.2	Topologia del laboratorio . . . . .	44
3.5.3	Prerequisiti perché l'attacco riesca . . . . .	45
3.5.4	Funzionamento dell'attacco (passo–passo) . . . . .	45
3.5.5	Esecuzione pratica (Linux) . . . . .	45
3.5.5.1	Cosa osservare con lo sniffer . . . . .	46
3.5.5.2	Limitazioni pratiche . . . . .	46
3.5.5.3	Mitigazioni . . . . .	46
<b>4</b>	<b>NET_04 — 802.1x</b>	<b>47</b>
4.1	Quadro generale e obiettivo . . . . .	47
4.2	Attori dell'architettura (vista di alto livello) . . . . .	47
4.3	EAP ed EAPoL: fondamenti . . . . .	47
4.3.0.1	EAP (Extensible Authentication Protocol). . . . .	47
4.4	EAP termination vs EAP relay (EAPoR) . . . . .	48
4.4.0.1	EAP termination mode (terminazione locale). . . . .	48
4.4.0.2	EAP relay mode (EAP over RADIUS, EAPoR). . . . .	48
4.5	Processi di autenticazione: dai diagrammi alle implicazioni operative . . . . .	49
4.5.0.1	Message exchange (visione di alto livello). . . . .	49
4.5.0.2	Relay Mode (EAP-MD5) — sequenza dettagliata. . . . .	50
4.5.0.3	EAP-TLS: esempio più complesso. . . . .	51
4.6	Operazioni aggiuntive: ri-autenticazione, logout, timer . . . . .	51
4.7	Autorizzazione post-autenticazione: VLAN, ACL, UCL . . . . .	51
4.7.0.1	VLAN dinamiche. . . . .	51
4.7.0.2	ACL per-utente. . . . .	51
4.7.0.3	UCL (User Control List). . . . .	52
4.8	Vulnerabilità storiche e motivazione per MACsec . . . . .	52
4.9	Lab 5 — 802.1X Port-Based Authentication & VLAN assignment . . . . .	53
4.9.0.1	Obiettivo. . . . .	53
4.9.0.2	Topologia di rete. . . . .	53
4.9.0.3	Configurazione dello switch (Authenticator). . . . .	53
4.9.0.4	Server di autenticazione (FreeRADIUS). . . . .	54
4.9.0.5	Configurazione dei client (Supplicant). . . . .	54
4.9.0.6	Controllo del traffico L2. . . . .	54
4.9.0.7	Sequenza operativa. . . . .	54
4.9.0.8	Osservazioni e risultati. . . . .	55
4.9.0.9	Conclusioni. . . . .	55
4.10	MACsec e MKA (802.1AE + 802.1X-2010) . . . . .	56
4.10.1	Perché MKA: il tassello mancante dopo 802.1X . . . . .	56

4.10.2	Le chiavi in gioco e come si ottengono . . . . .	56
4.10.2.1	CAK e CKN . . . . .	56
4.10.2.2	ICK e KEK . . . . .	56
4.10.2.3	SAK . . . . .	57
4.10.3	Due modalità d'uso del CAK . . . . .	57
4.10.3.1	Static CAK . . . . .	57
4.10.3.2	Dynamic CAK . . . . .	57
4.10.4	Cosa si scambiano i peer: il ciclo MKA . . . . .	57
4.10.4.1	Annuncio, stato e priorità . . . . .	57
4.10.4.2	Elezione del Key Server . . . . .	57
4.10.4.3	Apertura dei Secure Channel . . . . .	57
4.10.4.4	Vita della sessione . . . . .	58
4.10.5	Collegamento con le policy 802.1X/NAC . . . . .	58
4.11	Simple MKA lab with Linux . . . . .	59
4.12	Laboratorio: MKA/MACsec su Linux (Static CAK) . . . . .	59
4.12.1	Scopo . . . . .	59
4.12.2	Topologia . . . . .	59
4.12.3	Procedura . . . . .	59
4.12.3.1	Generazione delle chiavi (uguali su <i>tutti</i> i peer) . . . . .	59
4.12.3.2	Configurazione MKA su ciascun host . . . . .	59
4.12.3.3	Avvio di MKA e creazione dell'interfaccia <code>macsec0</code> . . . . .	60
4.12.3.4	Indirizzamento IP sulla <code>macsec0</code> . . . . .	60
4.12.3.5	Test base di connettività . . . . .	60
4.12.4	Verifiche . . . . .	60
4.12.4.1	Log MKA . . . . .	60
4.12.4.2	Stato interfaccia . . . . .	60
4.12.5	Troubleshooting . . . . .	60
4.12.6	Pulizia . . . . .	60
4.12.7	Estensioni & domande d'esame . . . . .	60
4.12.7.1	Dynamic CAK (accenno) . . . . .	60
<b>5</b>	<b>NET_05 — Firewall e Algoritmi di Classificazione dei Pacchetti</b>	<b>61</b>
5.1	Panoramica sui Firewall . . . . .	61
5.1.1	Obiettivi di Progettazione dei Firewall . . . . .	61
5.1.2	Politica di Accesso del Firewall . . . . .	61
5.1.3	Caratteristiche dei Firewall . . . . .	62
5.2	Firewall di Filtraggio dei Pacchetti . . . . .	62
5.2.1	Vantaggi e Svantaggi del Filtraggio dei Pacchetti . . . . .	63
5.3	Firewall di Ispezione Stateful . . . . .	64
5.3.1	Tabella dello Stato delle Connessioni . . . . .	64
5.4	Gateway e Funzione del Default Gateway . . . . .	65
5.4.1	Funzioni di un Gateway . . . . .	65
5.5	Application-Level Gateway (ALG) . . . . .	65
5.5.1	Funzionamento dell'ALG . . . . .	65
5.6	Circuit-Level Gateway . . . . .	65
5.6.1	Funzione del Circuit-Level Gateway . . . . .	66
5.7	Host-Based Firewall e Personal Firewall . . . . .	66
5.8	NETFILTER . . . . .	66
5.8.1	Funzionamento di NETFILTER . . . . .	66
5.8.2	Le Tabelle di NETFILTER . . . . .	67
5.8.3	Iptables: Il Frontend di NETFILTER . . . . .	68
5.8.3.1	Comandi principali di iptables . . . . .	68

5.8.4	Network Address Translation (NAT) . . . . .	69
5.8.4.1	Tipi di NAT . . . . .	69
5.8.5	Connection Tracking . . . . .	69
5.8.5.1	Esempi di stato delle connessioni . . . . .	69
5.8.6	Vantaggi e Limitazioni di NETFILTER . . . . .	69
5.9	Lab 5 — Firewall con <code>iptables</code> e policy di sicurezza gateway/server . . . . .	71
5.9.0.1	Obiettivo e componenti. . . . .	71
5.9.0.2	Struttura del laboratorio. . . . .	71
5.9.0.3	Gateway (firewall principale). . . . .	71
5.9.0.4	Server DMZ. . . . .	72
5.9.0.5	Testing. . . . .	72
5.10	Classificazione dei Pacchetti . . . . .	73
5.10.1	Perché è Importante la Classificazione dei Pacchetti? . . . . .	73
5.10.1.1	Requisiti e metriche. . . . .	73
5.10.2	Algoritmi di Classificazione dei Pacchetti . . . . .	73
5.10.2.1	Schema divide-and-conquer. . . . .	74
5.11	Problema della Classificazione dei Pacchetti . . . . .	74
5.11.1	Struttura delle Regole di Classificazione . . . . .	74
5.11.2	Combinazione di Regole con Match Multipli . . . . .	74
5.11.2.1	Selezione efficiente della miglior regola. . . . .	75
5.11.3	Algoritmi di Matching . . . . .	75
5.11.3.1	Altre famiglie chiave. . . . .	75
5.12	Approccio Bit Vector Linear Search . . . . .	75
5.12.1	Funzionamento del Bit Vector Linear Search . . . . .	76
5.12.1.1	Nota tecnica (integrazione e correzione). . . . .	76
5.12.2	Vantaggi e Limiti del Bit Vector Linear Search . . . . .	76
5.12.2.1	Precisazione sulla memoria. . . . .	76
5.13	Contenuto e Prestazioni della Classificazione dei Pacchetti . . . . .	76
5.13.1	Caching 5-tuple: benefici e limiti . . . . .	77
5.13.2	TCAM/CAM: pro e contro . . . . .	77
5.14	eBPF . . . . .	77
5.14.1	Hook eBPF (punti di aggancio) . . . . .	77
5.14.2	Focus sui percorsi di rete . . . . .	77
5.14.2.1	XDP . . . . .	77
5.14.2.2	JIT . . . . .	78
5.14.2.3	TC . . . . .	78
5.14.2.4	NETFILTER e Socket hooks . . . . .	78
5.15	Scrivere e caricare programmi eBPF . . . . .	78
5.15.0.1	Pipeline operativa . . . . .	78
5.15.0.2	Componenti di supporto . . . . .	78
5.16	Laboratorio X: Warm-up con eBPF e XDP . . . . .	79
5.16.1	Scenario . . . . .	79
5.16.2	Topologia del laboratorio . . . . .	79
5.16.3	Prerequisiti . . . . .	79
5.16.4	Passi del laboratorio (step-by-step) . . . . .	79
5.16.4.1	Step 1 — Basic PASS con logging . . . . .	79
5.16.4.2	Step 2 — Switch a DROP e detach . . . . .	80
5.16.4.3	Step 3 — Packet Filtering (hardcoded) con boundary checks . . . . .	80
5.16.4.4	Step 4 — Map Counter (pkt/byte) . . . . .	80
5.16.4.5	Step 5 — ACL denylist con mappa hash . . . . .	80
5.16.5	Esecuzione pratica (comandi essenziali) . . . . .	80

5.16.6	Cosa osservare con lo sniffer / tracing . . . . .	81
5.16.6.1	Troubleshooting . . . . .	81
5.16.7	Limitazioni pratiche . . . . .	81
5.16.8	Mitigazioni / buone pratiche . . . . .	81
<b>6</b>	<b>NET_06 - Secure Protocols and Overlay VPNs</b>	<b>82</b>
6.1	Sicurezza a livello applicativo: Secure Shell (SSH) . . . . .	82
6.1.0.1	Architettura e RFC. . . . .	82
6.1.0.2	Obiettivi di sicurezza. . . . .	82
6.1.0.3	Cifratura e handshake. . . . .	82
6.1.0.4	Autenticazione del server. . . . .	83
6.1.0.5	Autenticazione del client. . . . .	84
6.1.0.6	Servizi aggiuntivi. . . . .	84
6.1.0.7	Conclusione. . . . .	85
6.2	Sicurezza a livello di trasporto: Transport Layer Security (TLS) . . . . .	85
6.2.0.1	Obiettivi di TLS. . . . .	85
6.2.0.2	Ruolo nello stack di rete. . . . .	86
6.2.0.3	Esempio pratico. . . . .	86
6.2.0.4	Storia e porte standard. . . . .	86
6.2.0.5	Funzionamento. . . . .	87
6.2.0.6	Evoluzione a TLS 1.3. . . . .	87
6.2.0.7	Attacchi noti. . . . .	88
6.3	Laboratorio: HTTPS con Apache2 . . . . .	89
6.3.0.1	1. Creazione della CA. . . . .	89
6.3.0.2	2. Configurazione di Apache2. . . . .	89
6.3.0.3	3. Accesso e verifica. . . . .	89
6.3.0.4	Redirect da HTTP a HTTPS. . . . .	89
6.4	Attacchi di downgrade e contromisure . . . . .	90
6.4.0.1	HSTS. . . . .	90
6.5	Overlay VPNs e OpenVPN . . . . .	90
6.5.0.1	Concetto di tunneling. . . . .	90
6.5.1	OpenVPN . . . . .	90
6.5.1.1	Architettura. . . . .	90
6.5.1.2	Configurazione di base. . . . .	91
6.5.1.3	Routing e NAT. . . . .	91
6.5.1.4	Autenticazione e cifratura. . . . .	91
6.5.1.5	Vantaggi. . . . .	91
6.6	Conclusione . . . . .	91
<b>7</b>	<b>SYS_01 — Introduzione ai Security Frameworks</b>	<b>92</b>
7.1	Quadro generale e obiettivo . . . . .	92
7.2	Principi fondamentali . . . . .	92
7.2.0.1	Confidenzialità . . . . .	92
7.2.0.2	Integrità . . . . .	92
7.2.0.3	Disponibilità . . . . .	92
7.2.0.4	Autenticità e Autorizzazione . . . . .	93
7.2.0.5	Accountability . . . . .	93
7.2.0.6	Dipendibilità . . . . .	93
7.3	Perché servono i Security Frameworks . . . . .	94
7.4	Tre famiglie a confronto . . . . .	94
7.4.1	FIPS 200 — Requisiti minimi per i sistemi federali . . . . .	94
7.4.2	CIS Critical Security Controls — L'igiene prioritaria . . . . .	94

7.4.3 ISO/IEC 27000 — ISMS e gestione del rischio . . . . .	95
<b>8 SYS_02 — Hardware Primer</b>	<b>96</b>
8.1 Moore's Law e l'evoluzione del calcolo . . . . .	96
8.2 Fine della crescita lineare e parallelismo . . . . .	96
8.3 Pipeline e architettura superscalare . . . . .	96
8.4 Branch Prediction: principi e strategie . . . . .	96
8.4.0.1 Perché serve una predizione . . . . .	97
8.4.0.2 Tipi di predizione . . . . .	97
8.4.0.3 Contatore a 2 bit saturante . . . . .	97
8.4.0.4 Predittori correlati e multilivello . . . . .	97
8.4.0.5 Ottimizzazioni hardware . . . . .	98
8.4.0.6 Importanza crescente della predizione . . . . .	98
8.5 Simultaneous Multithreading (SMT) . . . . .	99
8.6 Pipeline interna nei processori Intel Xeon . . . . .	99
8.7 Gerarchia di memoria . . . . .	100
8.7.0.1 Inclusività e mappatura . . . . .	100
8.7.0.2 Sostituzione ed aggiornamento . . . . .	102
8.8 Cache coherence nei multicore . . . . .	103
8.8.0.1 Protocolli di coerenza . . . . .	104
8.9 Protocolli MOESI e VI . . . . .	105
8.9.0.1 Virtual vs. Physical Cache Indexing . . . . .	106
8.10 Hardware Transactional Memory (HTM) . . . . .	107
8.10.0.1 Principio di funzionamento . . . . .	107
8.10.0.2 Casi di abort e codici di stato . . . . .	108
8.11 DRAM e Refresh . . . . .	108
8.11.0.1 Effetti del refresh . . . . .	108
<b>9 SYS_04 - Hardware attacks and countermeasures</b>	<b>110</b>
9.1 Introduzione agli Attacchi Hardware . . . . .	110
9.2 Timing e Sicurezza . . . . .	110
9.2.0.1 Uso della Funzione strcmp . . . . .	110
9.2.0.2 Timing attack in un Programma di Autenticazione . . . . .	110
9.2.1 Attacchi di Temporizzazione nel Mondo Reale: RSA . . . . .	110
9.2.1.1 Attacchi alla Decrittazione RSA . . . . .	111
9.2.1.2 Algoritmi a Tempo Costante . . . . .	111
9.3 Preambolo: Side-Channel Attacks e Considerazioni Tecniche . . . . .	111
9.3.0.1 Fasi operative . . . . .	111
9.3.0.2 Scoprire i percorsi di codice . . . . .	111
9.3.1 Technical Considerations . . . . .	112
9.3.1.1 Nota pratica (TSX e detection) . . . . .	112
9.4 Attacchi side channel . . . . .	112
9.4.1 Timing della Cache . . . . .	112
9.4.1.1 Attacchi Flush + Reload . . . . .	112
9.4.1.2 Evict + Time . . . . .	112
9.4.1.3 Attacchi Prime + Probe . . . . .	113
9.4.1.4 Prime + Abort . . . . .	113
9.4.1.5 Flush + Flush . . . . .	113
9.4.2 Meeting Out-of-Order Pipelines: perché conta per la sicurezza . . . . .	113
9.4.3 Meltdown: come funziona (Primer) . . . . .	114
9.4.3.1 Idea chiave . . . . .	114
9.4.3.2 Schema operativo Meltdown . . . . .	114

9.4.3.3	Perché è possibile . . . . .	114
9.4.4	Fooling the Branch Prediction Unit (BPU) . . . . .	114
9.4.4.1	Dove si colpisce e perché funziona . . . . .	115
9.4.5	Spectre Primer v1 (Bounds Check Bypass) . . . . .	115
9.4.6	Spectre Primer v2 (BTB Poisoning e Gadget) . . . . .	115
9.4.6.1	Esempio di gadget dalle slide (Windows 10) . . . . .	116
9.4.6.2	Requisiti e note . . . . .	116
9.4.7	L1TF (Foreshadow): principio e flusso . . . . .	116
9.4.7.1	Perché funziona . . . . .	116
9.4.7.2	Schema operativo (dalle slide) . . . . .	116
9.4.8	Mitigating Side-Channel Attacks . . . . .	117
9.4.8.1	Possible Mitigations and Detection . . . . .	117
9.4.8.2	Mitigations: the hard way . . . . .	117
9.4.8.3	Mitigations to L1 Attacks . . . . .	118
9.4.8.4	Mitigations: the detection way . . . . .	118
9.4.8.5	Meltdown Mitigation: Kernel Page Table Isolation . . . . .	118
9.4.9	cpu_entry_area: Per-CPU Isolation . . . . .	118
9.4.10	Double Page General Directory e Switch to CR3 . . . . .	119
9.4.11	Retpoline: Protezione contro gli Attacchi di Branch Target Injection e Retpoline Thunks . . . . .	119
9.4.12	Prevent Branch Poisoning: IBRS, STIBP, IBPB . . . . .	120
9.5	RowHammer: Attacco alla Memoria e le sue Mitigazioni . . . . .	120
9.6	Memory Performance Attacks: Attacchi alle Prestazioni della Memoria . . . . .	121
9.6.0.1	Denial of Service nei Sistemi Multi-Core . . . . .	121
9.6.0.2	Funzionamento degli Attacchi a Memoria Multi-Banca . . . . .	122
9.6.0.3	Scheduling della Memoria e Contesa tra Banchi . . . . .	122
9.6.0.4	Mitigazioni e Strategie di Ottimizzazione . . . . .	122
<b>10 SYS_05 - OS Security Principles</b>		<b>123</b>
10.1	Introduzione . . . . .	123
10.2	Identificazione e Autenticazione degli Utenti . . . . .	123
10.2.0.1	/etc/passwd . . . . .	123
10.2.0.2	/etc/shadow . . . . .	123
10.3	Gestione degli UID e dei Privilegi . . . . .	124
10.3.0.1	su e sudo . . . . .	124
10.4	Principio del Minimo Privilegio . . . . .	124
10.5	Controllo degli Accessi . . . . .	124
10.5.1	Politiche di Sicurezza: DAC e MAC . . . . .	125
10.5.2	POSIX ACL e Access Control Fine-Grained . . . . .	125
10.5.2.1	Esempio di ACL POSIX . . . . .	125
10.5.2.2	Meccanismo di valutazione delle ACL . . . . .	126
10.5.2.3	Ruolo della Mask . . . . .	126
10.6	Linux Capabilities . . . . .	126
10.6.0.1	Esempi di capability . . . . .	127
10.6.0.2	Implementazione nel kernel . . . . .	127
10.6.0.3	Insiemi di capability di un thread . . . . .	127
10.6.0.4	File capabilities . . . . .	128
10.6.0.5	Capability Bounding Set . . . . .	128
10.7	Linux Security Modules (LSM) . . . . .	128
10.7.0.1	Concetto di base . . . . .	128
10.7.0.2	Struttura generale . . . . .	128
10.7.0.3	Funzionamento del framework . . . . .	129

10.7.0.4	Hook di sicurezza . . . . .	129
10.7.0.5	Categorie di hook . . . . .	130
10.7.0.6	Esempio: hook su un file system . . . . .	130
10.7.0.7	Estensioni del kernel per LSM . . . . .	130
10.7.1	Esempi di Moduli MAC . . . . .	131
10.7.1.1	S.M.A.C.K. (Simplified Mandatory Access Control Kernel) . . . . .	131
10.7.1.2	Etichette di default . . . . .	131
10.7.1.3	SMACKFS . . . . .	132
10.7.1.4	Caratteristiche e limiti . . . . .	132
10.7.1.5	Tomoyo Linux . . . . .	132
10.7.1.6	AppArmor . . . . .	132
10.7.1.7	SELinux (Security-Enhanced Linux) . . . . .	133
10.7.1.8	Fase 1: dal kernel alla query di autorizzazione . . . . .	133
10.7.1.9	Fase 2: ricerca della policy SELinux . . . . .	134
10.7.1.10	Stato di etichettamento (Labeling State) . . . . .	134
10.7.1.11	Stato di transizione (Transition State) . . . . .	134
10.8	Boot Time Security . . . . .	135
10.8.0.1	Funzionamento del Ramdisk durante il Boot . . . . .	135
10.8.0.2	Horse Pills . . . . .	136
10.8.0.3	Cosa può fare un ramdisk infetto . . . . .	136
10.8.0.4	Mitigazioni e strategie di difesa . . . . .	137
10.9	Sicurezza di Rete a Livello OS . . . . .	138
10.9.0.1	inetd e xinetd . . . . .	138
10.9.0.2	TCP Wrappers e Reverse DNS . . . . .	138
10.9.0.3	File di configurazione . . . . .	138
10.9.0.4	Esempi pratici . . . . .	139
10.9.0.5	Ordine e logica di valutazione . . . . .	139
10.9.0.6	Reverse DNS Tampering: il problema . . . . .	139
10.9.0.7	Esempio di attacco semplificato . . . . .	139
10.9.0.8	Contromisure pratiche . . . . .	139
10.9.0.9	Conclusione . . . . .	140
<b>11</b>	<b>SYS_06 - Antivirus</b>	<b>141</b>
11.1	Problemi principali degli antivirus . . . . .	141
11.2	Evoluzione degli antivirus . . . . .	141
11.3	Componenti di un antivirus . . . . .	141
11.4	Esempi di packing . . . . .	142
11.4.0.1	1. Poche o nessuna importazione nella IAT (Import Address Table)	142
11.4.0.2	2. Nomi di sezioni non standard o incoerenti . . . . .	142
11.4.0.3	3. Sezioni piccole su disco ma grandi in memoria . . . . .	142
11.4.0.4	4. Poche stringhe leggibili . . . . .	142
11.4.0.5	5. Sezioni con permessi RWX . . . . .	143
11.4.0.6	6. Salti o chiamate a registri/indirizzi anomali . . . . .	143
11.5	Approccio operativo alla rilevazione . . . . .	143
11.6	Tecniche di evasione . . . . .	144
11.7	Analisi statica e comportamentale . . . . .	144
11.7.0.1	Analisi statica . . . . .	145
11.7.0.2	Analisi comportamentale . . . . .	145
11.8	Emulatori e Fingerprinting . . . . .	147
11.8.0.1	Funzionamento degli emulatori . . . . .	147
11.8.0.2	Limiti e tecniche di evasione . . . . .	147
11.8.0.3	Fingerprinting dell'ambiente . . . . .	147

11.8.0.4	Contromisure adottate dagli antivirus . . . . .	148
11.9	Analisi del traffico cifrato . . . . .	148
11.10	Fiducia e limiti del software antivirus . . . . .	148
<b>12</b>	<b>SYS_07 - eBPF</b>	<b>149</b>
12.1	Motivazioni: Perché osservare ciò che accade nel cluster . . . . .	149
12.2	Il ruolo del kernel e la necessità di strumenti avanzati . . . . .	149
12.3	eBPF: Hooks nel kernel e nello user space . . . . .	150
12.4	Sicurezza e osservabilità: un modello unificato . . . . .	150
<b>13</b>	<b>Tipologie di Hook eBPF</b>	<b>150</b>
13.0.0.1	Tracepoints Kernel (statici). . . . .	151
13.0.0.2	Kprobes (dinamici). . . . .	151
13.0.0.3	USDT (User Statically Defined Tracing). . . . .	151
13.0.0.4	Uprobes (dinamici). . . . .	151
13.1	Tracepoints nel kernel . . . . .	151
13.2	Sviluppo e caricamento di programmi eBPF con libbpf-bootstrap . . . . .	151
13.3	Laboratorio: Tracepoint <code>openat</code> . . . . .	152
13.4	User Statically Defined Tracing (USDT) . . . . .	153
13.5	Kprobes e analisi dinamica del kernel . . . . .	153
13.6	Uprobes: analisi dinamica di applicazioni user-space . . . . .	153
13.7	Progetti reali basati su eBPF . . . . .	153
13.8	Abuso dei BPF filter: il caso BPFDoor . . . . .	153
13.9	Conclusioni . . . . .	153

# 1 NET\_01

## 1.1 Architettura di base e principi di rete

### 1.1.1 Definizione e interconnessione

Internet è un'*inter-rete*: un insieme di numerose sotto-reti eterogenee connesse tra loro. Le sotto-reti possono essere basate su tecnologie diverse al Livello 2 (L2) — ad esempio 802.11 (WiFi), 802.3 (Ethernet), tecnologie cellulari (3G/4G), fibra ottica o ADSL — ma comunicano grazie a uno stack di protocolli comune, implementato sopra i diversi livelli fisici e MAC, nella logica del paradigma OSI. Il protocollo di base che abilita l'interoperabilità è l'Internet Protocol (IP), affiancato da protocolli di livello superiore come TCP, UDP e protocolli applicativi (per es. DNS, HTTP).

### 1.1.2 Indirizzamento e instradamento

Ogni dispositivo in una rete IP possiede un identificatore numerico univoco: l'indirizzo IP (32 bit per IPv4, 128 bit per IPv6). Le sotto-reti sono connesse mediante router, dispositivi che effettuano l'inoltro (forwarding) dei pacchetti dalla sorgente alla destinazione seguendo regole presenti nelle tabelle di instradamento.

Il forwarding si basa sul principio del *Longest Prefix Match* (LPM): per decidere quale voce della routing table utilizzare si seleziona la corrispondenza con il prefisso di rete più lungo che include l'indirizzo di destinazione. L'inoltro può essere:

- *diretto*, quando la destinazione si trova nella stessa rete locale;
- *indiretto*, quando la destinazione è raggiungibile tramite un next hop (salto successivo).

Example: 192.168.0.5 and 192.168.0.6 on the same network

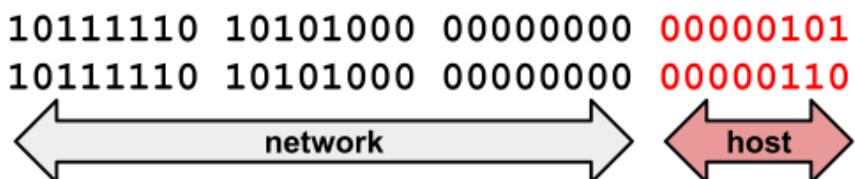


Figura 1: Instradamento diretto: sorgente e destinazione nella stessa rete locale (L2/L3).

Il compito di IP è consegnare il pacchetto alla rete finale; la consegna al dispositivo specifico è rimandata al livello L2, che si occupa della mappatura IP <-> L2 (per esempio tramite ARP per IP <-> MAC).

### 1.1.3 Anatomia dell'indirizzo IP e subnetting

Le reti IP sono suddivise in subnet logiche. Due host appartenenti alla stessa subnet condividono i primi  $X$  bit dell'indirizzo (la *parte di rete*), mentre i restanti  $32 - X$  bit identificano l'host. Dal 1984 si usa CIDR (Classless Inter Domain Routing): il prefisso non è più implicito per classi fisse ma viene specificato tramite una *subnet mask* (o notazione “/length”), che indica quali bit rappresentano la parte di rete. L' $i$ -esimo bit della subnet mask è settato a **0** se l' $i$ -esimo bit è

nella host part; **1** se invece è nel prefisso network .Ad esempio, con indirizzo 192.168.1.12 e maschera 255.255.255.0 (ovvero /24) la rete è 192.168.1.0.

### Example

<input type="checkbox"/> IP address: 192.168.1.12	10111110 10101000 00000001 00001100
<input type="checkbox"/> Network Mask: 255.255.255.0 (aka /24)	11111111 11111111 11111111 00000000

**Network Prefix (address AND mask)**  
**192.168.1.0**

Figura 2: Esempio network prefix

Ogni rete ha due indirizzi ip **RISERVATI** ovvero:

- *Net address* tutti i bits nella parte host sono 0.
- *Broadcast address* tutti i bits nella parte host sono 1.

### Example : find the network and broadcast addresses of host 209.85.129.99/27

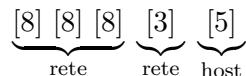
209.85.129.99 (IP addr host)	11010001 01010101 10000001 01100011
255.255.255.224 (Subnet Mask)	11111111 11111111 11111111 11100000
209.85.129.96 (IP addr network)	11010001 01010101 10000001 01100000
209.85.129.127 (IP addr broadcast)	11010001 01010101 10000001 01111111

Figura 3: trovare il network e broadcast address

### Esempio spiegato: trovare rete e broadcast di 209.85.129.99/27

Vogliamo ricavare l'indirizzo di **rete** e di **broadcast** per l'host 209.85.129.99/27.

**1.1.3.1 1) Cosa significa “/27”** La notazione /27 dice che i **primi 27 bit** dell'indirizzo sono di *rete* e i rimanenti sono di *host*. Nel formato “a ottetti”, i primi 24 bit coincidono con i primi tre ottetti; quindi la /27 “entra” nel **quarto ottetto**:



La maschera corrispondente è 255.255.255.224, perché nell'ultimo ottetto i 3 bit di rete valgono  $11100000_2 = 224$ .

**1.1.3.2 2) Perché basta guardare l'ultimo ottetto** Con /27 i primi tre ottetti (209.85.129) restano identici per rete/host/broadcast. Tutta la partizione in sottoreti avviene nel **quarto ottetto**. Qui rimangono **5 bit di host**  $\Rightarrow$  ogni sottorete ha  $2^5 = 32$  indirizzi contigui (un “blocco”).

**1.1.3.3 3) Trova il blocco in cui cade 99** I blocchi nel quarto ottetto sono a passi di 32: 0-31, 32-63, 64-95, 96-127, 128-159, ... Poiché 99 appartiene a 96-127, la nostra **rete** è 209.85.129.96 e il **broadcast** sarà l'ultimo del blocco, 209.85.129.127. Verifichiamo con l'AND bit-a-bit.

#### 1.1.3.4 4) Verifica con l'AND (ultimo ottetto) .

```
[basicstyle=\ttfamily\small,frame=single]
99 = 01100011
224 = 11100000 (maschera /27 nell'ultimo ottetto)
AND -----
01100000 = 96 --> indirizzo di rete (quarto ottetto)
```

Quindi l'indirizzo di rete è 209.85.129.96.

1.1.3.5 5) Broadcast: tutti i bit host a 1 Nel broadcast si *mantengono* i 3 bit di rete e si mettono a 1 i 5 bit di host:

$$\text{rete (ult. ottetto)} = 01100000 + 00011111 (\text{tutti i bit host a 1}) = 01111111 = 127.$$

Dunque broadcast = 209.85.129.127.

1.1.3.6 6) Intervallo host e conteggio Gli host validi sono i numeri compresi *tra* rete e broadcast:

$$209.85.129.97 \text{ fino a } 209.85.129.126,$$

per un totale di  $2^5 - 2 = 30$  host (si escludono rete e broadcast).

#### 1.1.4 Sistemi autonomi (AS) e routing globale

L'inter-rete è organizzata in Autonomous Systems (AS), domini amministrativi che gestiscono internamente le proprie politiche di instradamento. All'interno di un AS si adottano Interior Gateway Protocols (IGP) come OSPF, IS-IS o RIP; lo scambio di rotte tra AS diversi avviene tramite l'Exterior Gateway Protocol più usato: BGP, che garantisce la raggiungibilità globale.

#### 1.1.5 La routing table e l'algoritmo di lookup

La routing table contiene entry costituite tipicamente da: indirizzo di destinazione, maschera/-netmask, next hop e interfaccia d'uscita. La funzione di lookup per un pacchetto  $p$  itera le voci ordinate per lunghezza del prefisso e restituisce la voce  $i$  tale che

$$(p.daddr \& i.mask) = i.addr,$$

dove  $\&$  è l'AND bit-a-bit; se non si trova alcuna corrispondenza il pacchetto viene scartato.

### 1.2 II. Il viaggio del pacchetto: esempio di richiesta DNS

#### 1.2.1 Topologia e hop

Un pacchetto generato da un browser per risolvere un nome (ad esempio `www.google.com`) percorre una serie di hop: rete domestica (WiFi), access point/router, edge router dell'AS dell'utente, router di confine (border router), una sequenza di AS di transito e infine il data center che ospita il servizio DNS o il server web. Ogni tratto può utilizzare tecnologie e politiche diverse, e rappresenta un potenziale punto di vulnerabilità.

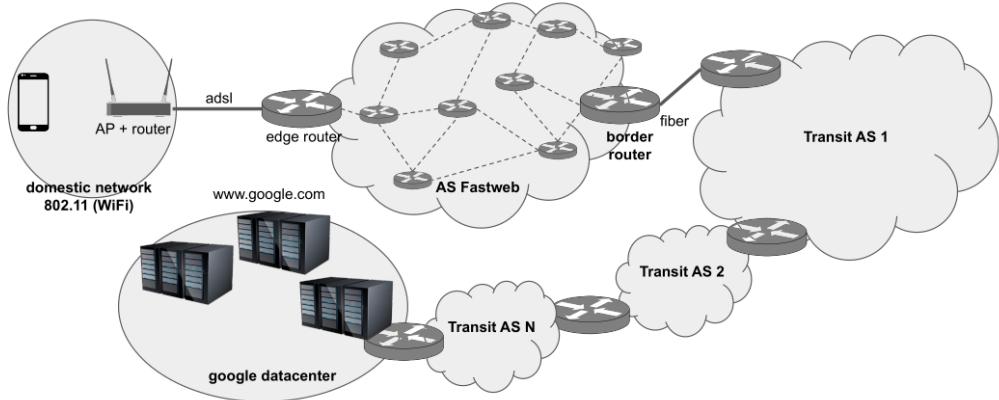


Figura 4: dal web browser al web server

### 1.2.2 Stack protocollare e encapsulamento

La richiesta DNS attraversa gli strati della pila:

- **Livello applicativo (DNS)**: genera la query di risoluzione ("dammi l'indirizzo ip di www.google.com").
- **Livello trasporto (UDP)**: aggiunge porta sorgente (es. 5000) e porta destinazione (53), oltre al checksum.
- **Livello rete (IP)**: inserisce indirizzi IP sorgente e destinazione (es. 10.0.0.100 e 85.18.200.200), TTL, eventuale fragmentation.
- **Livello accesso (L2)**: encapsula il frame con indirizzi MAC del next hop; questi cambiano ad ogni salto.

Durante il percorso gli indirizzi IP rimangono costanti, mentre gli indirizzi MAC vengono aggiornati hop-by-hop. Meccanismi come ARP permettono la traduzione dinamica IP <-> MAC all'interno di una stessa rete locale.

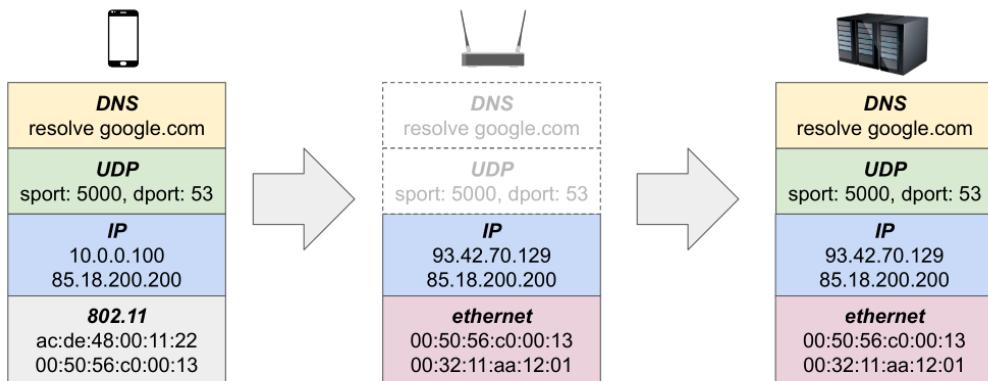


Figura 5: richiesta DNS (semplificata)

### 1.3 III. Vulnerabilità intrinseche di IP e TCP

I protocolli storici di Internet sono stati progettati principalmente per interoperabilità e scalabilità, non per la sicurezza. Questo ha lasciato diverse debolezze intrinseche.

### 1.3.1 Identificazione, spoofing e non-ripudio

Gli **identificatori di rete** (indirizzi IP e MAC) sono semplici stringhe binarie facilmente manipolabili: un mittente può generare pacchetti con sorgente falsata (**IP spoofing**) oppure modificare l'indirizzo sorgente di pacchetti che sta inoltrando. Questo fenomeno rende possibile, ad esempio, l'impersonificazione di server legittimi (un attaccante può inviare pacchetti che sembrano provenire da un DNS server affidabile). **IP non fornisce meccanismi di autenticazione dell'origine**: non esiste un modo intrinseco per dimostrare che l'indirizzo sorgente di un pacchetto corrisponda realmente al mittente fisico, provocando problemi di *repudiation*.

### 1.3.2 Confidenzialità

**Il protocollo IP non cifra il payload né fornisce protezione contro l'intercettazione**: catturare e decodificare pacchetti su un segmento di rete è, in molti casi, semplice. Inoltre gli utenti non controllano l'intero percorso seguito dai pacchetti; anche fidandosi del proprio ISP, sono necessari fiducia e verifiche su tutti gli AS attraversati. Attacchi di route hijacking o route leaking possono alterare il percorso e compromettere la riservatezza.

### 1.3.3 Integrità dei dati

IP, TCP e UDP usano checksum per rilevare errori di trasmissione (header e payload), ma questi meccanismi non sono progettati come primitive di sicurezza: sono vulnerabili a manipolazioni intenzionali poiché basta ricalcolare il checksum dopo la modifica del pacchetto. Per esempio, il checksum IP è una semplice somma/XOR su parole dell'header e non offre garanzie contro un attaccante attivo.

### 1.3.4 Packet replication e anti-replay

A livello IP **non** esistono numeri di sequenza o marcatori univoci che identifichino inequivocabilmente un pacchetto in un flusso; il problema anti-replay è quindi in gran parte non risolto a questo livello. TCP fornisce numeri di sequenza, ma essi sono destinati alla gestione dell'affidabilità e dell'ordine, non all'autenticazione. Poiché tali numeri non sono protetti criptograficamente, possono essere predetti o spoofati in alcuni scenari, consentendo replay o session hijacking se non vengono adottate contromisure a livello superiore (ad esempio TLS).

### 1.3.5 Insicurezza delle mappature dinamiche

Molti servizi critici si basano su mappature dinamiche non progettate per la sicurezza: DNS (nomi→IP), ARP (IP→MAC), tabelle di bridging (MAC→porta), e la stessa routing table (destinazione→next hop). Implementazioni legacy, come il DNS non autenticato, permettono a un attaccante di fornire risposte fasulle: la risoluzione nome→IP non è intrinsecamente verificabile senza meccanismi come DNSSEC.

# Laboratorio

## 1.4 Laboratorio 1: MiTM e DNS spoofing

### 1.4.1 Obiettivo e scenario

L'obiettivo è dirottare richieste HTTP non cifrate attraverso DNS spoofing e impersonificazione di un sito (es. <http://netgroup.uniroma2.it>). Lo scenario tipico prevede un attaccante nella stessa rete locale della vittima; il resolver della vittima è configurato su un DNS pubblico (per es. 8.8.8.8). L'attacco combina:

1. lo sfruttamento della mappatura IP <-> MAC (via ARP spoofing) per stabilire un MiTM;
2. la manipolazione delle risposte DNS per risolvere il nome del sito bersaglio verso un indirizzo controllato dall'attaccante.

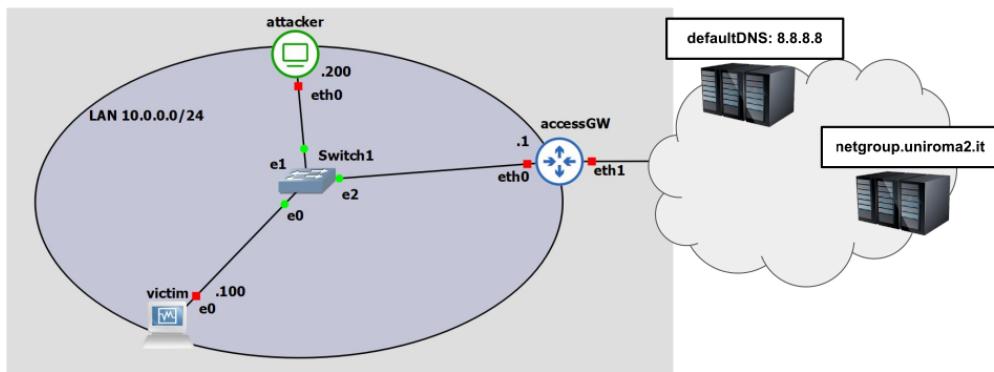


Figura 6: Topologia rete

### 1.4.2 Fasi dell'attacco

L'attacco tipico è articolato in quattro step principali:

1. **STEP 1 – MiTM (ARP spoofing):** l'attaccante dissipa nelle cache ARP della vittima e del gateway risposte ARP falsificate in modo da farsi passare per entrambi e intercettare il traffico.
2. **STEP 2 – Intercettazione della richiesta DNS:** una volta in posizione di MiTM, l'attaccante intercetta le query DNS emesse dalla vittima.
3. **STEP 3 – Spoofing della risposta DNS:** l'attaccante risponde con una risoluzione falsa per il dominio bersaglio, puntando a un IP di sua proprietà.
4. **STEP 4 – Impersonificazione del sito web:** l'attaccante serve una copia del sito (ottenuta tramite mirroring) dall'IP di controllo, così che la vittima riceva contenuti apparentemente legittimi.

### 1.4.3 STEP 1: ARP spoofing (MiTM)

L'attaccante invia risposte ARP non richieste (opcode 2) sia alla vittima che al default gateway:

- alla vittima: un frame ARP unicast indirizzato al MAC della vittima (`bb:bb:bb:bb:bb:bb`) affermando che l'IP del router (10.0.0.1) corrisponde al MAC dell'attaccante (`aa:aa:aa:aa:aa:aa`);

- al gateway: un frame ARP unicast indirizzato al MAC del router (`cc:cc:cc:cc:cc:cc`) affermando che l'IP della vittima (10.0.0.100) corrisponde al MAC dell'attaccante (`aa:aa:aa:aa:aa:aa`).

Ripetendo queste risposte periodicamente, l'attaccante mantiene la posizione di MiTM.

#### 1.4.4 STEP 2 & 3: intercettazione e DNS spoofing

Dopo aver stabilito il MiTM, l'attaccante può reindirizzare le richieste DNS verso la sua macchina:

Listing 1: Esempio: regola iptables per reindirizzare richieste DNS (UDP 53) alla macchina locale

```
iptables -t nat -A PREROUTING -p udp --dport 53 -j REDIRECT
```

Sulla macchina dell'attaccante viene eseguito un server DNS leggero (es. `dnsmasq`) con una configurazione del tipo:

Listing 2: Estratto di /etc/dnsmasq.conf

```
interface=eth0
no-dhcp-interface=eth0
server=1.1.1.1

# Risolvi il dominio bersaglio verso l'IP dell'attaccante
address=/netgroup.uniroma2.it/10.0.0.200
```

Questa configurazione restituisce per `netgroup.uniroma2.it` l'indirizzo 10.0.0.200; tutte le altre query vengono inoltrate al resolver pubblico (qui 1.1.1.1).

#### 1.4.5 STEP 4: impersonificazione del sito web

L'attaccante può aver replicato il contenuto del sito bersaglio tramite strumenti di mirroring, ad esempio:

```
wget --mirror --convert-links --html-extension --no-parent -l 1 \
--no-check-certificate http://netgroup.uniroma2.it
```

I contenuti mirrorati vengono serviti localmente (per es. con Apache2). Così la vittima, ricevendo l'IP dell'attaccante per il dominio richiesto, ottiene una copia apparentemente autentica del sito.

### Conclusione e contromisure (sintesi)

Le vulnerabilità descritte evidenziano che senza meccanismi di autenticazione, integrità e confidenzialità a livello superiore, l'infrastruttura IP/TCP è esposta a compromissioni. Contromisure pratiche includono:

- utilizzo diffuso di canali cifrati e autenticati (TLS/HTTPS) per proteggere le applicazioni;
- adozione di estensioni e protocolli progettati per la sicurezza (es. DNSSEC per autenticare risposte DNS, IPsec per integrità/confidenzialità a livello IP dove applicabile);
- tecniche di difesa a livello di rete locale (ARP inspection, dynamic ARP protection, filtraggio di pacchetti spoofati sui router e access control lists);
- pratiche operative: aggiornamento dei software, monitoraggio delle anomalie di routing e validazione delle rotte BGP.

## 2 NET\_02

### 2.1 Sicurezza delle Reti Ethernet LAN (Livello 2)

#### 2.2 Perché la LAN Ethernet è fragile per natura

Ethernet nasce per *autoconfigurarsi*: gli switch imparano indirizzi e percorsi (MAC learning), i protocolli di controllo (STP, ARP, DHCP) si basano su broadcast e sulla mancanza di autenticazione a L2. Questo rende immediati tre vettori: **osservare** (eavesdropping), **manipolare** (spoofing/MITM) e **interrompere** (DoS). Le minacce si organizzano in quattro famiglie: (i) accesso a rete/sistemi, (ii) confidenzialità, (iii) disponibilità, (iv) integrità.

##### 2.2.1 Frame, indirizzamento e forwarding

Gli indirizzi Mac sono a 48 bit.

nella versione originale dell'ethernet la tipologia del frame veniva usata per il demultiplexing del layer superiore per esempio 0x800=IP; mentre in 802.3 indica la lunghezza oppure il tipo in particolare se il frame supera i  $0x0600$  ( $1536_{10}$ ) allora indica la tipologia di frame altrimenti LLC per il demultiplexing e indica il payload size. Se frame inferiore ai 46 bit allora viene utilizzato del padding.

Il primo bit dell'indirizzo MAC indica se l'indirizzo è unicast (0) o di gruppo/multicast (1); il secondo bit distingue tra indirizzi globali (0, assegnati dal produttore) e locali (1, configurabili via software o driver).

**2.2.1.1 Multiport Repeaters (Hub)** Gli hub, o ripetitori multiporta, operano come un bus condiviso: tutto il traffico ricevuto viene rigenerato su tutte le porte. Appartengono quindi a un unico **dominio di collisione** e non offrono isolamento tra host; per questo sono oggi sostituiti dagli switch.

##### 2.2.1.2 Bridge/Switches :

Gli switch possono operare in:

- **Store&Forward**: lettura completa del frame (memorizzazione su buffer), controllo CRC, scarto dei frame *runt* (<64 bytes too short)/troppo lunghi oppure se fallisce il CRC; look up nella tabella e forwarding.
- **Cut-through**: lettura solo fino all'indirizzo, nessun check di integrità, look-up, forwarding.

Il **Forwarding Database** (FDB) mappa MAC→porta; le entry dinamiche sono apprese (MAC learning) e scadono con *ageing* tipicamente nell'ordine dei 300 s; altrimenti impostate staticamente da un sysadmin o da un db. Se la destinazione è sconosciuta, lo switch effettua *flooding*.

**2.2.1.3 Address Learning** Un frame arriva alla porta X quindi deve provenire dalla LAN connessa dalla porta X, il source address viene usato per l update del forwarding DB. Se arriva un frame da un source addr non presente nella tabella allora viene creata la entry con *age* = 0; se invece arriva un entry già presente viene refreshata la age di quella entry.

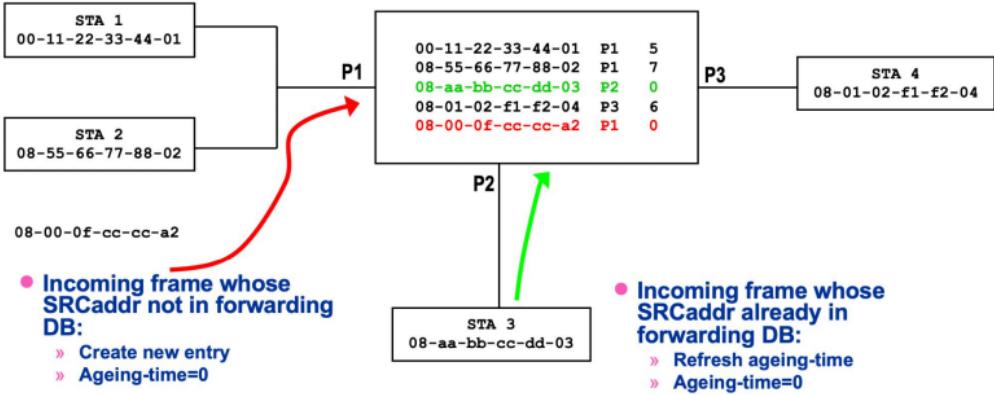


Figura 7: Topologia rete

Infine se arriva un frame da un source addr già presente nella tabella ma sotto una porta differente allora viene aggiornata la entry e refresh della age.

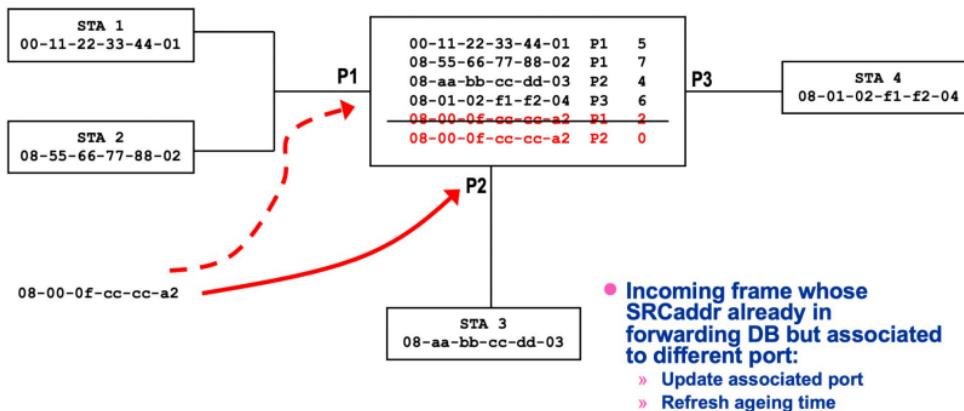


Figura 8: Topologia rete

## 2.2.2 Topologie e controllo dei loop: STP

Quando in una rete Ethernet esistono collegamenti ridondanti, possono formarsi *loop* a **Livello 2 (L2, Data Link)**: i frame possono circolare all'infinito tra switch, saturando la rete. Per evitare questo, entra in gioco lo **Spanning Tree Protocol**. L'idea è eleggere uno *switch radice* (**root bridge**) e disattivare alcune porte in modo che la topologia effettiva sia un albero evitando così loop, pur lasciando i link ridondanti pronti a subentrare in caso di guasti.

Gli switch si mettono d'accordo scambiandosi messaggi di controllo chiamati **BPDU (Bridge Protocol Data Unit)**. Ogni BPDU contiene l'identità dello switch (*Bridge ID*, che include *priorità* e MAC) e i *costi* dei percorsi. Viene eletto un *root bridge*; e poi per ogni altro switch, **STP** sceglie:

- una **porta radice (root port)**: la porta verso il root bridge con costo minore;
- eventuali porte in eccesso vengono messe in stato **bloccato (blocking)** per rompere i cicli.

Il risultato è che solo alcune porte sono di forwarding, mentre le altre restano **bloccate**; se un link o uno switch si guastano allora STP riconfigura la topologia evitando nuovamente loop.

Sul piano della sicurezza, però, STP ha un limite: a L2 non c'è **autenticazione** dei messaggi di controllo. Un host malevolo può inviare **BPDU** artefatto e farsi eleggere *root bridge* (alzando la priorità o manipolando i costi), dirottando o interrompendo il traffico. Per questo, in produzione si usano contromisure come *BPDU Guard*, *Root Guard*, *portfast* solo sugli host, e piani di controllo isolati.

### 2.2.3 Adattamento del Livello 3 su Livello 2: DHCP, ARP e NDP

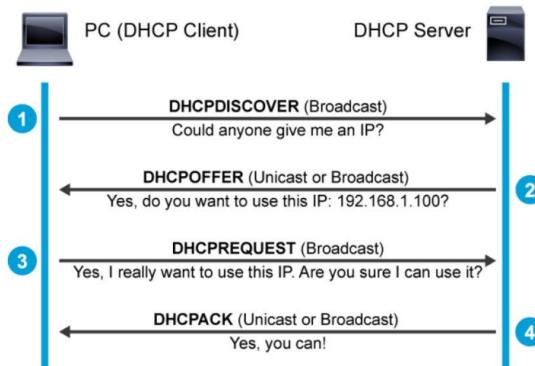
L'interoperabilità tra il **Livello 2 (Data Link)** e il **Livello 3 (Network)** è assicurata da una serie di protocolli di adattamento che consentono agli host di configurare automaticamente i propri parametri IP e di risolvere gli indirizzi fisici (MAC) dei dispositivi vicini. Questi protocolli, fondamentali per il funzionamento delle reti IP, nascono però in un contesto di fiducia implicita e **assenza di autenticazione**, diventando quindi bersagli ideali per attacchi di spoofing e manipolazione del traffico.

**2.2.3.1 DHCP — Dynamic Host Configuration Protocol (IPv4).** Il **DHCP** automatizza l'assegnazione degli indirizzi IP e dei parametri di rete,. Opera in modalità *client-server* e sfrutta il meccanismo di broadcasting a livello Ethernet per raggiungere il server anche quando il client non ha ancora un IP.

Il protocollo segue un tipico **handshake a quattro fasi**:

1. **DHCP Discover** — il client trasmette in broadcast (255.255.255.255) per cercare un server disponibile;
2. **DHCP Offer** — il server risponde offrendo un indirizzo IP e altri parametri (gateway, DNS, lease time);
3. **DHCP Request** — il client accetta esplicitamente un'offerta specifica;
4. **DHCP ACK** — il server conferma l'assegnazione e crea una voce di *lease* nel proprio database.

Ogni lease è temporaneo e può essere rinnovato tramite messaggi *DHCP Renew/Rebind*. Quando un host si trova in una rete diversa dal server, la comunicazione avviene tramite un **DHCP relay agent** — spesso un router — che incapsula i messaggi Discover/Request e li inoltra verso il server remoto, mantenendo così la visibilità dell'origine (campo *giaddr*).



**Sicurezza:** DHCP non autentica né il client né il server. Un host malevolo può rispondere più velocemente del server legittimo (**DHCP spoofing**) e assegnare gateway o DNS controllati,

dirottando il traffico. Per mitigare questi scenari, gli switch moderni implementano **DHCP Snooping**: una funzione che registra le associazioni IP–MAC–porta–VLAN apprese dai messaggi DHCP legittimi, utile anche per alimentare altri controlli di sicurezza come la *Dynamic ARP Inspection*.

**2.2.3.2 ARP — Address Resolution Protocol (IPv4).** L'ARP consente di scoprire l'indirizzo MAC associato a un indirizzo IP all'interno della stessa LAN. Quando un host deve inviare un pacchetto IP verso una destinazione della propria subnet, interroga la rete inviando un messaggio **ARP Request** in broadcast contenente l'indirizzo IP cercato. L'host corrispondente risponde con un **ARP Reply** unicast, fornendo il proprio MAC address. Ogni sistema mantiene una **cache ARP** che memorizza temporaneamente queste associazioni per ridurre il numero di richieste future (tipicamente 20 minuti su sistemi UNIX-like).

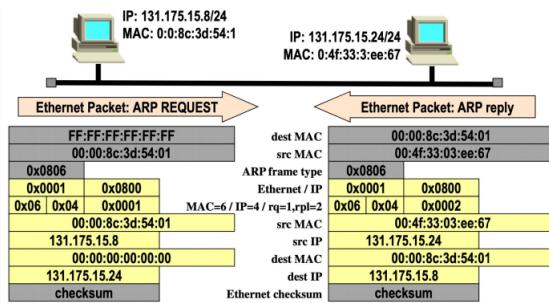


Figura 9

**Debolezze:** ARP è *stateless* e non prevede autenticazione. Un attaccante può quindi inviare **ARP Reply falsificati** (anche senza richiesta) per associare un IP legittimo al proprio MAC address: è il classico **ARP poisoning**, che consente di intercettare, modificare o bloccare il traffico (attacco *Man-in-the-Middle*). Meccanismi come **Dynamic ARP Inspection (DAI)**, basati sulle tabelle di DHCP Snooping, permettono di bloccare risposte ARP incoerenti rispetto alle associazioni IP–MAC note.

**2.2.3.3 NDP — Neighbor Discovery Protocol (IPv6).** Nel mondo IPv6, il **Neighbor Discovery Protocol (NDP)**, sostituisce ARP e parte del ruolo di DHCP. Basato su **ICMPv6** A differenza di ARP, NDP usa **multicast** anziché broadcast, riducendo l'impatto sulla rete e migliorando la scalabilità. Tuttavia, condivide la stessa assenza di autenticazione: un host può fingere di essere un router o un vicino legittimo, portando a *neighbor spoofing* o *router advertisement flooding*.

## 2.2.4 Vulnerabilità e minacce principali

La sicurezza di una rete Ethernet dipende fortemente dall'affidabilità del Livello 2, ma i protocolli su cui si basa — come ARP, DHCP e STP — sono stati progettati per un ambiente fidato, senza autenticazione o cifratura. Questo rende l'intera architettura LAN intrinsecamente vulnerabile ad attacchi che puntano al controllo dell'accesso fisico, alla manipolazione del traffico e al degrado delle prestazioni. Le principali minacce si raggruppano in quattro categorie: **(1) accesso alla rete, (2) riservatezza del traffico, (3) integrità e manipolazione, (4) disponibilità e prestazioni**.

### 2.2.4.1 Accesso alla rete .

**Accesso fisico non autorizzato.** L'attacco più basilare consiste nel collegarsi fisicamente a una porta Ethernet lasciata attiva e non monitorata. In ambienti non presidiati (aula, uffici, open space), un attaccante può semplicemente inserire il proprio dispositivo o un piccolo switch/access point (*rogue device*), espandendo la rete interna senza autorizzazione. Poiché lo standard Ethernet non prevede autenticazione a livello di porta, la connessione risulta immediatamente operativa. Questo tipo di “*join fisico*” rappresenta il punto d'ingresso di molte compromissioni LAN.

**Accesso remoto e ricognizione.** Una volta ottenuto l'accesso (fisico o logico), l'attaccante può mappare la topologia di rete sfruttando protocolli di base:

- *ARP scanning* — per enumerare gli indirizzi IP attivi nella LAN;
- richieste *DHCP* — per dedurre l'intervallo di indirizzi disponibili e i parametri di rete (gateway, DNS);
- *port scanning* — per identificare i servizi esposti e i sistemi operativi in uso.

Queste informazioni consentono di costruire una mappa logica della rete e di selezionare successivi obiettivi di attacco (*target profiling*).

**2.2.4.2 Compromissione e controllo dello switch.** Gli switch possono diventare obiettivi diretti se il *management plane* è esposto o debolmente protetto. Molti dispositivi arrivano con **credenziali di default** (o addirittura senza password) e spesso prevedono un **reset fisico** che ripristina i parametri di fabbrica: in entrambi i casi un attaccante può ottenere l'accesso amministrativo. Una volta dentro, è possibile **dirottare il traffico** abbassando link strategici, **farsi eleggere root bridge** in *Spanning Tree* aumentando la priorità dello switch, o **indurre DoS** su link selezionati. Inoltre, a seconda dei *protocolli di gestione* abilitati, l'attaccante può **attivare il port mirroring** per intercettare i flussi e (*in alcuni scenari*) ottenere visibilità o accesso a VLAN non previste.

Dal lato protocolli L2, l'assenza di autenticazione a livello Ethernet rende possibile **manipolare STP** (fingendosi switch legittimo) per alterare la topologia o causare riconvergenze continue: è la base di diversi vettori di *denial-of-service*.

**2.2.4.3 Riservatezza e intercettazione.** **Eavesdropping (intercettazione).** In una rete Ethernet tradizionale basata su hub, tutto il traffico viene propagato su tutte le porte, rendendo facile l'intercettazione passiva (*sniffing*). Anche negli switch moderni, l'attacco resta possibile tramite:

- installazione fisica di un dispositivo di ascolto (*tap*) su un cavo in rame o fibra ottica;
- abilitazione di una scheda di rete in *promiscuous mode* per ricevere frame non destinati al proprio MAC (annulla filtering MAC);
- abuso della funzione di *port mirroring* sugli switch, utile per intercettare il traffico di altre porte.

**MAC flooding.** Gli switch memorizzano le associazioni MAC→porta nella CAM (Content Addressable Memory) o FDB (Forwarding Database). Un attaccante può saturare questa memoria inviando migliaia di frame con indirizzi MAC sorgente casuali. Quando la tabella si riempie, lo switch passa alla modalità *flooding*, inoltrando i frame su tutte le porte come un hub. Questo permette di catturare traffico unicast normalmente privato e, in certi casi, di rompere l'isolamento tra VLAN (*cross-VLAN leakage*).

**MAC spoofing.** Manipolando l'indirizzo MAC sorgente dei frame inviati, un attaccante può sostituirsi a un host legittimo già presente nella FDB. Il risultato è un *hijacking* del traffico diretto alla vittima, che può essere intercettato, modificato o semplicemente bloccato. Lo spoofing è

efficace perché Ethernet non verifica la coerenza tra il MAC dichiarato nel frame e quello della scheda di rete.

#### 2.2.4.4 Integrità del traffico e attacchi Man-in-the-Middle (MITM)

**ARP poisoning.** Approfittando del fatto che l'ARP accetta qualunque risposta non autenticata, e anche quelle non richieste, un attaccante può inviare *ARP replies* falsificati per associare il proprio MAC address all'indirizzo IP di un altro nodo (ad esempio il gateway). In questo modo intercetta tutto il traffico tra la vittima e il router, realizzando un classico attacco *Man-in-the-Middle (MITM)*. Varianti di questo attacco esistono anche in IPv6 sotto forma di *Neighbor Advertisement spoofing* contro NDP.

**DHCP poisoning.** Simile nel principio, l'attacco DHCP poisoning consiste nel rispondere alle richieste DHCP più rapidamente del server legittimo. Il client riceve così parametri falsi (indirizzo IP, gateway, DNS), che permettono all'attaccante di deviare il traffico verso sistemi controllati o intercettarlo.

**Session hijacking.** Una volta intercettato il traffico (via ARP o DHCP poisoning), è possibile analizzare le sessioni di livello superiore (ad esempio TCP) e riprodurle, utilizzando numeri di sequenza o cookie d'autenticazione per impersonare un utente o un servizio.

**Replay attack.** Consiste nel riutilizzare pacchetti validi intercettati in precedenza — ad esempio messaggi di controllo o di autenticazione — per ottenere accesso o indurre comportamenti anomali nei dispositivi di rete. In mancanza di firme digitali o timestamp, questi messaggi vengono accettati come legittimi.

#### 2.2.4.5 Disponibilità e attacchi di Denial of Service (DoS)

**STP DoS e manipolazione topologica** Il protocollo **Spanning Tree Protocol** non prevede autenticazione dei messaggi di controllo (**BPDU**, **Bridge Protocol Data Units**). Un attaccante può sfruttare questa debolezza per:

- eleggersi come *root bridge* forzando il traffico a passare attraverso il proprio nodo;
- inondare la rete di BPDU falsi, causando continui ricalcoli dell'albero di spanning e interruzioni periodiche dei collegamenti (*STP reconvergence loops*).

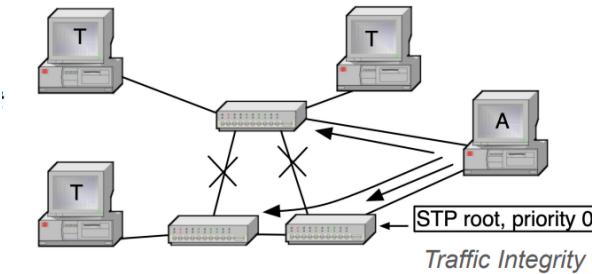


Figura 10

**Resource exhaustion e flooding.** Un altro vettore comune consiste nel sovraccaricare il piano di controllo degli switch o dei router. Attacchi di *unknown-unicast flooding* e tempeste di broadcast (*broadcast storms*) possono saturare la banda o la memoria, impedendo il corretto

forwarding dei frame. Simili risultati possono essere ottenuti generando una quantità eccessiva di richieste ARP o DHCP, portando al collasso del servizio (*DoS a livello L2*).

### 2.2.5 Contromisure: dal minimo sindacale al robusto

Per affrontare le vulnerabilità strutturali di Ethernet e le minacce che ne derivano, è fondamentale adottare un approccio stratificato. Le contromisure spaziano da soluzioni basilari, come la segmentazione della rete, a soluzioni avanzate che includono l'uso di crittografia L2 e l'autenticazione a livello di porta.

**2.2.5.1 Router-based security (segmentazione L3).** Sostituire (o affiancare) lo switch centrale con un **router IP** spezza la LAN in più *segmenti L2* separati (domini di broadcast distinti). Questo ha due effetti chiave:

1. **blocca i protocolli di controllo L2** e tutto il traffico broadcast/multicast tra segmenti (ARP, STP, DHCP non attraversano il confine L3 a meno di funzioni dedicate),
2. rende **impossibili gli attacchi L2 tra segmenti** (eavesdropping, MAC-table attacks, VLAN hopping, MITM via ARP/DHCP) perché gli header MAC si fermano al router e l'instradamento avviene sul piano IP. Le stesse minacce restano possibili *all'interno* di ciascun segmento se questo continua a essere uno switch L2.

*Perché funziona.* Il confine L3 elimina la connettività a livello Ethernet tra i segmenti: un host in segmento A non può inviare ARP o BPDU verso segmento B, né può “floodare” unicast sconosciuti oltre il router. Di conseguenza, **ARP spoofing**, **MAC flooding**, manipolazioni **STP** e **VLAN hopping** non propagano tra segmenti diversi. Il traffico inter-segmento è visibile solo sul router (o firewall), dove si possono applicare ACL e ispezioni più robuste.

Listing 3: Schema: ARP confinato dal confine L3 (il router non inoltra ARP)

```
[Host A 10.0.10.23] -- (Switch L2 VLAN10) -- [Router L3] -- (Switch L2 VLAN20) -- [Host B 10.0.20.45]

Domanda: A vuole parlare con 10.0.20.45 -> prima deve risolvere MAC.
- ARP Request "Chi ha 10.0.20.45?" \e un BROADCAST L2 (ff:ff:ff:ff:ff:ff).
- Il BROADCAST rimane nel DOMINIO L2 (VLAN10). Il router L3 NON inoltra ARP.
- A ARP-a solo il suo gateway (10.0.10.1) per uscire dal segmento; il resto \e routing IP.
Conclusione: ARP spoofing/flooding resta confinato entro la VLAN/segmento L2.
```

Listing 4: Schema: DHCP con relay (giaddr) oltre il confine L3

```
Client (senza IP) Switch/VLAN10 Router L3 (DHCP Relay) DHCP Server (172.16.0.10)

0.0.0.0:68 --[BCAST]--> DISCOVER --[riceve]--> aggiunge giaddr=10.0.10.1 --[UNICAST UDP/67]--> Server
                                                <---[UNICAST UDP/67]--- OFFER (
                                                dest: 10.0.10.1)
Relay inoltra in VLAN10 come BROADCAST verso il client (UDP/68)
Client --[BCAST]--> REQUEST --> Relay --[UNICAST]--> Server
                                                <---[UNICAST]--- ACK (lease, GW,
                                                DNS)
Relay inoltra l'ACK in VLAN10 (di solito come broadcast) al client.
Esito: lease IP (es. 10.0.10.50), gateway 10.0.10.1, DNS..., registrati dal server.
Nota: senza relay, i messaggi DHCP non attraversano il confine L3.
```

## 2.2.6 Controllo d'accesso (Access Control)

**2.2.6.1 Obiettivo.** (i) autenticazione all'ingresso, (ii) autorizzazione fine a cosa può fare l'endpoint, (iii) enforcement sullo switch con ACL/Port Security.

**2.2.6.2 802.1X (port-based NAC).** **Supplicant** (host), **Authenticator** (switch) e **Authentication Server** (tipicamente RADIUS) negoziano l'accesso usando **EAP** (extensible authentication protocol) veicolato su **EAPOL** (EAP over LAN). 802.1X supporta credenziali diverse (username/password o certificati) e lega l'identità alla *porta* di switch all'inizio della sessione; il controllo si appoggia a RADIUS (EAP relay/termination) e ai tipi di messaggi EAPOL (Start, EAP, Key, Logoff). In tal modo lo switch può ammettere/negare la porta e associare policy all'identità autenticata.

*Effetti attesi e limiti.* Gli switch *802.1X-capable* mitigano *MAC spoofing* e *flooding* legando il MAC alla porta autenticata; restano però possibili attacchi fuori dal perimetro di 802.1X (es. *ARP poisoning*) e il *piggybacking* inserendo un piccolo hub/switch tra host e porta autenticata. L'autenticazione tra switch può creare un “inner core” fidato per impedire che un host si finga switch.

**2.2.6.3 Autorizzazione dinamica con 802.1X.** Dopo l'*auth*, il server può spingere parametri di **autorizzazione**:

- **VLAN per-utente** (membership assegnata dall'identità);
- **ACL per-utente** (policy di filtro associate all'identità);
- **UCL (User Control List)**: gruppi di utenti che condividono una stessa policy/ACL.

Queste informazioni sono veicolate nell'esito dell'autenticazione e consumate dallo switch per configurare forwarding e filtri coerenti con l'identità.

**2.2.6.4 Port Security & ACL sugli switch.** **Port Security** limita il numero (e optionalmente l'identità) dei MAC appresi per porta: blocca il *MAC flooding* e rende più difficile l'espansione non autorizzata della LAN aggiungendo switch clandestini. Le **ACL** a livello Ethernet non sono parte dello standard, ma gli switch moderni possono filtrare su MAC sorgente/destinazione o **Ethertype**; gli switch L3 estendono il match a campi L3/L4. In laboratorio, le ACL L2 si possono esprimere con **ebtables** (NETFILTER).

Listing 5: Esempio minimale di binding MAC→porta con ebtables

```
ebtables -A FORWARD --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP  
ebtables -A FORWARD --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP  
ebtables -A FORWARD --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP
```

(Le ACL del laboratorio usano **ebtables**/NETFILTER; le regole si applicano per catena e per livello.)

**2.2.6.5 Segmentazione con VLAN (complemento).** Le **VLAN 802.1Q** riducono il dominio di broadcast e separano il traffico L2; l'efficacia dipende dalla corretta configurazione (trunk, VLAN nativa, porte *access*). È pratica comune associare la VLAN all'identità 802.1X (VLAN dinamiche) e poi controllare il traffico inter-VLAN a L3. *Nota: i default non sono sicuri; configurazioni errate possono abilitare VLAN hopping.*

# Laboratorio

## 2.3 Laboratorio 2A: ACL L2 con Linux Bridge ed `ebttables` (binding MAC→porta)

Questo laboratorio mostra come applicare un controllo d'accesso a Livello 2 vincolando, per ogni porta, i soli indirizzi MAC ammessi ( $MAC \rightarrow porta$ ). L'obiettivo è accettare frame in ingresso solo dagli host attesi e bloccare tentativi di *MAC flooding/spoofing*.

### Topologia di rete

Un host Linux con tre interfacce (`eth0`, `eth1`, `eth2`) funge da *bridge L2* (“switch” emulato). Su ciascuna porta è previsto un client:

- **client-1** collegato a `eth0`, MAC previsto `a0:a0:a0:a0:a0:a0`;
- **client-2** collegato a `eth1`, MAC previsto `b0:b0:b0:b0:b0:b0`;
- **client-3** collegato a `eth2`, MAC `c0:c0:c0:c0:c0:c0` da *non* accettare.

**Goal:** accettare solo i MAC previsti su ciascuna porta; verificare che il traffico in ingresso da **client-3** sia scartato.

### Configurazione di esempio

**2.3.0.1 1) Preparazione host (MAC/IP dei client).** Esempio di impostazione (lato client) dei MAC e degli IP:

Listing 6: Impostazione MAC/IP sui client (esempio)

```
# client-1
ip link set dev eth0 address a0:a0:a0:a0:a0:a0
ip addr add 10.0.0.1/24 dev eth0

# client-2
ip link set dev eth0 address b0:b0:b0:b0:b0:b0
ip addr add 10.0.0.2/24 dev eth0

# client-3 (non ammesso)
ip link set dev eth0 address c0:c0:c0:c0:c0:c0
ip addr add 10.0.0.3/24 dev eth0
```

(Questi valori riflettono quelli delle slide del laboratorio.)

**2.3.0.2 2) Creazione del bridge L2 (lato “switch” Linux).** .

Listing 7: Bridge L2 con tre porte

```
# crea il bridge e collega le porte
ip link add name bridge type bridge
ip link set bridge up
ip link set dev eth0 master bridge
ip link set dev eth1 master bridge
ip link set dev eth2 master bridge
```

Tutte le porte sono ora nello stesso dominio L2; senza ACL, i tre client comunicano liberamente.

### 2.3.0.3 3) ACL L2 con ebttables (NETFILTER). .

Listing 8: Binding MAC→porta con ebtables

```
# accetta su eth0 solo il MAC di client-1
ebtables -A FORWARD --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP
# accetta su eth1 solo il MAC di client-2
ebtables -A FORWARD --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP
# blocca su eth2 il MAC di client-3
ebtables -A FORWARD --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP

# opzionale: replica le stesse policy anche sulla chain INPUT del bridge-host
ebtables -A INPUT --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP
ebtables -A INPUT --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP
ebtables -A INPUT --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP
```

Le ACL sono espresse nel framework **NETFILTER**: ebttables opera a L2 (MAC/EtherType), iptables/ip6tables a L3/L4.

### Funzionamento del laboratorio

Le regole applicano un vincolo di identità L2: ogni porta accetta solo i frame dal MAC atteso; frame da MAC diversi vengono droppati (*anti-spoofing* di base, freno al *flooding*).

### Verifica del comportamento

- **client-1 ↔ client-2**: ping riuscito.
- **client-3 → altri**: i pacchetti in ingresso sono scartati (nessuna risposta al ping).

Suggerimento: osserva i contatori `ebtables -L -Lc` mentre invii traffico di test.

### Osservazione pratica

Questo schema è didattico ma realistico: i vendor implementano meccanismi simili (*Port Security*, ACL L2) sugli switch per contenere spoofing/flooding alla porta.

## 2.4 Laboratorio 2B: MACsec su Linux (confidenzialità, integrità e anti-replay a L2)

Questo laboratorio mostra come attivare **MACsec** (IEEE 802.1AE) tra due host Linux per proteggere il traffico L2 con integrità, optionalmente cifratura e protezione *anti-replay*.

### Topologia di rete

Due host (**client1**, **client2**) collegati alla stessa LAN Ethernet. Ogni host crea un'interfaccia `macsec0` legata alla NIC fisica (`eth0`) e configura *Secure Channel* (SC) e *Security Association* (SA) con chiavi simmetriche.

### Configurazione di esempio

#### 2.4.0.1 Host A (client1). .

Listing 9: MACsec lato client1

```
ip link add link eth0 macsec0 type macsec
```

```
# SA TX di client1 (chiave K1), SA RX verso il MAC di client2 (chiave K2)
ip macsec add macsec0 tx sa 0 pn 1 on key 01 09876543210987654321098765432109
ip macsec add macsec0 rx address b0:b0:b0:b0:b0:b0 port 1
ip macsec add macsec0 rx address b0:b0:b0:b0:b0:b0 port 1 sa 0 pn 1 on key 02
    12345678901234567890123456789012
ip link set macsec0 up
ip addr add 10.100.0.1/24 dev macsec0
```

#### 2.4.0.2 Host B (client2) . .

Listing 10: MACsec lato client2

```
ip link add link eth0 macsec0 type macsec
# SA TX di client2 (chiave K2), SA RX verso il MAC di client1 (chiave K1)
ip macsec add macsec0 tx sa 0 pn 1 on key 02 12345678901234567890123456789012
ip macsec add macsec0 rx address a0:a0:a0:a0:a0:a0 port 1
ip macsec add macsec0 rx address a0:a0:a0:a0:a0:a0 port 1 sa 0 pn 1 on key 01
    09876543210987654321098765432109
ip link set macsec0 up
ip addr add 10.100.0.2/24 dev macsec0
```

#### 2.4.0.3 Opzioni di sicurezza e test. .

Listing 11: Cifratura e anti-replay; test con ping

```
# Cifratura dei frame e protezione anti-replay (opzionali)
ip link set macsec0 type macsec encrypt on
ip link set macsec0 type macsec replay on

# Test
ping 10.100.0.2 # da client1 (verifica su Wireshark l'intestazione MACsec)
```

Con questa configurazione, MACsec fornisce integrità; attivando `encrypt on` aggiungi confidenzialità. La suite predefinita è *GCM-AES-128*; *GCM-AES-256* è disponibile in estensione. La gestione chiavi dinamica è demandata a 802.1X-2010 (fuori dallo scopo di questo lab).

### Funzionamento del laboratorio

MACsec inserisce, tra MAC header e payload, un *Security Tag* e calcola un ICV (codice d'integrità); i frame risultano protetti contro manomissioni e, se abilitata, cifrati in transito.

### Verifica del comportamento

- **Integrità only:** con `encrypt off` vedi i Layer-3 in chiaro ma i frame hanno tag/ICV MACsec.
- **Cifratura on:** abilita `encrypt on` e osserva in Wireshark che il payload non è leggibile.
- **Anti-replay:** abilita `replay on` e verifica che ritrasmissioni artificiose vengano scartate.

Suggerimento: cattura sul link fisico `eth0` per osservare l'incapsulamento MACsec.

## Osservazione pratica

MACsec elimina *sniffing/MITM* sul filo (anche in caso di flooding per timeout CAM), ma non impedisce DoS né abusi da host già autorizzati: serve comunque *hardening* (ACL/DAI/Port Security) e monitoraggio.

### 2.4.1 Secure Address Resolution (IPv4/IPv6)

**2.4.1.1 IPv4: dal piano di switching ai vincoli per-presa.** **DHCP Snooping** costruisce una tabella (IP, MAC, porta, VLAN) osservando i messaggi DHCP leciti; questa base di conoscenza alimenta la **Dynamic ARP Inspection (DAI)** che blocca risposte ARP incoerenti con i binding noti. **IP Source Guard** applica il binding direttamente a L2, filtrando i frame con IP sorgente non atteso su quella porta. *Limite pratico:* lo *scope* è per-switch; un apparato vede solo i lease che attraversano le sue porte. Progettare i domini DHCP in modo che il traffico passi dove serve lo snooping.

**2.4.1.2 IPv4: proposte crittografiche.** **S-ARP** estende ARP con un campo di autenticazione e un'infrastruttura di gestione chiavi; in alternativa, si può “estendere” la protezione L2 fornita da **MACsec** fino all'endpoint e al multicast. Queste soluzioni sono robuste sul piano tecnico, ma più complesse da distribuire.

**2.4.1.3 IPv6: NDP sicuro.** In IPv6, la sicurezza della risoluzione passa da **SEND (Secure Neighbor Discovery, RFC 3971/6494)**: usa **CGA** (Cryptographically Generated Addresses) e opzioni firmate per autenticare i messaggi NDP, fornendo un meccanismo alternativo a IPsec, specifico per la discovery. Nella pratica operativa, resta poco adottato; in molti contesti si preferiscono controlli sullo switch (es. RA/DHCPv6 Guard) per mitigare *router/neighbor spoofing*.

## 2.4.2 Security Monitoring

**2.4.2.1 Firewall & DPI.** I firewall delimitano il traffico tra segmenti (caso avanzato di ACL) e, nei prodotti moderni, possono operare a più livelli con **Deep Packet Inspection** e ricostruzione di sessione applicativa. Nel dominio L2, il concetto di “Ethernet firewall” è assorbito da *ACL di switch* e dallo stack *NETFILTER* (ebtables/iptables); i firewall tradizionali gestiscono i livelli superiori.

**2.4.2.2 IDS/IPS e accesso al traffico.** **IDS/IPS** identificano attacchi tramite firme/anomalie e necessitano di vedere i pacchetti: o in *inline* (come un firewall) oppure tramite **port mirroring (SPAN)**, che copia il traffico di porte selezionate verso una porta di ascolto. I tap passivi su rame/fibra sono alternative poco rilevabili, ma richiedono accesso fisico.

**2.4.2.3 Dove il monitoring aiuta davvero.** Il mirroring/IDS rende *osservabili* attacchi tipici L2 (es. *MAC flooding, double-tagging*): molte firme sono facili da catturare; l'analisi DPI può correlare eventi di *poisoning* e *replay*. In parallelo, le *ACL* sugli switch limitano la superficie a L2.

**2.4.2.4 Messaggio chiave (dalle slide).** Il livello di sicurezza cresce con l'impegno amministrativo: una Ethernet *out-of-the-box* resta insicura (MAC flooding, ARP spoofing, STP attacks sono banali). Anche con ACL/802.1X, l'ARP spoofing resta possibile senza *DHCP Snooping + ARP Inspection*; MACsec risolve *sniffing/MiTM* ma non *DoS/analisi del traffico*. *Conclusione:* servono difese stratificate e configurazioni accurate.

### 3 NET\_03 — Virtual LANs (VLAN)

#### 3.0.1 Definizione e motivazione

Gli **switch Ethernet** tradizionali segmentano i domini di collisione ma non i domini di broadcast. Ciò significa che tutti gli host connessi a uno switch appartengono allo stesso dominio di broadcast e ricevono tutti i frame inviati in broadcast (come richieste ARP o DHCP).

Questa architettura è semplice ma poco scalabile: in reti di grandi dimensioni, il traffico broadcast può saturare la banda disponibile e ogni problema (come un loop o un attacco) può propagarsi a tutti gli host della rete.

Per risolvere questo limite si introduce il concetto di **Virtual LAN (VLAN)**, cioè la creazione di *più domini di broadcast logici* all'interno della stessa infrastruttura fisica. Ogni VLAN rappresenta una “rete virtuale” indipendente, isolata logicamente dalle altre pur condividendo gli stessi apparati.

#### 3.0.2 Limiti delle reti fisicamente separate

Storicamente, la separazione dei domini di broadcast veniva realizzata con **sottoreti IP fisiche**, ciascuna connessa a un proprio switch e router. Tuttavia questo approccio presenta diversi limiti:

- necessità di cablaggi distinti e di apparati separati per ogni subnet anche se gli switch sono sullo stesso piano (fisico) come mostrato in Figura 11;
- difficoltà di riconfigurazione in caso di spostamento di host tra subnet diverse;
- costi di gestione e manutenzione elevati.

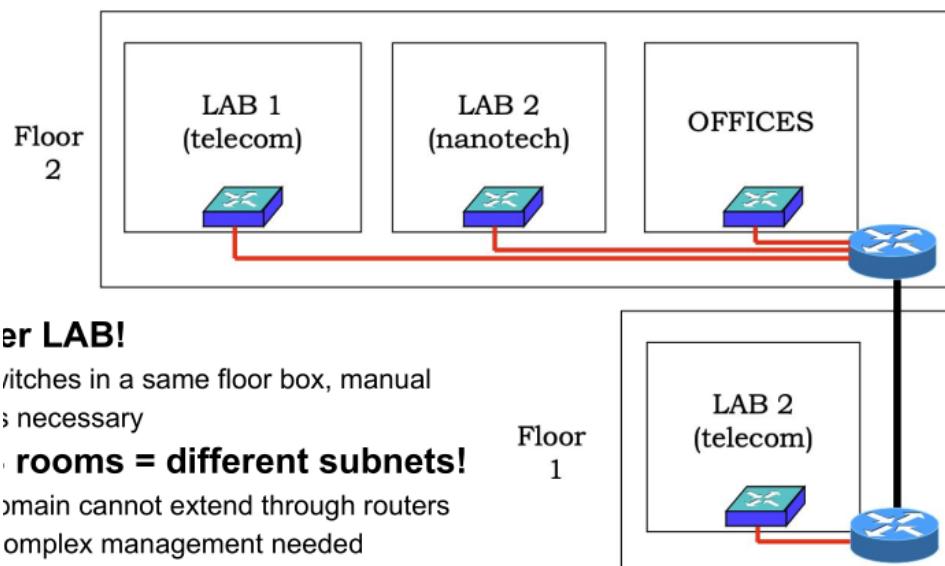


Figura 11: Physical ip subnet

Con l'introduzione degli **switch di Layer 3**, la velocità di routing non è più un problema, ma resta la necessità di una gestione logica più flessibile. Le VLAN forniscono questa flessibilità permettendo di isolare logicamente i gruppi di dispositivi in base a criteri funzionali, e non fisici.

#### 3.0.2.1 Benefici principali delle VLAN

- **Confinamento del broadcast:** il traffico broadcast resta confinato all'interno della VLAN di appartenenza.
- **Scalabilità e ordine:** la rete può essere gestita come un insieme di domini separati, semplificando la diagnostica.
- **Sicurezza:** la separazione logica riduce la superficie d'attacco e impedisce la propagazione di minacce L2 tra gruppi diversi.

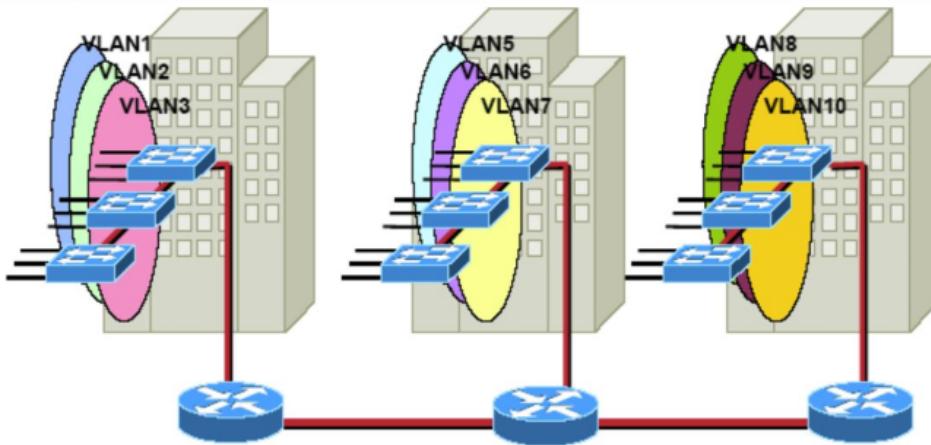


Figura 12: VLAN=area che limita il broadcast domain

### 3.1 Assegnazione e membership delle VLAN

#### 3.1.1 Criteri di assegnazione

Un dispositivo può essere assegnato a una VLAN in base a diversi criteri:

- **Per porta (Port-based VLAN):** lo switch associa staticamente una VLAN a ogni porta. È il metodo più comune e lo standard definito da IEEE 802.1Q.
- **Per MAC address (User-based VLAN):** la VLAN è determinata dal MAC del dispositivo o dall'identità dell'utente autenticato (es. via 802.1X).
- **Per protocollo (Protocol-based VLAN):** introdotto da IEEE 802.1v, assegna la VLAN in base al protocollo di livello 3 (IP, IPX, ecc.).
- **Combinato (Cross-layer):** alcune implementazioni permettono regole gerarchiche, ad esempio prima per protocollo, poi per MAC, e infine per porta.

#### 3.1.2 Vista logica

Ogni VLAN rappresenta un dominio di broadcast indipendente e, di conseguenza, è normalmente associata a una **sottorete IP dedicata**. La comunicazione tra VLAN diverse richiede un dispositivo L3 (router o Layer 3 switch (figura 13)) che svolga la funzione di *inter-VLAN routing*.

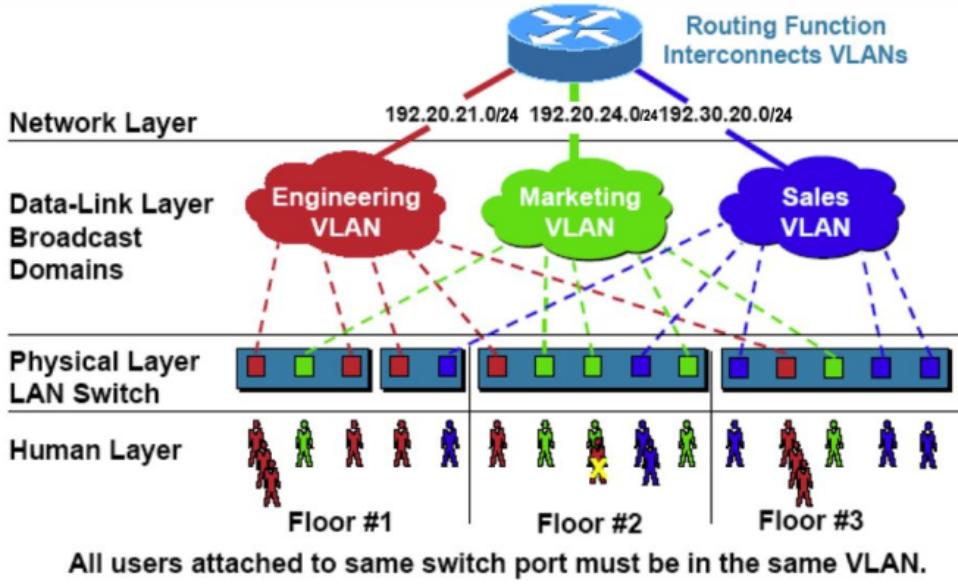


Figura 13: Physical vs logical view

**3.1.2.1 Router “one-armed”** In una configurazione one-armed router, una **singola interfaccia fisica** del router viene utilizzata per gestire **più VLAN** contemporaneamente. Ciò avviene tramite la creazione di **sub-interfacce virtuali**, ognuna associata a una VLAN specifica.

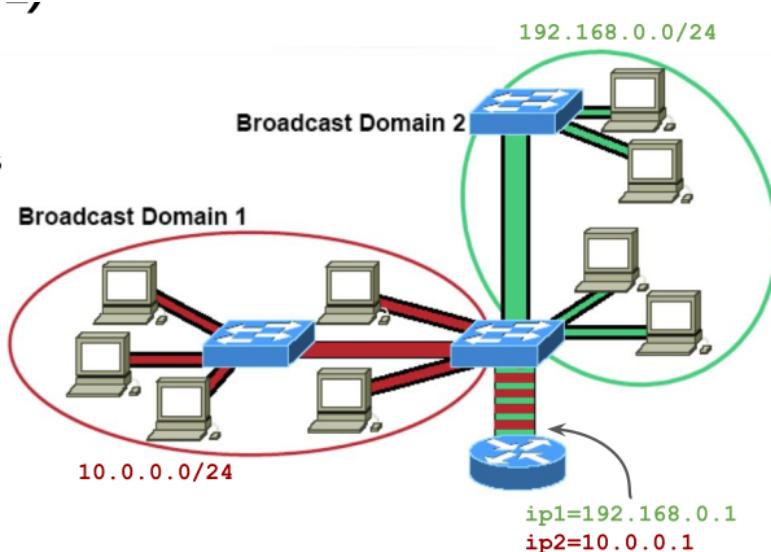


Figura 14: VLAN e sottoreti IP collegate tramite sub-interfacce

Listing 12: Esempio di configurazione router one-armed

```
ip link add link eth0 name eth0.10 type vlan id 10
ip link add link eth0 name eth0.20 type vlan id 20
ip addr add 10.0.10.1/24 dev eth0.10
ip addr add 10.0.20.1/24 dev eth0.20
```

In questo esempio, l’interfaccia fisica `eth0` viene suddivisa in due sub-interfacce virtuali, `eth0.10` e `eth0.20`, rispettivamente associate alle VLAN 10 e 20. A ciascuna sub-interfaccia viene assegnato un indirizzo IP appartenente alla sottorete della relativa VLAN.

In questo modo, il router può fornire servizi di **routing inter-VLAN**, agendo come **gateway predefinito** per ciascuna rete logica, pur utilizzando una sola porta fisica. Tale approccio è comune nei laboratori e negli ambienti di test, dove si vuole semplificare la topologia riducendo il numero di interfacce fisiche necessarie.

## 3.2 Trasporto dei frame: Tagging IEEE 802.1Q

Lo standard **IEEE 802.1Q** permette di far convivere più VLAN sullo stesso collegamento fisico. Per farlo, inserisce all’interno dei frame Ethernet un piccolo campo aggiuntivo, chiamato *tag VLAN*, che identifica la VLAN di appartenenza del frame. A seconda del tipo di collegamento, gli switch 802.1Q gestiscono questi tag in modo diverso, distinguendo tre tipologie di porte.

### 3.2.1 Porte di tipo Access, Trunk e Hybrid

- **Access Port:** collega un host finale (ad esempio un PC o una stampante). I frame che transitano su una porta di questo tipo sono sempre *senza tag (untagged)*. Lo switch associa internamente la porta a una specifica VLAN, aggiungendo o rimuovendo automaticamente il tag durante il transito interno.
- **Trunk Port:** collega due apparati di rete (ad esempio switch–switch o switch–router). Trasporta frame appartenenti a più VLAN contemporaneamente, inviandoli e ricevendoli *con tag 802.1Q*. In questo modo, ciascun frame mantiene l’informazione sulla VLAN di origine anche attraverso un link condiviso.
- **Hybrid Port:** gestisce sia frame *taggati* che *non taggati*. I frame non taggati vengono automaticamente associati alla **Native VLAN**, mentre quelli taggati mantengono il proprio VLAN ID. Questo tipo di porta è utile quando si collegano dispositivi che supportano il tagging VLAN insieme ad altri che non lo supportano.

### 3.2.2 Access links

Un **Access link** è un collegamento che parte da una **porta di tipo Access**, ossia una porta configurata per appartenere a una singola VLAN. Questo tipo di collegamento è utilizzato per connettere dispositivi finali (come PC, stampanti o server) oppure piccoli hub o switch non gestiti.

I frame che transitano su una porta di tipo Access sono sempre *non taggati (untagged)*: gli host collegati inviano e ricevono normali frame Ethernet, senza alcuna informazione sulla VLAN. È lo switch che, internamente, associa la porta a una VLAN e gestisce l’inserimento o la rimozione del *tag 802.1Q* quando il frame entra o esce dalla rete VLAN-aware.

In questo modo, i dispositivi connessi non devono essere consapevoli dell’esistenza delle VLAN: dal loro punto di vista fanno parte semplicemente di una rete Ethernet dedicata, tipicamente corrispondente a una specifica sottorete IP.

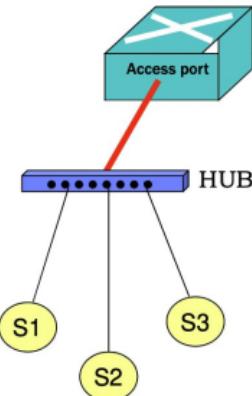


Figura 15: Esempio di collegamento Access link tra host e switch VLAN-aware

### 3.2.3 Access links nelle regioni legacy

In alcuni contesti, detti **legacy regions**, un access link può estendersi attraverso piccole LAN composte da più switch che non supportano le VLAN (*VLAN-unaware switches*). In questi casi, l'intera rete tradizionale viene vista dallo switch VLAN-aware come un unico segmento Ethernet appartenente a una sola VLAN.

Tutti i dispositivi collegati all'interno di tale regione condividono la stessa VLAN, anche se gli switch intermedi non gestiscono i tag 802.1Q. Questo approccio consente di integrare reti esistenti non VLAN-aware in un'infrastruttura moderna basata su VLAN.

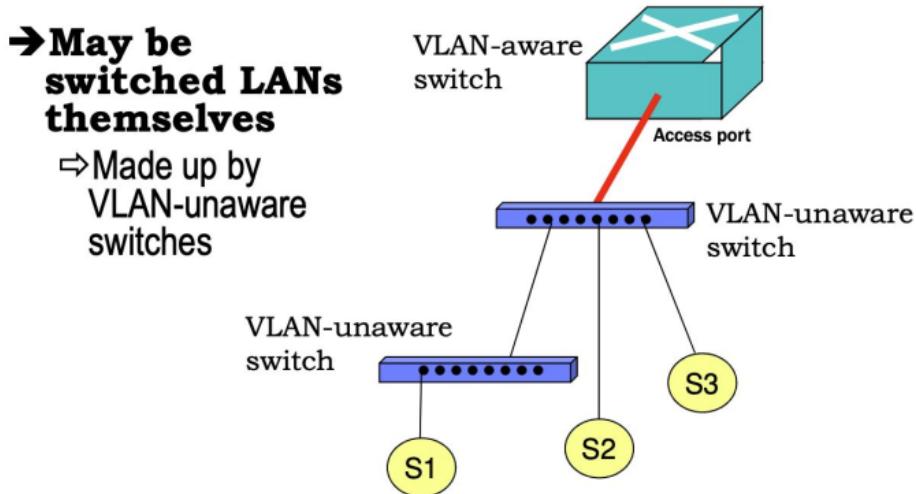


Figura 16: Access link esteso in una regione legacy con switch non VLAN-aware

### 3.2.4 Trunk links

Un **Trunk link** è un collegamento che parte da una **porta di tipo Trunk**, utilizzato per trasportare frame appartenenti a più VLAN contemporaneamente. È tipicamente impiegato nei collegamenti *switch–switch* o *switch–router*, dove è necessario far transitare traffico di più reti logiche sullo stesso mezzo fisico.

A differenza delle porte Access, le porte Trunk trasmettono e ricevono **frame taggati** con l'identificatore VLAN secondo lo standard **IEEE 802.1Q**. Il tag 802.1Q consente di distinguere

a quale VLAN appartiene ciascun frame, evitando la confusione tra traffici di reti diverse che condividono il link.

Un trunk link **non appartiene direttamente a una VLAN**, ma può trasportare:

- frame provenienti da *tutte* le VLAN configurate sullo switch;
- oppure frame appartenenti solo a un sottoinsieme di VLAN selezionate.

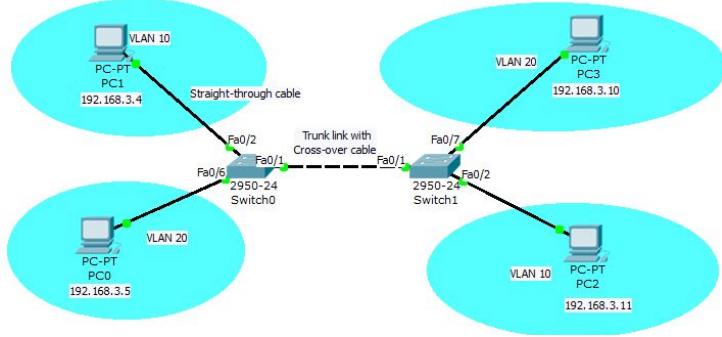


Figura 17: Esempio di collegamento Trunk tra apparati VLAN-aware

### 3.2.5 Hybrid links

Le **Hybrid links** rappresentano un'evoluzione dei trunk link e supportano sia **frame taggati** sia **frame non taggati**. I frame taggati mantengono il proprio VLAN ID, mentre i frame non taggati vengono associati a una VLAN predefinita (la **Native VLAN**).

Questo tipo di collegamento è utile quando sullo stesso link devono transitare:

- traffici di apparati VLAN-aware (che utilizzano frame taggati);
- e traffici di dispositivi legacy o VLAN-unaware (che inviano frame non taggati).

In sostanza, un hybrid link permette di far convivere traffico VLAN multiplo e traffico Ethernet standard sullo stesso collegamento, garantendo compatibilità tra apparati di diversa generazione. Nelle implementazioni moderne, molti switch trattano di fatto tutti i link come *ibridi*, in grado di gestire dinamicamente entrambe le tipologie di frame.

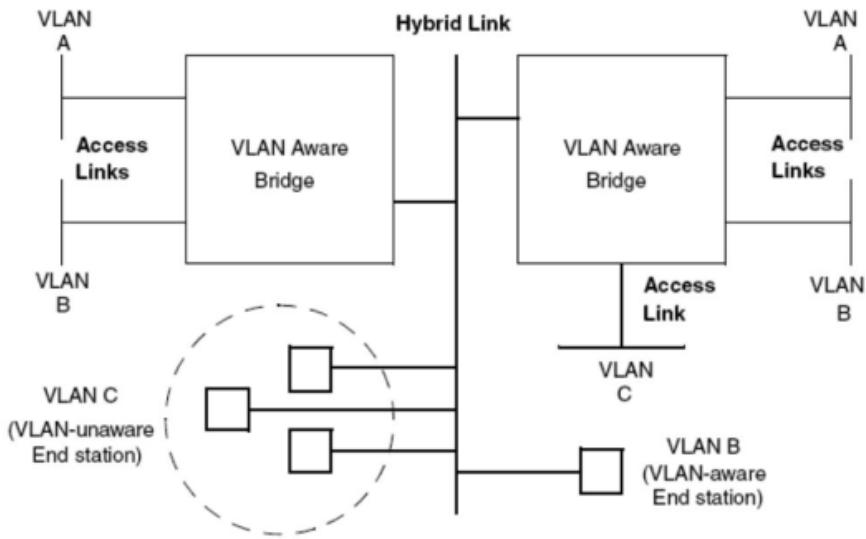


Figura 18: Esempio di Hybrid link che trasporta frame taggati e non taggati

In figura si nota come il collegamento ibrido consenta il transito simultaneo di traffico proveniente da VLAN diverse, includendo anche host non VLAN-aware (VLAN C), senza compromettere la separazione logica delle altre VLAN taggata.

### 3.2.6 Una stazione può appartenere a più VLAN?

In generale, un host connesso tramite una **Access port** appartiene a una sola VLAN, poiché i frame che transitano su quella porta sono sempre *non taggati* e associati a una singola rete logica.

Tuttavia, è possibile che una stessa stazione appartenga a **più VLAN** contemporaneamente. Questo avviene quando la stazione dispone di una **interfaccia trunk**, in grado di inviare e ricevere *frame taggati 802.1Q* appartenenti a VLAN diverse.

Un caso tipico è quello dei **server multi-VLAN**, che devono comunicare con più reti logiche (o sottoreti IP) attraverso un'unica interfaccia fisica. In tali scenari, il sistema operativo del server crea **sub-interfacce virtuali** (es. eth0.10, eth0.20), ciascuna configurata con un VLAN ID differente, consentendo la separazione logica del traffico pur utilizzando la stessa scheda di rete.

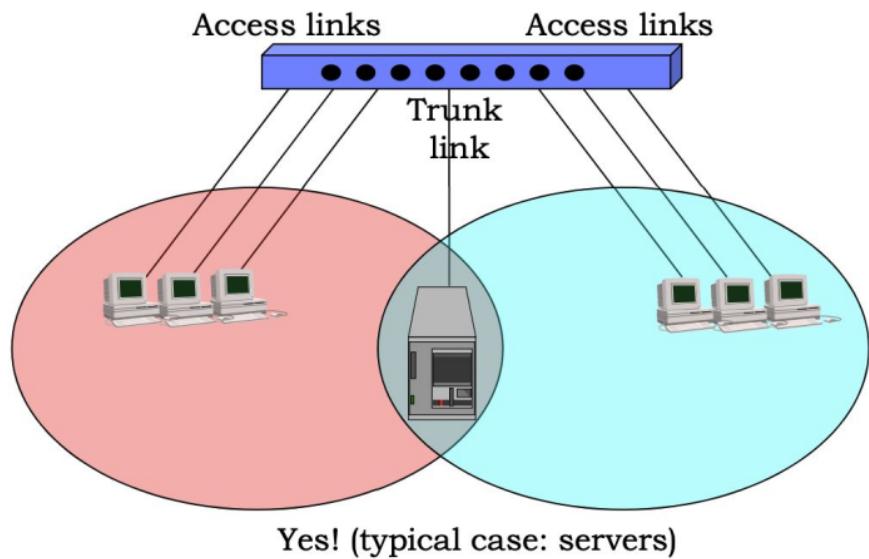


Figura 19: Esempio di stazione connessa a più VLAN tramite interfaccia trunk

In sintesi, solo le stazioni dotate di interfacce *VLAN-aware* possono appartenere a più VLAN: sono esse a gestire il tagging e l'instradamento del traffico tra le diverse reti logiche.

# Laboratorio

## 3.3 Laboratorio 3: Configurazione VLAN e router one-armed

In questo laboratorio si analizza il funzionamento delle VLAN e del **routing inter-VLAN** attraverso un router connesso tramite una singola interfaccia fisica (*one-armed router*). L'obiettivo è comprendere come i frame vengano trasportati attraverso collegamenti di tipo *Access*, *Trunk* e *Hybrid* secondo lo standard IEEE 802.1Q.

### Topologia di rete

La rete è composta da:

- uno **switch VLAN-aware** con tre porte configurate come Access (VLAN 10 e 20) e una porta configurata come Trunk verso il router;
- due host collegati alle porte Access, ciascuno appartenente a una VLAN distinta;
- un router configurato con sub-interfacce virtuali (eth0.10, eth0.20) per fornire connettività inter-VLAN.

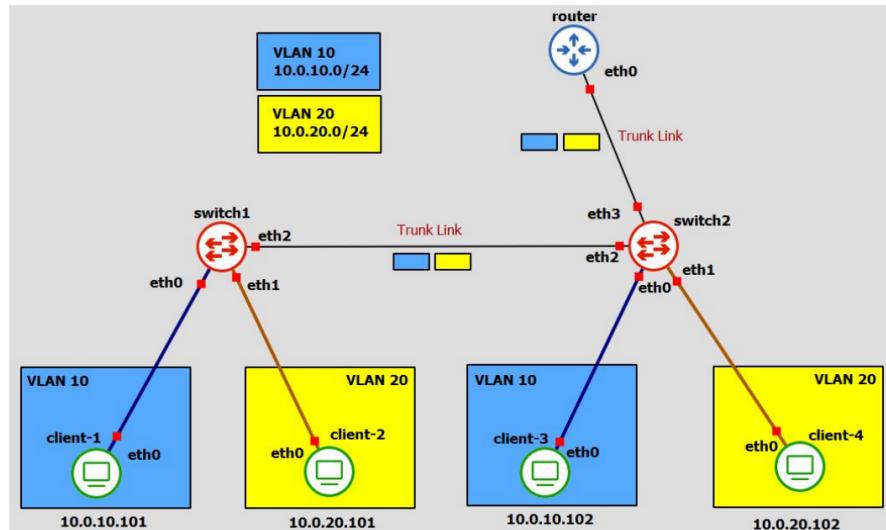


Figura 20: Topologia del Laboratorio 3: VLAN e router one-armed

### Configurazione di esempio

Listing 13: Configurazione delle VLAN sul router one-armed

```
# Creazione delle sub-interfacce VLAN
ip link add link eth0 name eth0.10 type vlan id 10
ip link add link eth0 name eth0.20 type vlan id 20

# Assegnazione degli indirizzi IP (gateway delle rispettive VLAN)
ip addr add 10.0.10.1/24 dev eth0.10
ip addr add 10.0.20.1/24 dev eth0.20

# Attivazione delle interfacce
ip link set eth0.10 up
ip link set eth0.20 up
```

Sul lato switch:

Listing 14: Configurazione delle VLAN sullo switch

```
# Creazione VLAN
vlan 10
vlan 20

# Assegnazione delle porte
interface eth1
switchport mode access
switchport access vlan 10

interface eth2
switchport mode access
switchport access vlan 20

# Porta trunk verso il router
interface eth0
switchport mode trunk
switchport trunk allowed vlan 10,20
```

## Funzionamento del laboratorio

Il laboratorio ha lo scopo di mostrare in modo pratico il funzionamento delle **VLAN** e del **router one-armed**, ovvero una configurazione in cui un unico collegamento fisico tra router e switch trasporta il traffico di più VLAN tramite **frame taggati IEEE 802.1Q**.

**3.3.0.1 Isolamento del traffico** Gli host collegati alle **porte Access** appartengono ciascuno a una VLAN distinta. I frame che transitano su queste porte sono *non taggati* e vengono separati dallo switch in base alla VLAN di appartenenza. In questo modo, gli host di VLAN diverse non possono comunicare direttamente: lo switch isola i domini di broadcast e impedisce la comunicazione a livello 2.

**3.3.0.2 Routing inter-VLAN** Per permettere la comunicazione tra VLAN diverse, il traffico deve passare attraverso il router. La connessione tra router e switch avviene tramite una **porta di tipo Trunk**, che trasporta i frame di più VLAN aggiungendo un tag 802.1Q a ciascun frame. Sul router, l'interfaccia fisica (ad esempio `eth0`) è suddivisa in più **sub-interfacce virtuali** (`eth0.10`, `eth0.20`, ecc.), ognuna configurata con:

- un **VLAN ID** specifico;
- un indirizzo IP che funge da **gateway** per la relativa VLAN.

Quando un host della VLAN 10 invia un pacchetto verso un host della VLAN 20:

1. il frame raggiunge lo switch sulla porta Access e viene inoltrato sul trunk verso il router con tag VLAN 10;
2. il router riceve il frame su `eth0.10`, lo elabora a livello 3 e decide di inoltrarlo sulla sub-interfaccia `eth0.20`;
3. il router rimanda il frame allo switch, questa volta con tag VLAN 20;
4. lo switch rimuove il tag e lo invia alla porta Access corrispondente alla VLAN 20.

## **Communicating between VLANs? Only via R1!!!**

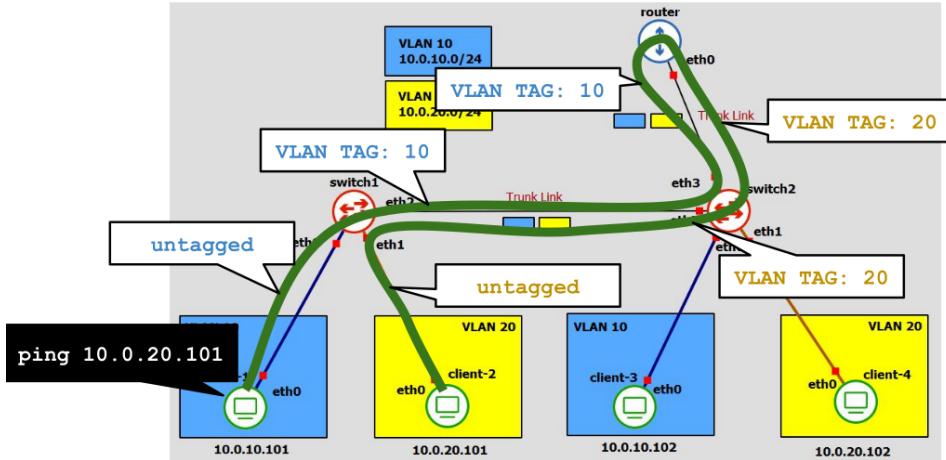


Figura 21: Comunicazione tra diverse vlan

**3.3.0.3 Verifica del comportamento** Nel laboratorio si può verificare che:

- gli host della stessa VLAN comunicano direttamente a livello 2, senza passare dal router;
- la comunicazione tra VLAN diverse avviene tramite il router (routing inter-VLAN);
- tutto il traffico tra router e switch è trasportato sul link trunk mediante frame taggati 802.1Q.

**3.3.0.4 Osservazione pratica** Catturando il traffico con `tcpdump` o `Wireshark` sull'interfaccia trunk, è possibile osservare i **tag VLAN** all'interno dei frame Ethernet. Ciò consente di verificare visivamente il meccanismo di separazione e instradamento del traffico tra le diverse VLAN.

### 3.4 Sicurezza delle VLAN e vulnerabilità di livello 2

Le VLAN migliorano l'isolamento logico del traffico, ma non eliminano le minacce presenti a livello di collegamento. Un attaccante connesso alla rete locale può sfruttare debolezze dei protocolli di livello 2 (Ethernet e ARP) o configurazioni errate degli switch per intercettare, modificare o dirottare il traffico di rete.

#### 3.4.1 Minacce principali

**3.4.1.1 1) MAC Flooding (CAM Overflow)** Gli switch mantengono in memoria una **Content Addressable Memory (CAM)**, che associa indirizzi MAC a porte fisiche. Un attaccante può inviare migliaia di frame con indirizzi MAC falsi, riempiendo la tabella CAM e provocando un *overflow*. Quando la tabella è satura, lo switch non riesce più a determinare su quale porta si trova un determinato MAC e inizia a inoltrare i frame in **broadcast**, esponendo il traffico all'attaccante.

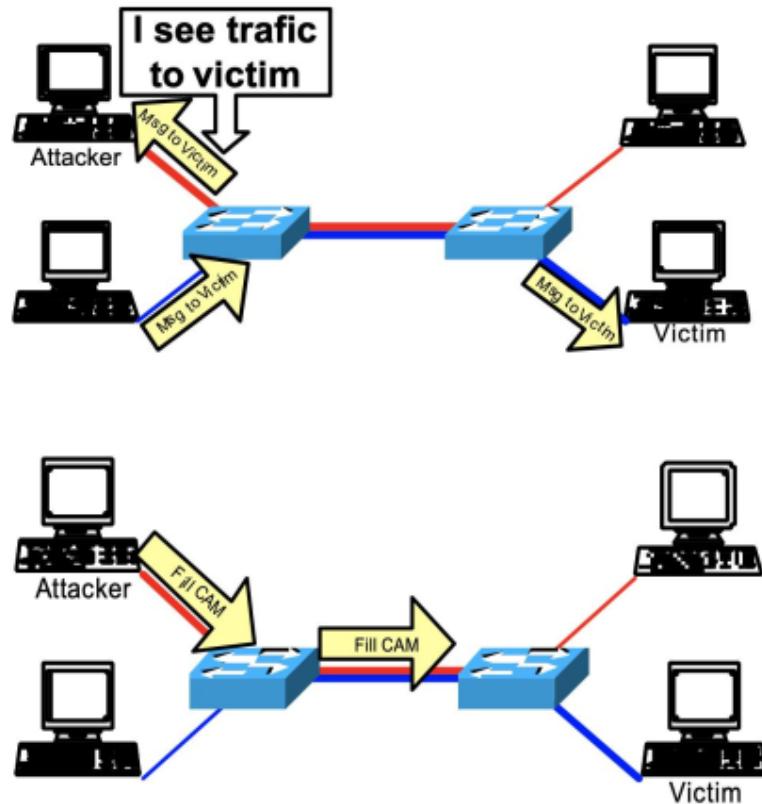


Figura 22: Esempio di MAC flooding

*Mitigazione:* abilitare la **port security**, limitando il numero di MAC address appresi per ciascuna porta, e disattivare l'apprendimento dinamico dove non necessario.

**3.4.1.2 2) ARP Spoofing / Poisoning** Il protocollo ARP non prevede autenticazione, quindi un attaccante può inviare **risposte ARP falsificate** per associare il proprio MAC all'indirizzo IP del gateway o di altri host nella stessa VLAN. In questo modo, intercetta o altera il traffico tra due dispositivi (*Man-in-the-Middle*).

*Mitigazione:* configurare **ARP statici** per i dispositivi critici, utilizzare strumenti di monitoraggio come `arpwatch` o implementare sistemi di protezione come **Dynamic ARP Inspection (DAI)** sugli switch gestiti.

**3.4.1.3 3) VLAN Hopping** Questa categoria di attacchi consente a un host di inviare o ricevere traffico appartenente a una VLAN diversa da quella assegnata, violando l'isolamento logico. Le due tecniche principali sono:

- **Basic VLAN hopping:** l'attaccante sfrutta il protocollo DTP (*Dynamic Trunking Protocol*) per negoziare automaticamente una connessione di tipo trunk con lo switch, ottenendo accesso a più VLAN.

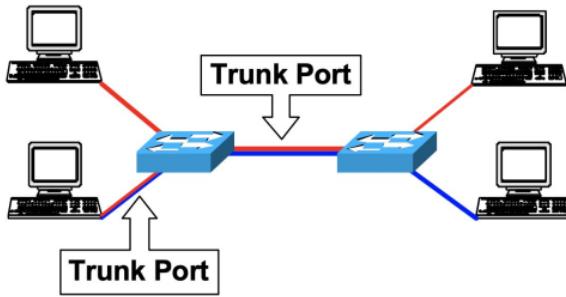


Figura 23: Collegamento tra due switch tramite porte trunk che trasportano il traffico di più VLAN sullo stesso link fisico.

- **Double Tagging:** il frame viene costruito con due tag 802.1Q annidati. Il primo tag (relativo alla VLAN nativa) viene rimosso dallo switch di ingresso, lasciando il secondo, che identifica la VLAN bersaglio.

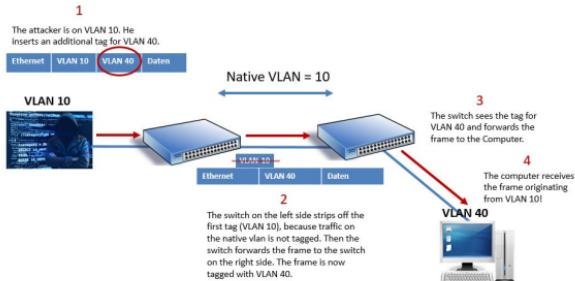


Figura 24: Esempio di attacco **VLAN Double Tagging**: un host malevolo appartenente alla VLAN 10 inserisce due tag 802.1Q (VLAN 10 e VLAN 40). Il primo switch rimuove il tag della VLAN nativa (10) e inoltra il frame, che conserva il secondo tag (40). Il frame attraversa quindi il trunk ed entra nella VLAN 40, violando l'isolamento tra VLAN.

*Mitigazione:* disabilitare DTP e configurare manualmente le porte come **Access** dove necessario; evitare l'uso della **VLAN 1** come VLAN nativa e assegnare VLAN native dedicate non utilizzate per il traffico utente.

**3.4.1.4 4) Attacchi ai protocolli di controllo** Oltre agli attacchi diretti alle VLAN, anche i protocolli di **controllo e gestione** utilizzati dagli switch di livello 2 possono essere sfruttati

da un attaccante per alterare la topologia della rete o raccogliere informazioni sensibili. Tra i principali:

#### 3.4.1.4.1 Spanning Tree Attack (BPDU spoofing)

- **Cosa fa lo STP:** gli switch si scambiano BPDU (bridge protocol data units) per eleggere il *root bridge* e stabilire quali porte siano forwarding o blocking, evitando loop.
- **Come attacca l'avversario:** l'attaccante invia BPDU falsi con una priorità molto bassa (o con un bridge ID fittizio), facendo credere agli switch che il suo dispositivo sia il nuovo root.
- **Effetto pratico:** la topologia si ricalcola; alcuni link possono essere forzati in forwarding creando percorsi non previsti, perdita di connettività o instradamento del traffico attraverso il nodo dell'attaccante.
- **Mitigazione:** *BPDU Guard* spegne la porta se riceve BPDU su una porta che dovrebbe essere una porta access (cioè verso host), mentre *Root Guard* impedisce a un certo segmento di diventare root forzando il comportamento previsto.

#### 3.4.1.4.2 VTP Attack (VLAN Trunking Protocol)

- **Cosa fa VTP:** permette di distribuire automaticamente la lista delle VLAN a tutti gli switch del dominio VTP.
- **Come attacca l'avversario:** un dispositivo malevolo si presenta come **server VTP** con una *revision number* superiore; gli altri switch accettano la nuova configurazione e sovrascrivono le VLAN locali.
- **Effetto pratico:** le VLAN possono essere cancellate o modificate su larga scala, causando indisponibilità o perdita dell'isolamento tra reti.
- **Mitigazione:** impostando VTP in *transparent* o disabilitandolo si evita che uno switch accetti e propaghi automaticamente configurazioni provenienti da fonti non affidabili.

#### 3.4.1.4.3 Cisco Discovery Protocol Attack (information leakage)

- **Cosa fa CDP:** i dispositivi Cisco pubblicano informazioni (IP, modelli, versioni) su CDP verso i vicini.
- **Come attacca l'avversario:** un host malintenzionato cattura i pacchetti CDP o ascolta il traffico e ottiene dettagli utili per attacchi mirati (es. versioni vulnerabili).
- **Effetto pratico:** ricognizione facilitata: l'attaccante conosce quali dispositivi e software colpire.
- **Mitigazione:** disabilitando CDP sulle porte utenti si riduce la quantità di informazioni esposte ai terminali non fidati.

# Laboratorio

## 3.5 Laboratorio 4: VLAN Hopping e Double Tagging

### 3.5.1 Scenario

L'obiettivo è dimostrare un attacco di **double tagging 802.1Q**: un host nella **VLAN nativa (VLAN 1)** tenta di inviare traffico a una vittima in **VLAN 20** *senza* routing inter-VLAN, sfruttando la rimozione del tag della VLAN nativa sul primo switch.

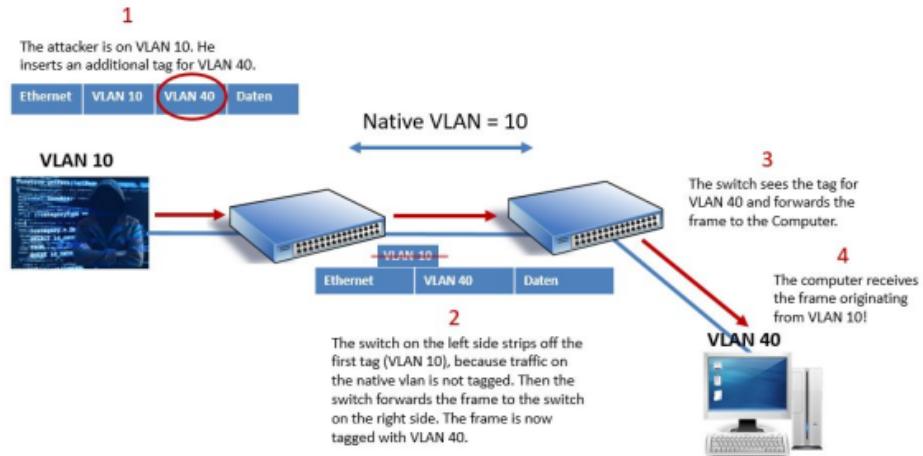


Figura 25: Attacco Double Tagging su trunk con VLAN nativa impostata a 1.

### 3.5.2 Topologia del laboratorio

La topologia è in Figura 26: attaccante in **VLAN 1 (nativa)** connesso a una porta che insiste sul *trunk*; vittima in **VLAN 20** su uno switch a valle; nessun router tra le VLAN.

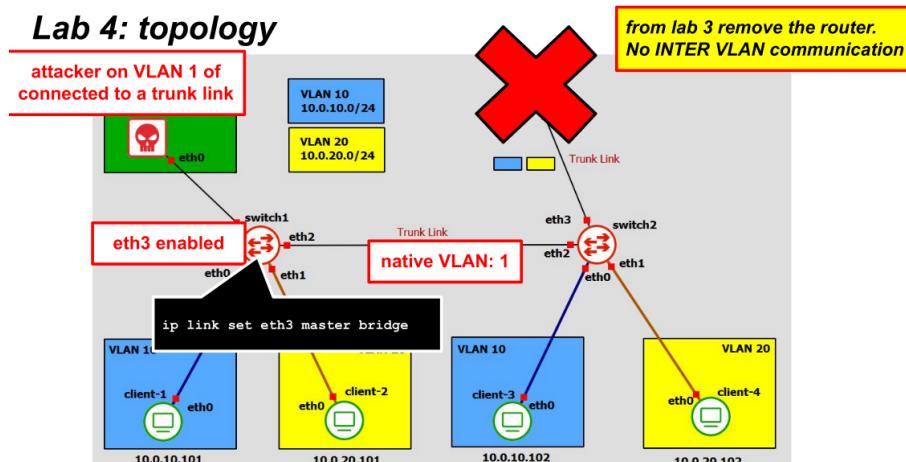


Figura 26: Lab 4 — Topologia: attaccante su VLAN 1 (nativa) verso trunk; vittima in VLAN 20; no inter-VLAN routing.

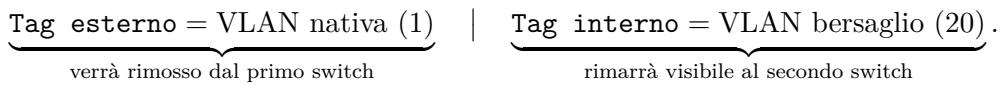
*Nota operativa.* Le porte `eth3` dei due switch formano il *trunk* con **VLAN nativa = 1**. L'attaccante è su una porta *access* in VLAN 1; gli host `client-3` (VLAN 10) e `client-4` (VLAN 20) sono su porte *access* degli switch opposti.

### 3.5.3 Prerequisiti perché l'attacco riesca

- esiste un **link trunk** tra due switch;
- stessa **VLAN nativa** configurata su entrambe le estremità del trunk (qui: VLAN 1);
- l'attaccante è connesso a **VLAN 1** (nativa) e può inviare frame con *doppio tag*;
- la vittima appartiene a una **VLAN diversa** (qui: VLAN 20).

### 3.5.4 Funzionamento dell'attacco (passo-passo)

L'attaccante costruisce un frame con **due tag 802.1Q annidati**:



1. **Invio dal nodo malevolo.** Il frame parte dall'attaccante con due tag: VLAN 1 (esterno) e VLAN 20 (interno).
2. **Primo switch (ingresso trunk).** Per definizione, i frame *taggati con la VLAN nativa* sul trunk vengono trattati come *untagged*: lo switch **rimuove il tag esterno** (VLAN 1) e inoltra il frame sul trunk *senza quel tag*.
3. **Attraversamento del trunk.** Il frame prosegue e, non avendo più il tag esterno, **rimane solo il tag interno** (VLAN 20) nel payload.
4. **Secondo switch (uscita trunk).** Vede un frame *ancora taggato* con **VLAN 20** (il tag interno) e lo inoltra correttamente all'*access port* della VLAN 20.
5. **Consegna alla vittima.** L'host in VLAN 20 riceve il frame come se provenisse da un nodo legittimo della propria VLAN.

**Unidirezionalità.** La risposta della vittima *non* segue lo stesso percorso: l'host in VLAN 20 non inserisce doppio tag e il frame di risposta non potrà rientrare verso l'attaccante in VLAN 1 senza routing. Per questo l'attacco è tipicamente **monodirezionale** (utile per inviare pacchetti/sondare servizi, meno per instaurare dialoghi completi).

### 3.5.5 Esecuzione pratica (Linux)

Listing 15: Costruzione di un frame con doppio tag via sub-interfacce annidate

```
ip link add link eth0 name eth0.1 type vlan id 1
ip link set eth0.1 up
ip link add link eth0.1 name eth0.1.20 type vlan id 20
ip link set eth0.1.20 up
ip addr add 10.0.20.250/24 dev eth0.1.20
arp -s 10.0.20.102 <MAC-vittima> -i eth0.1.20
ping 10.0.20.102
```

In questo modo lo `stack` inserisce **due tag 802.1Q** (outer=1, inner=20). Il `ping` raggiunge la vittima, ma la risposta generalmente non torna all'attaccante (vedi unidirezionalità).

### 3.5.5.1 Cosa osservare con lo sniffer

- sul **primo switch lato trunk**: si vedono frame che escono *senza* tag nativo e *con* il tag interno (20) ancora presente;
- sul **secondo switch**: si vedono frame **con VLAN 20** che vengono consegnati alla porta **access** della vittima;
- sull'host attaccante: assenza di risposte ICMP (o ARP) dalla vittima per la natura monodirezionale.

### 3.5.5.2 Limitazioni pratiche

- se la **VLAN nativa non è usata** (o è diversa e dedicata), il meccanismo di rimozione del primo tag non produce l'effetto desiderato;
- molti switch moderni applicano **filtr** su frame anomali (ad es. doppio tag con *outer* uguale alla nativa).

### 3.5.5.3 Mitigazioni

- **Disabilitare l'auto-trunking (DTP)** e configurare *manualmente* le porte utente come **access**;
- **Non usare VLAN 1 come nativa**; scegliere una **VLAN nativa dedicata** e non operativa per il traffico utente;
- **Consentire sul trunk solo le VLAN necessarie** (*allowed VLAN list*);
- abilitare controlli di **storm/unknown-unicast** e ispezioni su frame con *double tag* dove supportato;
- monitoraggio con IDS/port mirroring su link trunk per **pattern di doppio tagging**.

## Conclusioni

Il **double tagging** sfrutta la gestione della **VLAN nativa** sui trunk per far accettare a uno switch un *tag interno* verso una VLAN diversa, *senza* routing. È spesso **monodirezionale**. La difesa passa da: hardening dei trunk (nativa dedicata, allowed-VLAN), porte utente in **access** statico, autenticazione 802.1X e controllo a livello di firewall/IDS.

## 4 NET\_04 — 802.1x

### 4.1 Quadro generale e obiettivo

IEEE 802.1X è lo standard per il *Port-based Network Access Control* (PNAC): l'accesso alla rete avviene “per porta” a livello 2, così che solo i dispositivi autenticati possano oltrepassare l'autenticatore (switch/AP). Lavorando a L2, 802.1X evita processamento IP durante l'onboarding: i frame di autenticazione e i dati applicativi passano su interfacce logiche differenti, riducendo costi e superficie d'attacco. 802.1X definisce l'incapsulamento di EAP su LAN (**EAPoL**). EAP, invece, è un *framework* IETF indipendente dallo standard 802.1X.

### 4.2 Attori dell'architettura (vista di alto livello)

Tre ruoli: **Supplicant** (host/utente che richiede accesso), **Authenticator** (switch/AP che filtra e media), **Authentication Server** (AAA, tipicamente RADIUS). In stato iniziale la porta è *unauthorized* e lascia passare soltanto EAPoL; a successo la porta diventa *authorized* e il traffico dati è ammesso secondo la policy decisa dal server.

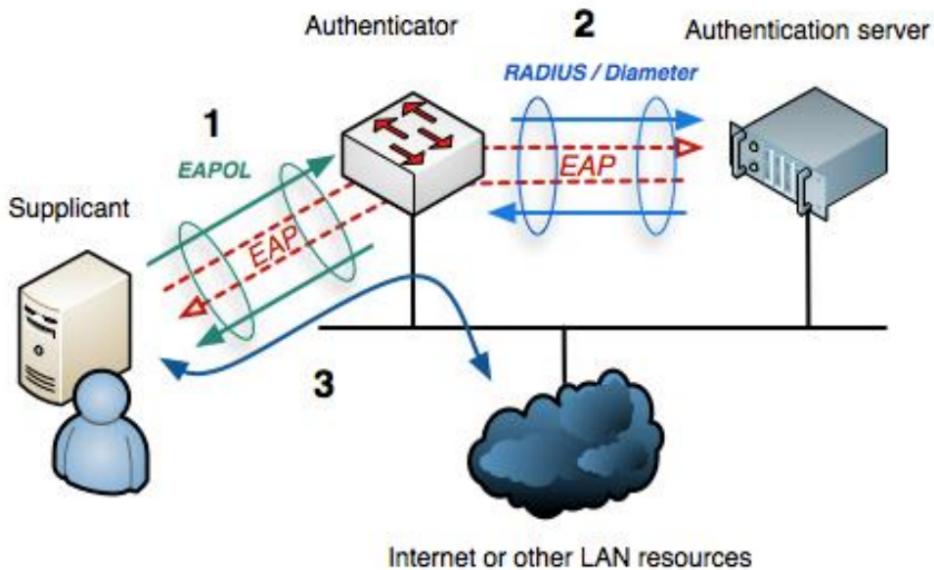


Figura 27: Architettura 802.1X e piani di traffico: (1) EAP incapsulato in EAPoL tra *supplicant* e *authenticator*; (2) EAP trasportato verso il server AAA tramite RADIUS/DIAMETER; (3) a autenticazione riuscita, sblocco del piano dati verso risorse LAN/Internet.

### 4.3 EAP ed EAPoL: fondamenti

**4.3.0.1 EAP (Extensible Authentication Protocol).** EAP è un *framework di autenticazione* standardizzato (RFC 3748, aggiornato da RFC 5247): definisce il *dialogo* tra peer (supplicant) e server e il formato dei messaggi, ma **non** impone un meccanismo crittografico unico. In altre parole, EAP stabilisce come negoziare ed eseguire un *metodo EAP* (la vera ricetta di autenticazione), mentre il trasporto su rete è demandato allo *strato sottostante*.

- **Che cos’è:** un livello di astrazione che fornisce messaggi *Request/Response* e *Success/Failure*, più un canale per scambiarsi i parametri del metodo scelto; esistono una quarantina di metodi (tra cui **EAP-TLS**, **PEAP**, **EAP-TTLS**, **EAP-SIM/AKA**, **EAP-FAST**).

- **Che cosa non è:** *non* è un *wire protocol* L2/L3; non cifra né autentica “da solo”; la sicurezza **dipende** dal metodo selezionato (es.: *EAP-MD5* non offre mutua autenticazione né protezione contro attacker attivi; *EAP-TLS* sì).
- **Dove gira:** EAP può essere incapsulato su vari “lower layer” (PPP, 802.11, 802.3); in 802.1X su Ethernet/Wi-Fi si usa **EAP over LAN (EAPoL)** per portare EAP tra supplicant e authenticator, che a sua volta inoltra verso AAA (tipicamente RADIUS).

*Formato dei messaggi:* un pacchetto EAP contiene **Code** (*Request, Response, Success, Failure*), **Identifier** (accoppia richiesta/risposta), **Length** e **Data** (il payload del metodo). Il server AAA decide l'esito del metodo e, se positivo, l'authenticator sblocca la porta e applica l'autorizzazione (VLAN/ACL).

#### 4.4 EAP termination vs EAP relay (EAPoR)

Tra *authenticator* (switch/AP) e *server AAA* esistono due modelli operativi, con effetti concreti su sicurezza, scalabilità e troubleshooting.

**4.4.0.1 EAP termination mode (terminazione locale).** Lo switch *implementa* il metodo EAP e parla con il server usando i messaggi RADIUS tradizionali (*Access-Request/Accept/Reject*) senza trasportare il payload EAP end-to-end.

- **Dove “vive” il metodo:** dentro lo switch (NAS). Il server AAA verifica credenziali/attributi, ma non esegue direttamente il metodo EAP.
- **Vantaggi:** latenza ridotta; minor banda verso AAA; utile per metodi semplici (es. EAP-MD5 in lab) o quando si desidera *offload* di logica sul bordo.
- **Limiti:** meno flessibilità (aggiornare/estendere metodi richiede supporto sul NAS); minore *visibilità* diagnostica lato server; la derivazione delle chiavi (MSK) non arriva automaticamente al server, quindi integrazioni con altre fasi (es. roaming/derivazioni) sono più macchinose.
- **Uso tipico:** ambienti didattici o reti con metodo basico e policy semplici.

**4.4.0.2 EAP relay mode (EAP over RADIUS, EAPoR).** Lo switch *inoltra inalterati* i messaggi EAP del supplicant al server incapsulandoli negli attributi RADIUS **EAP-Message + Message-Authenticator**. Il metodo EAP viene eseguito *end-to-end* tra supplicant e server.

- **Dove “vive” il metodo:** sul *server AAA*. Lo switch resta un ponte L2/AAA “consapevole”.
- **Vantaggi:** massima compatibilità con metodi complessi (EAP-TLS, PEAP/TTLS, FAST, SIM/AKA); la **MSK/EMSK** è generata sul server e può alimentare altri servizi (es. derivazioni chiave, MACsec con dynamic CAK, Wi-Fi 802.11i); aggiornare/aggiungere metodi richiede solo aggiornare il server.
- **Sicurezza:** l'attributo **Message-Authenticator** protegge l'integrità del trasporto EAP su RADIUS; con link non fidati si usa RADIUS/TLS (RadSec) o IPsec per confidenzialità.
- **Diagnostica:** il server vede *tutto* lo scambio EAP (tracce/PCAP più utili); più semplice fare *troubleshooting* su handshake TLS, certificati, inner method, ecc.
- **Uso tipico:** quasi tutti i deployment reali 802.1X enterprise.

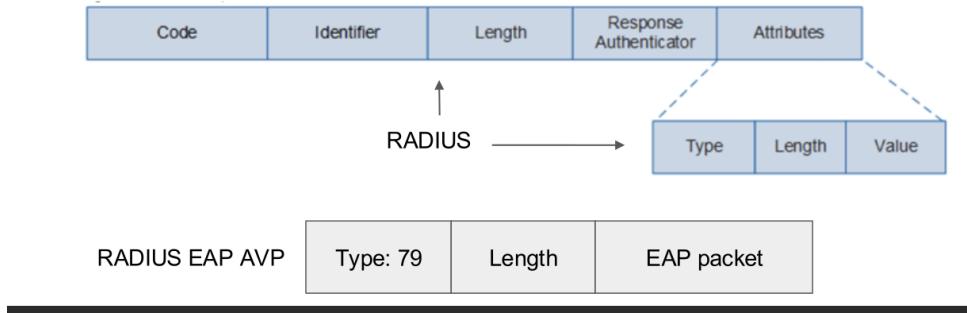


Figura 28: struttura pacchetto EAPoR

## 4.5 Processi di autenticazione: dai diagrammi alle implicazioni operative

**4.5.0.1 Message exchange (visione di alto livello).** Lo scambio coinvolge tre attori:

1. **Suplicant ↔ Authenticator (EAPoL):** il controllo d'accesso viaggia su *EAP over LAN*. Tipicamente: *EAP-Request/Identity* → *EAP-Response/Identity* e poi i messaggi del metodo EAP.
2. **Authenticator ↔ Server AAA (RADIUS/DIAMETER):** l'authenticator incapsula/-termina EAP e dialoga con il server usando messaggi *Access-Request/Challenge/Accept/Reject* (più accounting). In *relay* l'EAP del client è inoltrato end-to-end dentro l'attributo *EAP-Message*.
3. **Sblocco del piano dati:** a successo, l'authenticator cambia lo stato della porta in *authorized* e applica la policy (VLAN/ACL/UCL). In caso di fallimento o *EAPoL-Logoff*, la porta torna *unauthorized*.

*Implicazioni:* il troubleshooting si fa catturando EAPoL lato accesso (tcpdump ether proto 0x888e) e RADIUS lato control-plane; timer e ritrasmissioni sono gestiti dall'authenticator e impattano l'esperienza di login.

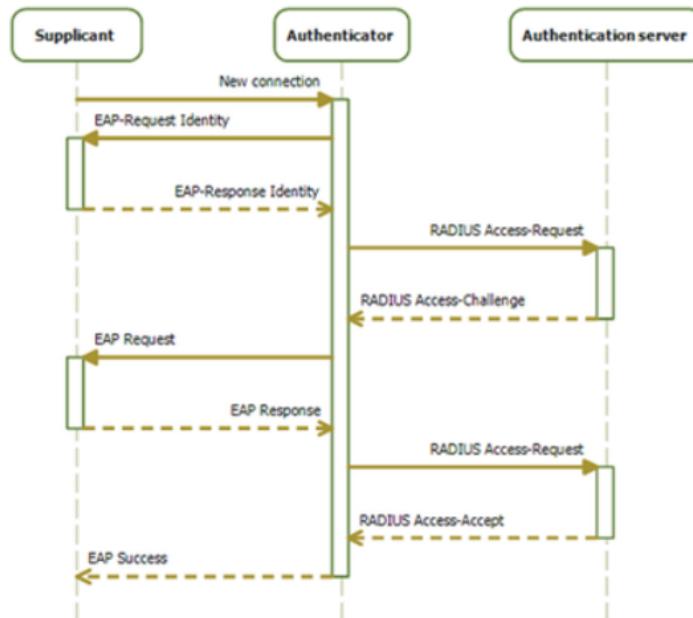


Figura 29: Message exchange ad alto livello

**4.5.0.2 Relay Mode (EAP-MD5) — sequenza dettagliata.** Nel *relay* l'authenticator non “capisce” il metodo: inoltra i messaggi EAP al server dentro RADIUS.

1. **EAPoL-Start** (client → authenticator): il supplicant innesca la procedura.
2. **EAP-Request/Identity** (authenticator → client): richiesta dell'identità.
3. **EAP-Response/Identity** (client → authenticator) ⇒ **Access-Request** con EAP-Message (authenticator → server).
4. **Access-Challenge** (server → authenticator): il server genera la *MD5 challenge*.
5. **EAP-Request/MD5-Challenge** (authenticator → client): inoltro della challenge.
6. **EAP-Response/MD5-Challenge** (client → authenticator): il client calcola  $MD5(id + challenge + password)$  e risponde ⇒ **Access-Request** con EAP-Message (authenticator → server).
7. **Access-Accept/Reject** (server → authenticator): esito dell'autenticazione (in *Accept* possono esserci attributi di autorizzazione: VLAN, ACL, UCL).
8. **EAP-Success/Failure** (authenticator → client) e **cambio stato porta**: *authorized* a successo, *unauthorized* a fallimento.
9. **Handshake/keepalive**: l'authenticator interroga periodicamente il client (es. *EAP-Request/Identity*) per verificare che la sessione sia ancora viva; timeout consecutivi ⇒ chiusura sessione.
10. **EAPoL-Logoff** (client → authenticator): logout proattivo; l'authenticator ferma l'accouting, rimuove autorizzazioni e riporta la porta a *unauthorized*.

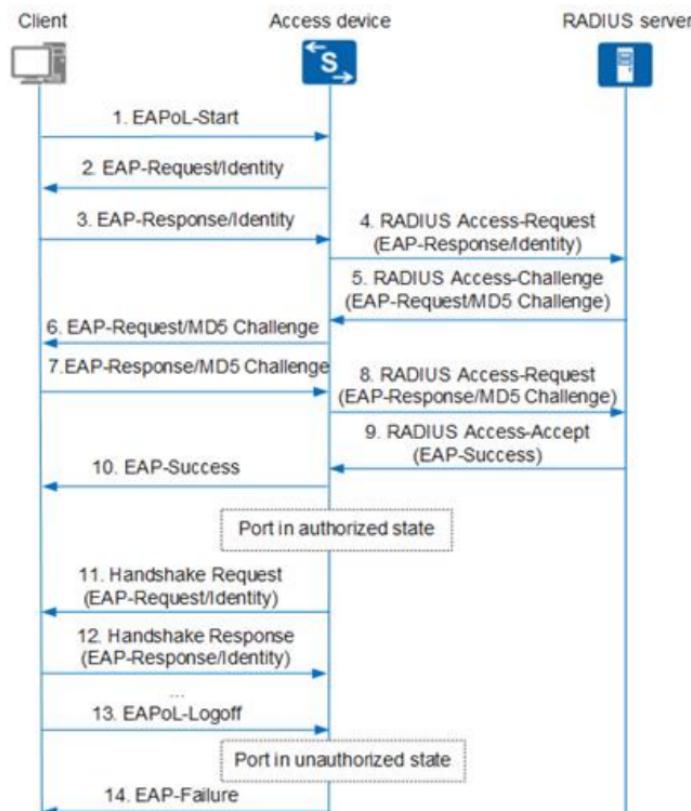


Figura 30: RelayMode md5 challenge

**4.5.0.3 EAP-TLS: esempio più complesso.** Rispetto a EAP-MD5, qui il processo è molto più sicuro perché si basa su un **handshake TLS** completo tra il client (supplicant) e il server AAA.

In pratica, dopo l'avvio (*EAPoL-Start*), l'autenticazione procede così:

- Il server invia una richiesta EAP di tipo **EAP-TLS**, che avvia la sessione TLS (come un normale *client\_hello/server\_hello* HTTPS).
- Il client risponde con il proprio **certificato** e completa la fase di *key exchange* e *cipher negotiation*.
- Entrambe le parti verificano i certificati: così si ottiene una **autenticazione reciproca (mutual authentication)**.
- Alla fine, viene derivata una **Master Session Key (MSK)** sicura, usata per proteggere il traffico o per altri protocolli (es. MACsec o WPA2-Enterprise).

Se qualcosa va storto (certificato non valido, handshake interrotto, errore TLS), il server invia *EAP-Failure*. In sintesi, EAP-TLS trasporta l'intero handshake TLS dentro EAP, fornendo un livello di sicurezza equivalente a una connessione HTTPS cifrata, ma applicato all'autenticazione di rete.

## 4.6 Operazioni aggiuntive: ri-autenticazione, logout, timer

Le slide enfatizzano aspetti gestionali spesso sottovalutati:

- **Re-auth:** se cambiano parametri (o stato) dell'utente, si forza ri-autenticazione; in condizioni anomale è previsto un flusso per utenti in *pre-connection* (ottimizza l'accesso “rapido”).
- **Logout e accounting:** se l'uscita non è rilevata (né dallo switch né da RADIUS), l'accounting resta “attivo”, creando incongruenze e possibili *spoof* su IP/MAC “appesi”. L'access device deve *immediatamente* rilevare logout, cancellare l'entry utente e fermare l'accounting.
- **Timer:** 802.1X si appoggia a timer per ritrasmissioni e timeout; la loro taratura impatta esperienza utente e affidabilità della sessione.

Tutti questi punti sono rappresentati esplicitamente nelle slide “Re-Authentication”, “Log out and Timers”.

## 4.7 Autorizzazione post-autenticazione: VLAN, ACL, UCL

**4.7.0.1 VLAN dinamiche.** Utenti non autenticati e risorse “ristrette” sono messi in VLAN diverse; a successo, il server assegna una *authorized VLAN*, che ha precedenza sulla configurazione statica dell'interfaccia. Attributi RADIUS standard obbligatori: *Tunnel-Type=VLAN(13)*, *Tunnel-Medium-Type=802(6)*, *Tunnel-Private-Group-ID=<VLAN>*. La VLAN statica torna attiva quando l'utente va offline.

**4.7.0.2 ACL per-utente.** Dopo l'autenticazione, il server può inviare all'*authenticator* una **Access Control List (ACL)** personalizzata per filtrare il traffico dell'utente. L'ACL definisce quali pacchetti possono attraversare la porta: i pacchetti che corrispondono a regole *permit* vengono inoltrati, mentre quelli che incontrano regole *deny* vengono bloccati.

Le ACL possono essere assegnate in due modi:

- **Assegnazione statica:** il server invia solo un riferimento all'ACL tramite l'attributo **Filter-Id**. Le regole vere e proprie devono essere già configurate localmente sull'apparato di accesso.
- **Assegnazione dinamica:** il server invia insieme al nome anche le *regole complete*, tramite attributi estesi RADIUS (ad esempio **HW-Data-Filter** nei sistemi Huawei). In questo modo le policy possono essere generate o aggiornate in tempo reale, senza modificare la configurazione dello switch.

Questa funzione consente di applicare controlli di traffico differenti per utente, ruolo o gruppo, rendendo la fase di *autorizzazione* molto più granulare e adattabile.

**4.7.0.3 UCL (User Control List).** La **User Control List (UCL)** è un meccanismo che permette di gestire in modo collettivo gruppi di utenti o dispositivi che condividono le stesse esigenze di accesso. Invece di assegnare policy individuali a ogni terminale, gli utenti vengono *raggruppati* in una UCL che eredita un insieme comune di regole.

Durante l'autenticazione, il server può:

- assegnare la UCL tramite il suo **nome**, usando l'attributo standard **Filter-Id**;
- oppure indicarla tramite un **identificatore numerico (ID)**, usando attributi RADIUS estesi, come **HW-UCL-Group** nei sistemi Huawei.

In entrambi i casi, la policy associata alla UCL deve essere *preconfigurata* sull'apparato di accesso, così che l'authenticator sappia quali permessi e restrizioni applicare agli utenti appartenenti a quel gruppo. Questo approccio semplifica notevolmente la gestione amministrativa, riducendo errori e duplicazioni nella configurazione delle policy di rete.

## 4.8 Vulnerabilità storiche e motivazione per MACsec

Se un attaccante intercetta i frame su una porta autorizzata (es. stesso hub del legittimo), può *spoofare* MAC/IP e accedere al mezzo; inoltre gli *EAPoL-Logoff* essendo in chiaro sono falsificabili per causare DoS. Per mitigare questi limiti la revisione 802.1X-2010 introduce **MACsec Key Agreement (MKA)** e, con MACsec, confidenzialità/integrità/anti-replay dei frame L2.

# Laboratorio

## 4.9 Lab 5 — 802.1X Port-Based Authentication & VLAN assignment

**4.9.0.1 Obiettivo.** In questo laboratorio viene dimostrato il funzionamento del controllo d'accesso 802.1X su porta con autenticazione basata su **EAP-MD5** e l'assegnazione dinamica di VLAN da parte del server RADIUS. L'esperimento mostra come solo gli host autenticati possano generare traffico sulla rete e come l'autenticazione permetta al server di determinare la VLAN o le ACL applicate alla porta di accesso.

**4.9.0.2 Topologia di rete.** L'ambiente di test comprende:

- due host cablati che agiscono da *supplicant*;
- un server RADIUS (FreeRADIUS) che gestisce AAA e assegna VLAN dinamiche;
- uno switch Linux che funge da *authenticator* 802.1X e da router L3;
- un access point o gateway che fornisce l'accesso a Internet.

Le interfacce **eth0** e **eth1** dello switch rappresentano le porte di accesso controllate da 802.1X; **eth2** è la porta verso il server RADIUS; **eth3** collega la rete interna e fornisce connettività verso Internet.

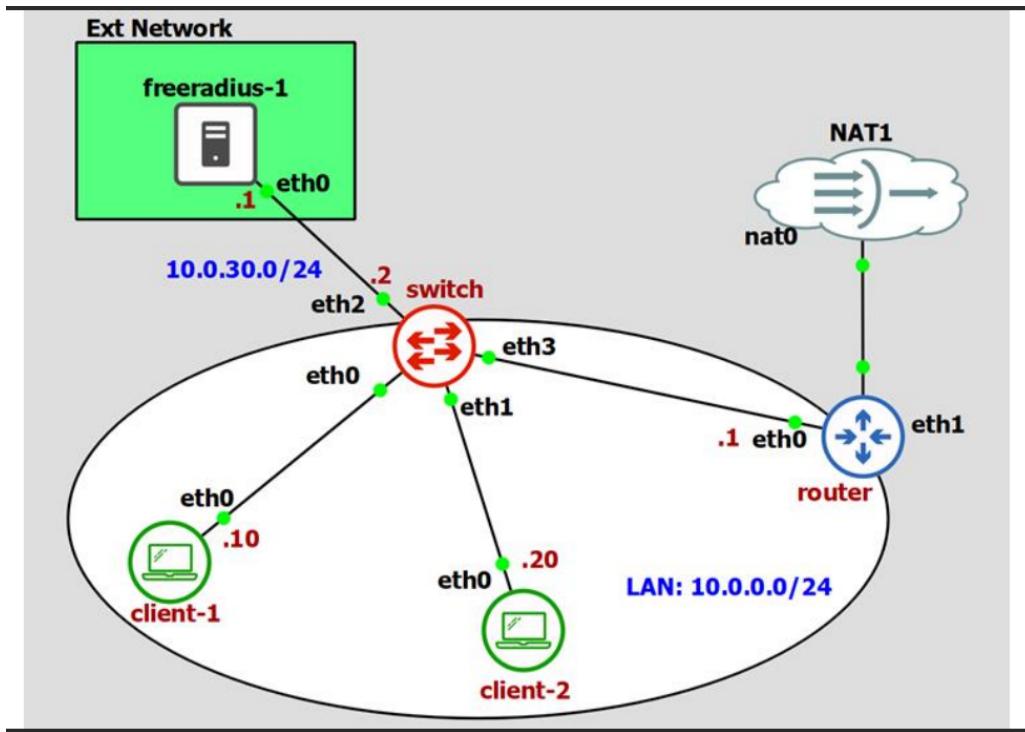


Figura 31: Topologia rete

**4.9.0.3 Configurazione dello switch (Authenticator).** Lo switch agisce da intermediario 802.1X grazie a `hostapd` in modalità `driver=wired`. Le interfacce di accesso sono inizialmente bloccate (*unauthorized*) e vengono sbloccate solo dopo l'autenticazione positiva. Le operazioni principali:

- creazione del `bridge` e associazione delle interfacce `eth0`, `eth1` e `eth3`;

- abilitazione dell'inoltro EAPoL con: `echo 8 > /sys/class/net/bridge/bridge/group_fwd_mask;`
- impostazione di policy DROP su `ebtables` per il traffico L2 non autorizzato;
- configurazione `hostapd.conf` con: `ieee8021x=1, use_pae_group_addr=1, auth_server_addr, auth_server_port, auth_server_shared_secret.`

**4.9.0.4 Server di autenticazione (FreeRADIUS).** Il server riceve le richieste RADIUS dagli *authenticator* e valida le credenziali degli utenti. La configurazione include:

- file `/etc/freeradius/3.0/clients.conf`: definisce i NAS autorizzati (es. lo switch Linux);
- file `/etc/freeradius/3.0/users`: definisce gli utenti e le VLAN da assegnare, ad esempio:

```
pippo Cleartext-Password := "pippo"
  Service-Type = Framed-User,
  Tunnel-Type = 13,
  Tunnel-Medium-Type = 6,
  Tunnel-Private-Group-ID = 10
```

```
pluto Cleartext-Password := "pluto"
  Service-Type = Framed-User,
  Tunnel-Type = 13,
  Tunnel-Medium-Type = 6,
  Tunnel-Private-Group-ID = 20
```

- avvio del demone con `freeradius -X` per visualizzare in tempo reale i pacchetti RADIUS.

**4.9.0.5 Configurazione dei client (Supplicant).** Ogni host utilizza `wpa_supplicant` in modalità wired con metodo **EAP-MD5**. Esempio di configurazione:

```
ap_scan=0
network={
  key_mgmt=IEEE8021X
  eap=MD5
  identity="pippo"
  password="pippo"
  eapol_flags=0
}
```

Gli host sono inizialmente configurati con IP statico per semplicità, ma in un contesto reale si può integrare un server DHCP che rilasci gli indirizzi dopo l'autenticazione.

**4.9.0.6 Controllo del traffico L2.** Per abilitare o bloccare dinamicamente le stazioni autorizzate, viene utilizzato uno **script Python** collegato a `hostapd_cli`. Lo script intercetta gli eventi *EAP-SUCCESS* ed *EAP-FAILURE* e modifica le regole di `ebtables`:

- aggiunge una regola ACCEPT per il MAC dell'utente autenticato;
- rimuove la regola quando l'utente si disconnette (**EAPoL-Logoff**) o fallisce la ri-autenticazione.

Le associazioni vengono salvate in un file JSON per garantire persistenza.

#### 4.9.0.7 Sequenza operativa.

1. Il supplicant invia **EAPoL-Start**.

2. L'autenticator risponde con **EAP-Request/Identity**.
3. Il client replica con **EAP-Response/Identity**, incapsulato in **Access-Request** verso il server.
4. Il server RADIUS valuta le credenziali e restituisce **Access-Challenge** (MD5) o **Access-Accept**.
5. A successo, l'autenticator invia **EAP-Success** e abilita la porta: l'host entra nella VLAN autorizzata.
6. Se l'autenticazione fallisce, la porta resta bloccata e il traffico viene scartato.

#### **4.9.0.8 Osservazioni e risultati.**

- Gli host autenticati vengono inseriti automaticamente nella VLAN definita nel profilo RADIUS.
- Il controllo del traffico via `ebtables` garantisce isolamento tra utenti non autenticati e autenticati.
- I pacchetti EAPoL e RADIUS possono essere catturati con `tcpdump` per analizzare lo scambio.
- La struttura è facilmente estendibile ad altri metodi EAP (es. **EAP-TLS**) per autenticazione con certificati.

**4.9.0.9 Conclusioni.** Il laboratorio dimostra come lo standard IEEE 802.1X realizzi un **controllo d'accesso per porta** efficace e scalabile, spostando l'intelligenza di autenticazione sul server centrale e permettendo politiche per-utente (VLAN, ACL, UCL). La combinazione di `hostapd`, `FreeRADIUS` e `wpa_supplicant` rappresenta un'implementazione open-source completa del paradigma 802.1X.

## 4.10 MACsec e MKA (802.1AE + 802.1X-2010)

### 4.10.1 Perché MKA: il tassello mancante dopo 802.1X

Dopo l'autenticazione 802.1X, la porta diventa autorizzata ma il *traffico L2* rimane in chiaro e vulnerabile a spoofing e manomissione. **MACsec** (IEEE 802.1AE) aggiunge confidenzialità, integrità e anti-replay ai frame Ethernet sul *link locale*, mentre **MKA** (MACsec Key Agreement, parte di 802.1X-2010) *negozi e distribuisce* in modo sicuro le chiavi necessarie a MACsec. In breve: 802.1X decide *chi entra*, MACsec+MKA decide *come proteggere* i frame una volta dentro.

### 4.10.2 Le chiavi in gioco e come si ottengono

Il cuore di MKA è una piccola filiera crittografica.

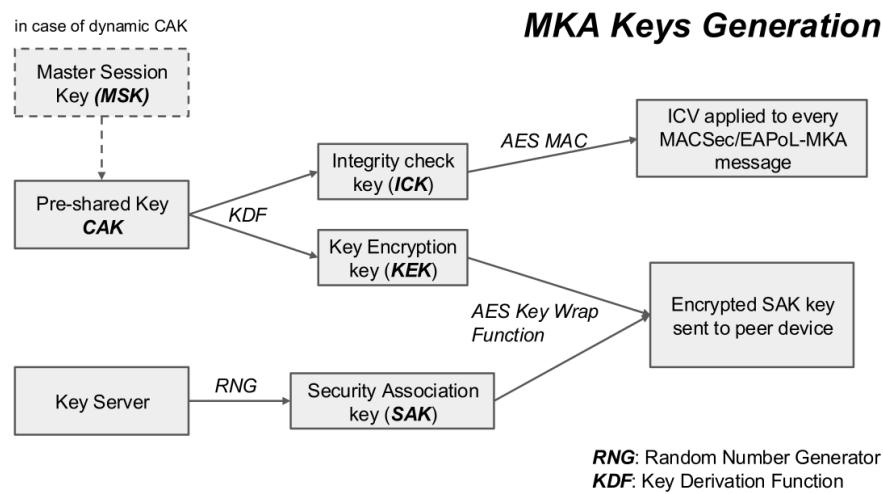


Figura 32: MKA keys generation: dal CAK (statico o derivato da MSK) si ottengono ICK e KEK (KDF); il Key Server genera la SAK (RNG) e la distribuisce cifrata (AES Key Wrap), mentre l'ICK protegge i messaggi EAPOL-MKA (ICV).

**4.10.2.1 CAK e CKN** La **CAK** (*Connectivity Association Key*) è la *chiave di base* condivisa (pre-shared) da tutti i partecipanti alla stessa *Connectivity Association* (CA), cioè al gruppo logico che userà MACsec su quel link. La CAK *non cifra i dati* direttamente: serve per proteggere il piano di controllo MKA e per derivare le chiavi operative. Il **CKN** (*Connectivity association Key Name*) è il *nome/identificatore della CA*: non è segreto, ma consente ai peer di riconoscere a quale associazione stanno aderendo e di far coesistere più CA sullo stesso mezzo. In pratica, *CKN individua il gruppo, CAK ne è il segreto condiviso*. Cambiando CKN o CAK si forza la (ri)formazione della CA e un nuovo ciclo di chiavi.

**4.10.2.2 ICK e KEK** A partire dalla CAK, MKA applica una **KDF** (Key Derivation Function) e ricava due chiavi di controllo con ruoli distinti: **ICK** (*Integrity Check Key*) e **KEK** (*Key Encryption Key*). L'ICK viene usata per calcolare un *ICV* (Integrity Check Value) sui messaggi EAPOL-MKA, garantendo che l'annuncio di capacità, l'elezione del Key Server e la distribuzione delle chiavi non possano essere manipolati. La KEK, invece, serve per *cifrare e incapsulare* (es. con AES Key Wrap) le chiavi dati che il Key Server invia ai peer. Nota importante: *ICK e KEK proteggono solo il piano di controllo*; il traffico utente su Ethernet verrà protetto da chiavi diverse (le SAK).

**4.10.2.3 SAK** La **SAK** (*Secure Association Key*) è la chiave *operativa* che MACsec usa sui *Secure Channel* per proteggere i frame (confidenzialità, integrità e anti-replay). Le SAK sono generate dal *Key Server* con un RNG, poi distribuite ai peer cifrandole con la KEK e firmandole con l'ICK. Ogni direzione di trasmissione usa la propria Secure Association (quindi SAK distinte per TX/RX), con contatori di pacchetto (*Packet Number*) per impedire riutilizzi (*replay*). Periodicamente, o al verificarsi di certe condizioni (es. soglie di PN), il Key Server esegue un *rekey* installando nuove SAK senza interrompere il traffico.

#### 4.10.3 Due modalità d'uso del CAK

**4.10.3.1 Static CAK** La CAK è precondivisa manualmente tra i nodi (insieme al CKN). Pro: avvio semplice, ideale su link infrastrutturali stabili (switch–switch, switch–router). Contro: rotazione chiavi e governance sono manuali; non c'è un legame diretto con un'identità utente o con policy per-utente.

**4.10.3.2 Dynamic CAK** La CAK non si configura a mano: è derivata dal **MSK** (*Master Session Key*) rilasciato dal server RADIUS al termine dell'autenticazione 802.1X/EAP. Questo lega la CA a un'identità autenticata e consente rotazioni automatiche al rinnovo della sessione EAP. È lo schema tipico host–switch (NAC): l'host si autentica, il dispositivo di accesso e il peer ricavano localmente CAK/CKN tramite KDF, si elegge il Key Server e partono distribuzione SAK e protezione MACsec.

#### 4.10.4 Cosa si scambiano i peer: il ciclo MKA

MKA usa messaggi **EAPOL-MKA** (*MKPDU*) periodici per mantenere la lista dei peer *vivi*, concordare chiavi e parametri, e rilevare variazioni di membership. Ogni MKPDU è protetto con un *ICV* calcolato tramite **ICK**, così che solo chi possiede la **CAK** possa partecipare.

**4.10.4.1 Annuncio, stato e priorità** Ad intervalli regolari, ciascun nodo invia:

- le proprie capacità *MACsec* (suite, offset di confidenzialità, supporto replay protection);
- il proprio identificatore di membro (*MI*) e contatore di messaggi (*MN*) per il controllo di liveness;
- la lista dei peer *vivi* (chi ho visto di recente) per allineare la vista di gruppo;
- la priorità di *Key Server* e parametri di tie-break (es. indirizzo MAC).

Questo scambio, protetto da ICK, impedisce a terzi di iniettare o alterare lo stato del gruppo.

**4.10.4.2 Elezione del Key Server** Tutti i partecipanti applicano la stessa regola di elezione (priorità amministrativa e tie-break deterministico). Il vincitore diventa **Key Server** e:

1. genera le **SAK** con un RNG robusto;
2. sceglie i parametri di uso (es. *Association Number*, finestra anti-replay);
3. distribuisce le SAK ai peer cifrandole con la KEK e firmando i messaggi con l'ICK (es. AES Key Wrap + ICV).

Il rekey può essere attivato da timer, soglie di *Packet Number*, o cambi di membership.

**4.10.4.3 Apertura dei Secure Channel** Installata la SAK:

- ogni direzione crea un **Secure Channel** (SC) con una o più **Secure Association** (SA), identificate da *AN*;

- il trasmettitore usa la SAK per la propria SA *TX*, i ricevitori la installano come SA *RX*;
- MACsec aggiunge il *SecTAG* e calcola l'ICV sul frame; il contatore di pacchetto (*PN*) cresce monotonicamente e abilita la protezione *anti-replay* (con finestra configurata).

**4.10.4.4 Vita della sessione** Gli *heartbeat* (MKPDU periodici) mantengono sincronizzati MI/MN e la lista dei peer. Se un nodo non si vede per oltre il timeout, viene rimosso dal gruppo e il Key Server invalida le SA pertinenti, avviando un *rekey* per i restanti membri. In caso di rientro o di nuovo peer, la membership viene aggiornata e si negoziano SAK fresche senza interrompere il traffico.

#### 4.10.5 Collegamento con le policy 802.1X/NAC

**802.1X/NAC** decide *chi entra e con quali permessi*; **MKA/MACsec** decide *come vengono protetti i frame* sul link.

1. L'host si autentica via 802.1X/EAP; il NAC applica policy (*VLAN dinamiche, ACL/UCL* per utente/gruppo).
2. (Opzionale) In *Dynamic CAK* la **CAK** è derivata dal **MSK** della sessione EAP, legando la sicurezza L2 all'identità autenticata.
3. MKA forma la *Connectivity Association* (CKN/CAK), elegge il Key Server e distribuisce le **SAK**.
4. Da quel momento, il traffico sulla porta autorizzata è protetto da **MACsec** (confidenzialità, integrità, anti-replay) *indipendentemente* dalla VLAN o dalle ACL applicate.

In sintesi: 802.1X/NAC governa *identità e policy*; MKA/MACsec fornisce *protezione crittografica del link*. Le due componenti sono complementari e coordinate: un cambio di identità/policy (ri-autenticazione) può innescare una nuova CAK e un nuovo ciclo di SAK.

# Laboratorio

## 4.11 Simple MKA lab with Linux

## 4.12 Laboratorio: MKA/MACsec su Linux (Static CAK)

### 4.12.1 Scopo

Configurare **MKA** (802.1X-2010) per distribuire le **SAK** di **MACsec** su un link Ethernet, creando l’interfaccia `macsec0` e verificando cifratura/integrità del traffico L2.

### 4.12.2 Topologia

Due host Linux collegati back-to-back (o tramite switch trasparente a L2). Ogni host esegue `wpa_supplicant` con driver `macsec_linux` per negoziare MKA.

### Topology

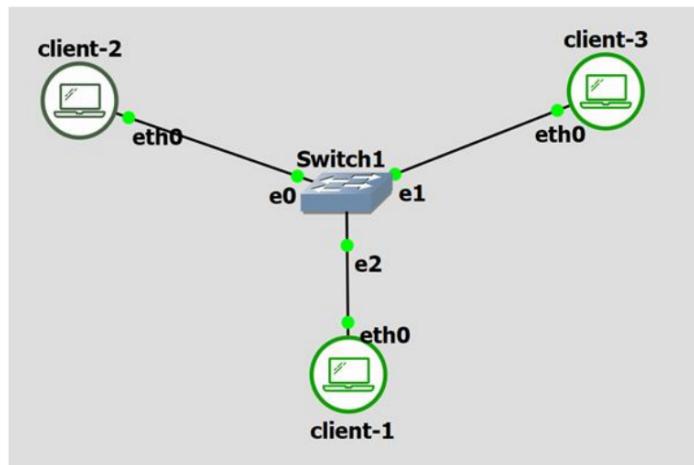


Figura 33: (topologia della rete)

### 4.12.3 Procedura

#### 4.12.3.1 Generazione delle chiavi (uguali su *tutti* i peer)

```
# CAK = 16 byte esadecimali (32 hex chars)
openssl rand -hex 16
# CKN = 32 byte esadecimali (64 hex chars)
openssl rand -hex 32
```

#### 4.12.3.2 Configurazione MKA su ciascun host

Creare `macsec.conf` (uguale su entrambi, con stessi `mka_cak` e `mka_ckn`):

```
eapol_version=3          # 802.1X-2020; richiesto per wired
ap_scan=0                # profilo wired
network={
    key_mgmt=NONE        # wired
    eapol_flags=0         # wired
```

```

    macsec_policy=1    # abilita MACsec se possibile
    mka_cak=0123456789abcdef0123456789abcdef
    mka_ckn=00112233445566778899aabbccddeeff0011223344556677
}

```

#### 4.12.3.3 Avvio di MKA e creazione dell'interfaccia `macsec0`

```
wpa_supplicant -i eth0 -B -D macsec_linux -c macsec.conf
```

Se l'handshake MKA va a buon fine, il kernel crea `macsec0`.

#### 4.12.3.4 Indirizzamento IP sulla `macsec0`

```
ip addr add 10.0.1.1/24 dev macsec0    # Host A
ip addr add 10.0.1.2/24 dev macsec0    # Host B
ip link set macsec0 up
```

#### 4.12.3.5 Test base di connettività

```
ping 10.0.1.2      # da Host A verso Host B
```

Il traffico passa attraverso `macsec0` con cifratura/integrità a L2.

### 4.12.4 Verifiche

#### 4.12.4.1 Log MKA

Nei log di `wpa_supplicant` attendersi, in sequenza: *Key Server announcement* → *election process* → *SAK distribution*.

#### 4.12.4.2 Stato interfaccia

```
ip link show macsec0
```

L'interfaccia deve essere UP e usata come device IP.

### 4.12.5 Troubleshooting

- **Interfaccia `macsec0` assente:** verificare `-D macsec_linux, ap_scan=0, eapol_version=3`.
- **Errore sulle chiavi:** `mka_cak` deve essere esattamente 16B (32 hex); `mka_ckn` 32B (64 hex).
- **Niente traffico IP:** assegnare l'IP su `macsec0`, non su `eth0`.
- **Mancata elezione del Key Server:** assicurarsi che entrambi i peer siano *vivi* (link up) e condividano CKN/CAK.

### 4.12.6 Pulizia

```
pkill wpa_supplicant
ip link del macsec0  # se presente
```

### 4.12.7 Estensioni & domande d'esame

#### 4.12.7.1 Dynamic CAK (accenno)

In alternativa allo *Static CAK*, il **CAK** può essere derivato dal **MSK** ottenuto via 802.1X/EAP con backend RADIUS; il resto (elezione Key Server, distribuzione SAK) è invariato.

## 5 NET\_05 — Firewall e Algoritmi di Classificazione dei Pacchetti

### 5.1 Panoramica sui Firewall

I firewall sono componenti fondamentali per la sicurezza delle reti, in quanto controllano il flusso di traffico tra reti interne di fiducia e reti esterne non sicure, come Internet. Agiscono come una barriera, bloccando o consentendo selettivamente i pacchetti in base a una politica di sicurezza definita. I firewall operano come un confine tra reti interne sicure e ambienti esterni non sicuri, e possono essere implementati sia in hardware che in software.

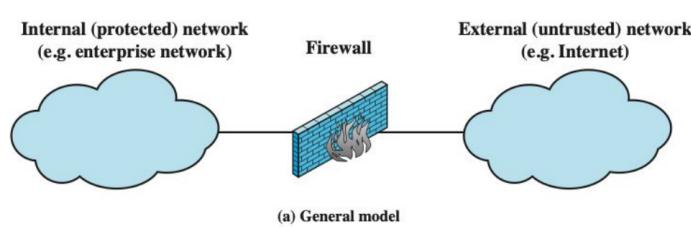


Figura 34: Modello generale di firewall

#### 5.1.1 Obiettivi di Progettazione dei Firewall

Gli obiettivi principali dei firewall includono:

- **Monitoraggio del Traffico:** Tutto il traffico deve passare attraverso il firewall, creando un unico punto di controllo per il monitoraggio e il controllo degli accessi alla rete.
- **Applicazione della Politica di Sicurezza:** Solo il traffico autorizzato dalla politica di sicurezza è consentito. Ciò include il filtraggio basato su indirizzi IP, porte, protocolli e tipi di applicazioni.
- **Resistenza alla Penetrazione:** Il firewall stesso deve essere altamente sicuro e resistente a tentativi di penetrazione, assicurando che gli attaccanti non possano facilmente eluderlo.

#### 5.1.2 Politica di Accesso del Firewall

La politica di accesso definisce i tipi di traffico che possono attraversare il firewall. Essa include:

- **Intervalli di Indirizzi IP:** Definisce quali intervalli di indirizzi IP sono autorizzati o bloccati.
- **Protocolli:** Specifica i protocolli di rete consentiti o negati (ad esempio, TCP, UDP, ICMP).
- **Applicazioni:** Identifica applicazioni o servizi specifici che sono permessi o bloccati.
- **Tipi di Contenuto:** Esamina il contenuto dei pacchetti (ad esempio, i metodi di richiesta HTTP) per far rispettare le politiche di sicurezza.

Queste politiche devono essere sviluppate sulla base della valutazione del rischio dell'organizzazione. Le politiche sono inizialmente definite a un livello elevato e successivamente perfezionate in regole dettagliate da implementare nel firewall. Questo processo implica comprendere i tipi di traffico necessari per le operazioni aziendali, minimizzando al contempo l'esposizione ai potenziali rischi.

### 5.1.3 Caratteristiche dei Firewall

Le caratteristiche principali dei firewall includono:

- **Punto di Controllo Unico:** Un firewall definisce un punto di choke in cui tutto il traffico in ingresso e in uscita viene ispezionato.
- **Monitoraggio Centralizzato della Sicurezza:** Il firewall funge da piattaforma per monitorare eventi di sicurezza e traffico di rete. Questa centralizzazione consente una gestione e un logging più facili.
- **Protezione dei Confini:** Il firewall fornisce una protezione sicura tra le reti interne e quelle esterne, controllando l'accesso in base a politiche di sicurezza definite.

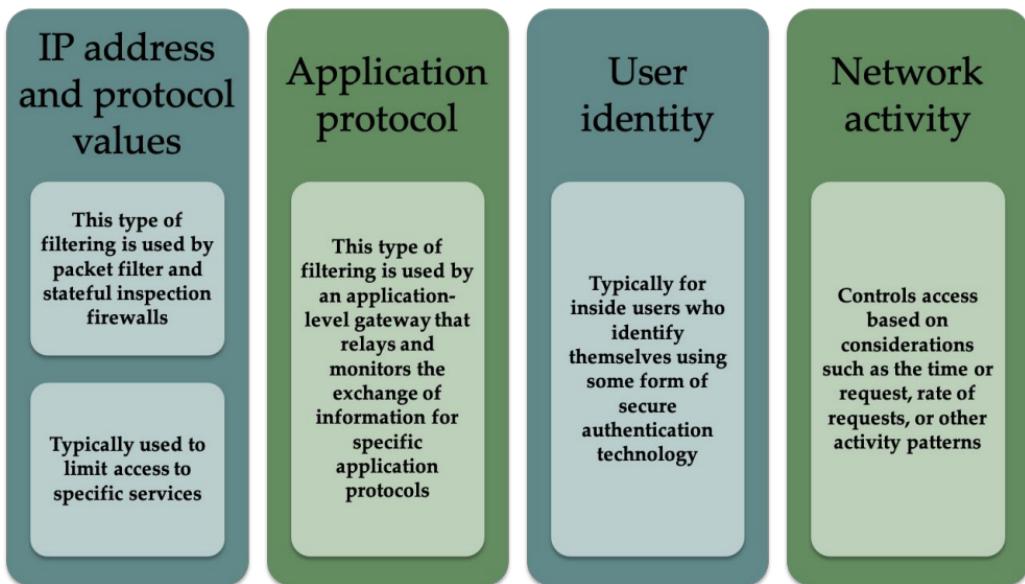


Figura 35: Criteri di filtraggio adottati dai firewall: indirizzi IP e protocolli, protocolli applicativi, identità dell'utente e attività di rete. Ogni categoria definisce un diverso livello di controllo sull'accesso e sul traffico.

Tuttavia, i firewall hanno delle limitazioni:

- **Non Proteggono dai Rischi Interni:** I firewall non sono generalmente in grado di proteggere da attacchi che provengono da sistemi compromessi all'interno della rete o da utenti non autorizzati già presenti nel sistema.
- **Reti Wireless e VPN:** I firewall potrebbero non proteggere sempre dagli attacchi tramite reti wireless non sicure o VPN, che possono bypassare i confini tradizionali del firewall.

## 5.2 Firewall di Filtraggio dei Pacchetti

Un **firewall di filtraggio dei pacchetti** applica un insieme di regole a ciascun pacchetto in ingresso e in uscita. Il firewall esamina l'intestazione di ciascun pacchetto e determina se deve essere inoltrato o scartato in base ai criteri definiti. I criteri di solito includono:

- **Indirizzo IP Sorgente:** L'indirizzo IP da cui origina il pacchetto.
- **Indirizzo IP di Destinazione:** L'indirizzo IP a cui è destinato il pacchetto.
- **Numero di Porta Sorgente e di Destinazione:** Questi numeri vengono utilizzati per identificare applicazioni o servizi specifici.

- **Campo Protocollo:** Specifica il protocollo utilizzato (ad esempio, TCP, UDP, ICMP).

I firewall di filtraggio dei pacchetti operano valutando ciascun pacchetto rispetto a un insieme di regole predefinite, e i pacchetti che soddisfano i criteri per l'accettazione vengono lasciati passare, altrimenti vengono scartati.

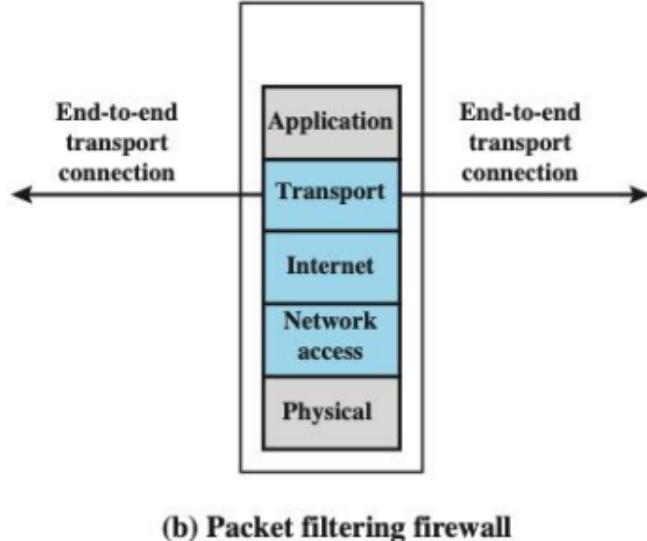


Figura 36: Schema di un **packet filtering firewall**: il controllo avviene ai livelli inferiori dello stack (fino al trasporto), filtrando i pacchetti in base a indirizzi, porte e protocolli prima che raggiungano le applicazioni.

### 5.2.1 Vantaggi e Svantaggi del Filtraggio dei Pacchetti

**Vantaggi:**

- **Semplicità:** Il filtraggio dei pacchetti è relativamente semplice da implementare e configurare. È un processo rapido che non richiede un'ispezione approfondita del contenuto del pacchetto.
- **Prestazioni:** I firewall di filtraggio dei pacchetti sono efficienti e trasparenti per gli utenti, con un minimo overhead che consente un alto throughput.

Rule	Direction	Src address	Dest address	Protocol	Dest port	Action
1	In	External	Internal	TCP	25	Permit
2	Out	Internal	External	TCP	>1023	Permit
3	Out	Internal	External	TCP	25	Permit
4	In	External	Internal	TCP	>1023	Permit
5	Either	Any	Any	Any	Any	Deny

Figura 37: Esempio di regole di filtraggio di un firewall basato su policy TCP: le regole consentono il traffico in ingresso e in uscita per porte specifiche (25 e >1023), mentre tutto il resto viene negato per impostazione predefinita.

**Svantaggi:**

- **Vulnerabilità Specifiche dell'Applicazione:** Un firewall di filtraggio dei pacchetti non può prevenire attacchi che sfruttano vulnerabilità specifiche delle applicazioni. Ad esempio,

un attaccante può sfruttare debolezze nelle applicazioni web che non vengono rilevate dal filtraggio dei pacchetti.

- **Limitate Capacità di Logging e Audit:** I firewall di filtraggio dei pacchetti di solito hanno capacità limitate di registrare e analizzare dettagliatamente il traffico di rete, il che rende più difficile rilevare attacchi sofisticati.
- **Vulnerabilità nei Protocolli TCP/IP:** I firewall di filtraggio dei pacchetti potrebbero essere vulnerabili a attacchi che prendono di mira le debolezze nella suite di protocolli TCP/IP (ad esempio, flooding SYN, spoofing IP).

### 5.3 Firewall di Ispezione Stateful

Un **firewall di ispezione stateful** migliora il filtraggio dei pacchetti tracciando lo stato delle connessioni attive. A differenza dei firewall di filtraggio dei pacchetti, che ispezionano ogni pacchetto individualmente, i firewall stateful mantengono una *tabella dello stato delle connessioni*, che registra lo stato delle connessioni attive. Ciò consente al firewall di tracciare l'intero ciclo di vita di una connessione, assicurandosi che solo i pacchetti appartenenti a connessioni legittime siano consentiti.

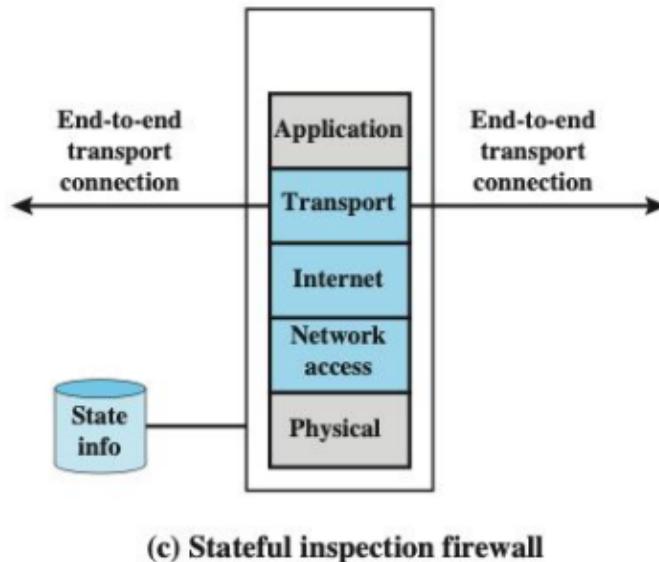


Figura 38: Schema di uno **stateful inspection firewall**: monitora lo stato delle connessioni e mantiene informazioni di sessione (*state info*) per consentire solo pacchetti appartenenti a flussi legittimi.

#### 5.3.1 Tabella dello Stato delle Connessioni

La tabella dello stato delle connessioni registra le informazioni su ciascuna connessione stabilita, tra cui:

- **Inizio della Connessione:** Identifica l'inizio di nuove connessioni, come i pacchetti SYN in TCP.
- **Stato della Connessione:** Traccia lo stato della connessione, che può essere NEW, ESTABLISHED o RELATED.
- **Protocolli Rilevanti:** Il firewall traccia i protocolli coinvolti, come FTP, HTTP o altri protocolli a livello applicativo.

Questo fornisce una difesa più robusta contro gli attacchi, poiché il firewall si assicura che solo i pacchetti che appartengono a una sessione stabilita siano consentiti. Questo previene che gli attaccanti possano falsificare pacchetti facendoli sembrare appartenenti a connessioni legittime.

## 5.4 Gateway e Funzione del Default Gateway

Un **gateway** è un dispositivo che funge da punto di accesso tra due reti, permettendo la comunicazione tra reti con tecnologie diverse. Nel contesto di un firewall, il gateway è spesso posizionato come perimetro di rete per proteggere l'interno della rete da attacchi provenienti dall'esterno. Nel caso di un firewall stateful, il gateway monitora anche lo stato delle connessioni per garantire che solo il traffico legittimo venga consentito attraverso il firewall.

### 5.4.1 Funzioni di un Gateway

Il gateway ha il compito di:

- **Filtraggio del Traffico:** Il gateway può applicare regole di filtraggio per determinare quali pacchetti possono passare attraverso la rete. Queste regole sono generalmente impostate in base a politiche di sicurezza definite.
- **Network Address Translation (NAT):** Il gateway può svolgere il NAT per modificare gli indirizzi IP di origine o destinazione dei pacchetti che transitano. Questo è utile in ambienti domestici o aziendali con pochi indirizzi IP pubblici.
- **Routing:** Il gateway indirizza il traffico tra reti locali (LAN) e reti esterne (Internet), gestendo il flusso di pacchetti tra reti diverse.

## 5.5 Application-Level Gateway (ALG)

Un **Application-Level Gateway** (ALG) è un tipo di gateway che agisce come un proxy per i protocolli applicativi. Invece di gestire il traffico a livello di rete, come fanno i firewall di filtraggio dei pacchetti, un ALG analizza e modula il traffico applicativo, come HTTP o FTP.

### 5.5.1 Funzionamento dell'ALG

L'ALG agisce come un intermediario tra il client e il server:

- L'utente invia una richiesta a un gateway applicativo tramite un'applicazione TCP/IP.
- Il gateway applicativo autentica l'utente e inoltra la richiesta al server remoto.
- I dati sono quindi inoltrati tra l'utente e il server attraverso il gateway.

Questo tipo di gateway è spesso più sicuro rispetto ai firewall packet filtering, poiché ispeziona il contenuto applicativo dei pacchetti. Tuttavia, ha anche uno svantaggio significativo in termini di overhead computazionale, poiché ogni connessione deve essere gestita dal gateway. Un grande svantaggio di questa tipologia è l'overhead di processamento che viene introdotto su ogni connessione.

## 5.6 Circuit-Level Gateway

Il **Circuit-Level Gateway** (CLG) è simile all'ALG, ma opera a livello di connessione piuttosto che di applicazione. Il CLG stabilisce due connessioni TCP: una tra il client interno e il gateway, e una tra il gateway e il server esterno. Non esamina il contenuto del pacchetto, ma si limita a verificare che le connessioni siano valide.

### 5.6.1 Funzione del Circuit-Level Gateway

Il gateway stabilisce e gestisce due connessioni TCP separate:

- Una connessione tra il client interno e il gateway,
- Una connessione tra il gateway e il server esterno.

Questo tipo di gateway è meno sicuro rispetto agli ALG, poiché non esamina il contenuto delle connessioni, ma ha un overhead inferiore rispetto agli ALG.

## 5.7 Host-Based Firewall e Personal Firewall

Un **host-based firewall** è un firewall che viene implementato direttamente sui dispositivi terminali (computer, server, dispositivi mobili), monitorando il traffico in ingresso e in uscita dal singolo dispositivo. Il suo compito principale è proteggere il dispositivo da attacchi interni o esterni, applicando regole di filtraggio definite.

I principali vantaggi del **host-based firewall** sono:

- **Protezione mirata** per il dispositivo.
- **Controllo fine del traffico** tra il dispositivo e la rete.

Gli svantaggi includono un possibile **overhead** sulle prestazioni e la **gestione complessa** su larga scala, poiché ogni dispositivo deve essere configurato separatamente.

Il **personal firewall**, invece, è un firewall progettato per proteggere dispositivi individuali (ad esempio computer portatili o desktop) da attacchi provenienti dalla rete o da internet. Funziona a livello dell'utente, bloccando traffico indesiderato in ingresso o uscita, proteggendo dalle scansioni di porte e impedendo l'accesso non autorizzato.

I vantaggi di un **personal firewall** sono:

- **Protezione specifica** per ogni dispositivo.
- **Facilità di configurazione** e gestione da parte dell'utente.

Gli svantaggi comprendono la **protezione limitata** al solo dispositivo e la **difficoltà di gestione** in ambienti con molti dispositivi da proteggere.

## 5.8 NETFILTER

NETFILTER è un framework che permette di intercettare e manipolare i pacchetti di rete all'interno del kernel Linux. Utilizza punti di hook (entry point) nel sottosistema di rete IPv4/IPv6 del kernel, consentendo operazioni di mangling(modifiche dirette all header dei pacchetti in transito) sui pacchetti. Questi hook vengono invocati quando un pacchetto attraversa la pila IP, permettendo di applicare regole di filtraggio e trasformazione prima che il pacchetto venga inoltrato.

### 5.8.1 Funzionamento di NETFILTER

I pacchetti che attraversano la rete (in ingresso, in uscita o inoltrati) vengono intercettati dai seguenti hook:

- **PRE\_ROUTING**: Il pacchetto viene intercettato prima che venga presa una decisione di routing.
- **LOCAL\_INPUT**: Il pacchetto è destinato al dispositivo locale.

- **FORWARD**: Il pacchetto viene inoltrato a un altro dispositivo.
- **LOCAL\_OUTPUT**: Il pacchetto viene originato da un processo locale.
- **POST\_ROUTING**: Il pacchetto è stato instradato e sta per uscire dal dispositivo.

Ogni pacchetto intercettato dai hook passa attraverso una serie di tabelle predefinite, che gestiscono diversi tipi di attività di rete e sono controllate da catene di trasformazione e filtraggio dei pacchetti.

### 5.8.2 Le Tabelle di NETFILTER

Esistono quattro tabelle principali in NETFILTER:

- **Filter**: Utilizzata per il filtraggio dei pacchetti (accettare, rifiutare).
- **NAT**: Gestisce la traduzione degli indirizzi di rete (SNAT, DNAT, Masquerading).
- **Mangle**: Modifica le intestazioni dei pacchetti, ad esempio il TTL (Time-to-Live) o il TOS (Type of Service).
- **Raw**: Viene utilizzata per configurare le eccezioni dal tracciamento delle connessioni, come nel caso della connessione "NOTRACK".

Le regole in ciascuna tabella definiscono come devono essere trattati i pacchetti che soddisfano determinate condizioni. Ogni tabella ha una propria catena di regole che determinano le azioni da intraprendere, come "ACCEPT", "DROP" o "MASQUERADE".

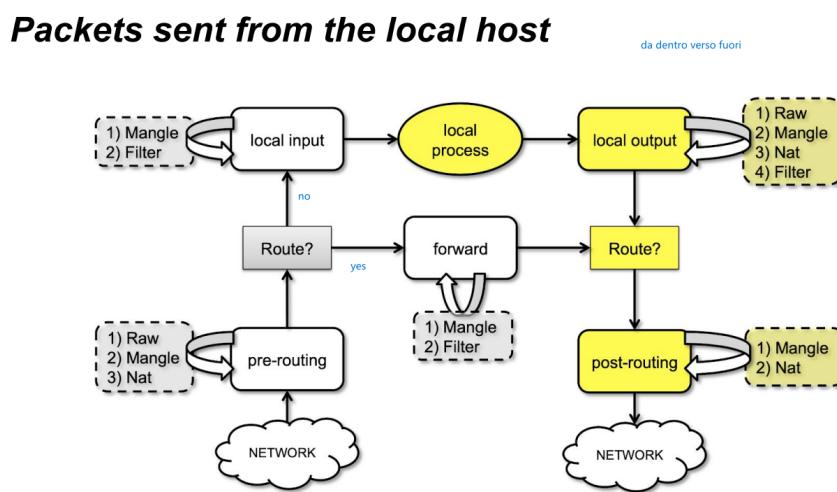


Figura 39: processamento interno con forwarding esterno

## Packets sent to a local address

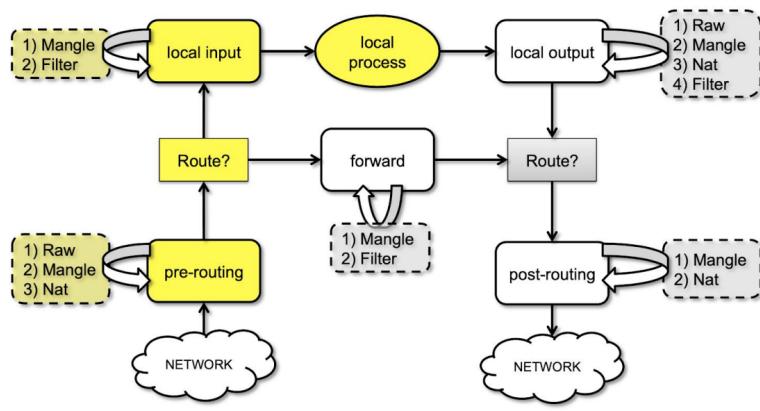


Figura 40: arrivo esterno con processamento interno

## Forwarded Packets

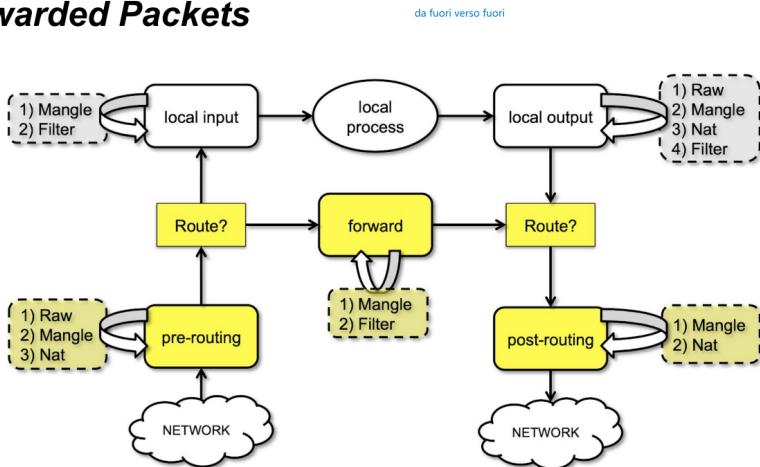


Figura 41: Forwarding puro

### 5.8.3 Iptables: Il Frontend di NETFILTER

Iptables è l'applicazione utente che consente di configurare le tabelle di NETFILTER nel kernel Linux. Con iptables, è possibile aggiungere, rimuovere e ispezionare le regole delle tabelle per il filtraggio dei pacchetti IPv4. Ogni tabella contiene una serie di catene predefinite, ma è anche possibile definire catene personalizzate.

#### 5.8.3.1 Comandi principali di iptables

Alcuni dei comandi principali di iptables includono:

- **-A:** Aggiunge una regola alla fine di una catena.
- **-D:** Rimuove una regola da una catena.
- **-I:** Inserisce una regola all'inizio di una catena.
- **-R:** Sostituisce una regola esistente.
- **-P:** Imposta la politica di una catena.
- **-F:** Elimina tutte le regole in una catena.
- **-S:** Mostra tutte le regole di una catena.

Un esempio di comando per aggiungere una regola in iptables è:

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

Questo comando consente il traffico TCP in ingresso sulla porta 80.

#### 5.8.4 Network Address Translation (NAT)

Il NAT è una funzione importante di NETFILTER che consente di modificare gli indirizzi IP di origine o destinazione dei pacchetti. In Linux, il NAT è implementato da NETFILTER e può essere configurato per applicare la traduzione degli indirizzi in modo statico o dinamico.

##### 5.8.4.1 Tipi di NAT

- **Source NAT (SNAT)**: Modifica l'indirizzo IP di origine di un pacchetto, utile per mascherare gli indirizzi IP di una rete privata dietro un singolo indirizzo IP pubblico.
- **Destination NAT (DNAT)**: Modifica l'indirizzo di destinazione di un pacchetto, spesso utilizzato per il port forwarding.
- **Masquerading**: Una forma dinamica di SNAT che cambia l'indirizzo di origine ogni volta che un nuovo flusso di traffico viene avviato.

Un esempio di comando per il masquerading con iptables è:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

#### 5.8.5 Connection Tracking

NETFILTER tiene traccia dello stato delle connessioni attraverso il modulo di **conntrack**. I pacchetti vengono classificati in quattro stati: NEW, ESTABLISHED, RELATED, e INVALID. Questo permette di applicare regole più specifiche in base allo stato della connessione.

##### 5.8.5.1 Esempi di stato delle connessioni

- **NEW**: La connessione è appena stata avviata.
- **ESTABLISHED**: La connessione è stata completamente negoziata.
- **RELATED**: Il pacchetto è associato a una connessione già stabilita.
- **INVALID**: Il pacchetto non è valido.

Per visualizzare le connessioni tracciate, è possibile utilizzare il comando:

```
cat /proc/net/ip_conntrack
```

#### 5.8.6 Vantaggi e Limitazioni di NETFILTER

**Vantaggi:**

- Alta flessibilità nella configurazione delle regole di filtraggio e NAT.
- Supporto per diversi tipi di trasformazione dei pacchetti (filtraggio, NAT, mangling).
- Integrazione nativa con il kernel Linux.

**Limitazioni:**

- Scalabilità limitata quando si gestiscono grandi quantità di regole (come nei grandi data center).

- La gestione di regole complesse può diventare difficoltosa in ambienti con configurazioni avanzate.

# Laboratorio

## 5.9 Lab 5 — Firewall con `iptables` e policy di sicurezza gateway/server

**5.9.0.1 Obiettivo e componenti.** Realizzare e testare una configurazione firewall completa con **Linux netfilter/iptables**, seguendo la topologia logica *LAN–DMZ–WAN*. Dimostrare come applicare politiche di accesso, NAT e filtraggio basate sullo stato delle connessioni. Ambiente: un **gateway/firewall** Linux con tre interfacce (**LAN**, **DMZ**, **WAN**), un **server DMZ** e host di test nella **LAN** e nella rete esterna.

**5.9.0.2 Struttura del laboratorio.** Si configurano le tre zone:

- **LAN** (10.0.0.0/24): rete interna aziendale, soggetta a NAT in uscita;
- **DMZ** (160.80.200.0/24): rete pubblica per server esposti;
- **WAN** (1.0.0.0/30): collegamento al router Internet (R1).

Il gateway funge da *firewall perimetrale* e da router tra le tre reti, con forwarding IPv4 abilitato.

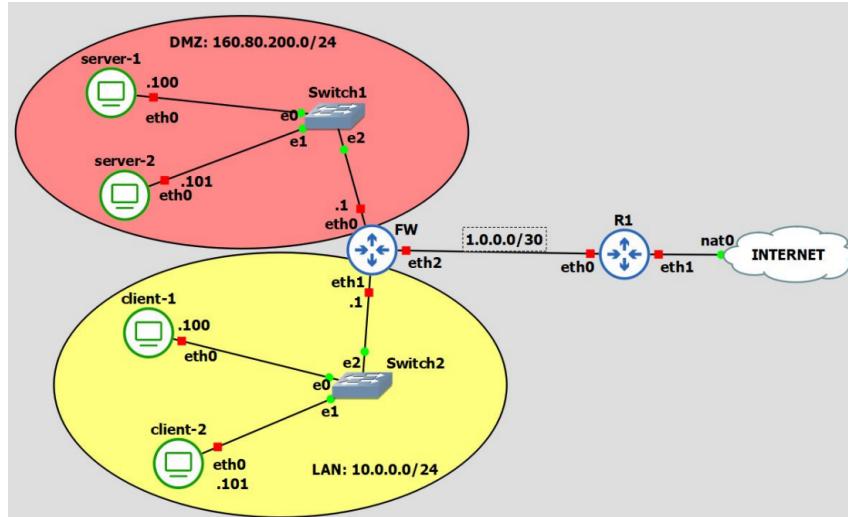


Figura 42: Topologia di rete con firewall e zona demilitarizzata (DMZ): i server pubblici risiedono nella DMZ (160.80.200.0/24), separata dalla LAN interna (10.0.0.0/24). Il firewall (FW) filtra il traffico tra LAN, DMZ e Internet, collegato al router R1 tramite la rete 1.0.0.0/30.

**5.9.0.3 Gateway (firewall principale).** `iptables -F` per azzerare le regole; politiche di default:

```
iptables -P FORWARD DROP
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
```

Le regole successive implementano la policy di sicurezza:

- Consenti solo connessioni già **stabilite**:

```
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
```

- Consenti dalla LAN verso Internet servizi HTTP/HTTPS/SSH/DNS:

```
iptables -A FORWARD -i $LAN -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i $LAN -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -i $LAN -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i $LAN -p udp --dport 53 -j ACCEPT
```

- Consenti traffico LAN<->DMZ solo se iniziato dalla LAN:

```
iptables -A FORWARD -i $LAN -o $DMZ -j ACCEPT
```

- Permetti ICMP per diagnosi (ping, echo reply):

```
iptables -A INPUT -p icmp -j ACCEPT
iptables -A FORWARD -p icmp -j ACCEPT
```

- NAT in uscita (masquerading verso WAN):

```
iptables -t nat -A POSTROUTING -o $WAN -j MASQUERADE
```

#### 5.9.0.4 Server DMZ.

Firewall locale con policy di default:

```
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
```

Regole per accettare solo i servizi pubblici e connessioni di ritorno:

```
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

#### 5.9.0.5 Testing.

Utilizzare ping e nc (netcat) per verificare la raggiungibilità e l'apertura delle porte:

- nc -l -p 80 sul server DMZ per aprire una porta TCP;
- nc <ip\_server> 80 dalla LAN o dal router R1 per verificare le regole;
- analizzare con tcpdump o iptables -L -v il traffico filtrato.

# Classificazione dei Pacchetti e Algoritmi di Matching

## 5.10 Classificazione dei Pacchetti

La **classificazione dei pacchetti** è una parte fondamentale dei firewall, della gestione della qualità del servizio (QoS), del routing multicast, e di altre applicazioni di rete come l'IDS, NAT, e il traffic engineering. In sostanza, si tratta di identificare il miglior “match” tra le informazioni contenute nell'intestazione di un pacchetto e le regole di classificazione definite.

**Esempio pratico:** Un router in una rete aziendale potrebbe dover classificare i pacchetti in base alla destinazione, alla porta, o persino al protocollo. Se il pacchetto proviene da un'applicazione di VoIP, potrebbe essere trattato con priorità rispetto a un pacchetto generico di dati HTTP.

### 5.10.1 Perché è Importante la Classificazione dei Pacchetti?

La classificazione dei pacchetti è essenziale non solo per i firewall, ma anche per:

- **Riservazione di risorse e QoS routing:** per garantire che il traffico critico riceva la priorità.
- **Routing unicast e multicast:** per instradare i pacchetti correttamente in base alle politiche predefinite.
- **NAT e monitoraggio del traffico:** per tracciare e gestire il flusso di pacchetti e proteggere la rete.

Le funzioni di classificazione vanno oltre il semplice matching degli indirizzi di destinazione IP e richiedono un'analisi più approfondita di campi multipli come l'indirizzo di origine, le porte, e persino i flag TCP. O magari per assegnare delle priorità.

**5.10.1.1 Requisiti e metriche.** In contesti ad alta velocità è cruciale bilanciare: **throughput** (cicli/tempo per pacchetto a 10/40/100+ Gbps), **memory footprint** (strutture dati e bitset), e **tempo di aggiornamento** (inserimento/rimozione regole, aggiornamento di stati). Tali vincoli guidano la scelta tra approcci software (cache-friendly, parallelizzabili) e hardware (TCAM/ASIC), oltre che il disegno di strutture multi-campo con buona località in memoria.

### 5.10.2 Algoritmi di Classificazione dei Pacchetti

I pacchetti vengono analizzati in base a una serie di campi dell'intestazione (header), come l'indirizzo di origine e destinazione, la porta di origine e destinazione, il protocollo IP, e altri. Questi campi vengono utilizzati per abbinare i pacchetti a regole definite in un database di regole.

Le regole di matching possono essere:

- **Exact Match:** il campo del pacchetto deve corrispondere esattamente con il campo della regola.
- **Prefix Match:** il campo del pacchetto viene confrontato con un prefisso della regola.
- **Range Match:** il campo del pacchetto deve rientrare in un intervallo specificato.

**Esempio pratico:** Un esempio di **Exact Match** potrebbe essere una regola che blocca i pacchetti provenienti da un determinato indirizzo IP, come nel caso di un attacco DoS da un IP malevolo. Un esempio di **Prefix Match** è l'abbinamento di un pacchetto con una rete intera, come nel caso di un filtro che consente solo pacchetti destinati a un range di indirizzi IP

‘192.168.1.0/24’. Infine, un **Range Match** potrebbe essere usato per consentire l’accesso a una gamma di porte, come 1000–2000 per un servizio FTP.

**5.10.2.1 Schema divide-and-conquer.** Un flusso tipico multi-campo è: (1) estrarre, per *ogni* campo, una struttura di lookup specializzata (es. trie per prefissi IP, strutture per range, hash per exact); (2) eseguire i lookup per campo; (3) **combinare** i risultati con tecniche come *Bit Vector AND*, *cross-producing* o *decision trees*, ottenendo l’insieme finale di regole candidate su cui applicare la selezione della “miglior regola”.

## 5.11 Problema della Classificazione dei Pacchetti

Il problema della classificazione dei pacchetti riguarda l’identificazione della regola di matching più appropriata per ciascun pacchetto in arrivo. Questo problema diventa complesso quando ci sono molte regole, e la ricerca di un match può richiedere una grande quantità di risorse computazionali.

Ogni pacchetto può essere confrontato con più regole, e la soluzione prevede l’adozione di algoritmi efficienti per ridurre la complessità della ricerca, specialmente in ambienti con un numero elevato di regole.

**Esempio pratico:** Immagina una rete aziendale con 10.000 regole nel firewall. Se il firewall utilizza una ricerca lineare per confrontare i pacchetti, la ricerca di ogni pacchetto potrebbe richiedere tempi significativi, rallentando il traffico in caso di picchi di accesso. Questo problema è ancora più evidente nei router di backbone, dove ogni millisecondo può fare la differenza nelle prestazioni complessive della rete.

### 5.11.1 Struttura delle Regole di Classificazione

Ogni regola di classificazione è composta da un insieme di campi di intestazione ( $H[1]$ ,  $H[2]$ , ...,  $H[K]$ ) e specifica come il traffico deve essere trattato. Le regole possono includere:

- Indirizzo di origine e destinazione.
- Porta di origine e destinazione.
- Tipo di protocollo (TCP, UDP, ICMP).
- Altri parametri come i flag TCP.

Il compito del classificatore è determinare quale regola corrisponde meglio a un pacchetto in ingresso e applicare l’azione associata a tale regola (ad esempio, ACCEPT, DROP).

**Esempio pratico:** Un pacchetto potrebbe corrispondere a una regola che consente solo pacchetti TCP sulla porta 80 (HTTP). Un altro pacchetto potrebbe corrispondere a una regola che accetta solo pacchetti ICMP (ping). Il sistema di classificazione determina quale regola applicare in base ai criteri specificati in ciascun campo dell’intestazione del pacchetto.

### 5.11.2 Combinazione di Regole con Match Multipli

Poiché un pacchetto può corrispondere a più regole, è necessario definire un metodo per risolvere questa ambiguità. Una soluzione comune è utilizzare una funzione di costo associata a ogni regola, che aiuta a determinare quale regola applicare in caso di conflitto.

Nel caso di un firewall, ad esempio, si può configurare l’ordine delle regole in modo che la regola con la priorità più alta venga applicata per prima.

**Esempio pratico:** Se un pacchetto corrisponde a una regola di tipo **DROP** e a una regola di tipo **ACCEPT**, la regola che ha la priorità più alta (determinata dalla posizione nella lista di regole) sarà applicata. Se le regole sono ordinate in ordine di priorità (ad esempio, le regole più restrittive vengono applicate per prime), il pacchetto verrà accettato o scartato in base alla prima regola che corrisponde.

**5.11.2.1 Selezione efficiente della miglior regola.** Dopo la combinazione (es. con Bit Vector), è utile trovare rapidamente la regola “migliore” secondo il costo/priorità. Un’ottimizzazione classica è l’uso di *de Bruijn* per individuare velocemente il bit più significativo/impostato nel vettore di candidati (massima priorità).

### 5.11.3 Algoritmi di Matching

Diversi algoritmi sono stati sviluppati per ottimizzare il processo di classificazione. I più comuni sono:

- **Algoritmi di Tries:** Utilizzati per il Longest Prefix Match (LPM), particolarmente efficaci per l’abbinamento degli indirizzi IP.
- **Ricerche Binaria:** Usate per trovare il miglior match tra un pacchetto e un insieme di regole, soprattutto in ambienti con regole ordinate.
- **Bit Vector Linear Search:** Ottimizza la ricerca riducendo il numero di confronti necessari utilizzando una rappresentazione in bit.
- **Content-Addressable Memory (CAM):** Memorie specializzate utilizzate per il matching rapido di regole, ma non sempre scalabili in software.

**Esempio pratico:** Il **Longest Prefix Match (LPM)** è utilizzato in scenari in cui è necessario fare il matching dell’indirizzo di destinazione di un pacchetto con un database di regole di routing. L’algoritmo trie è particolarmente utile in questo caso, in quanto permette di trovare rapidamente il prefisso più lungo che corrisponde a un pacchetto, riducendo significativamente il tempo di ricerca rispetto a una ricerca lineare.

**5.11.3.1 Altre famiglie chiave.** Oltre agli approcci sopra, nella pratica multi-campo si usano tre famiglie fondamentali:

- **Bit Vector + AND per campo:** modello divide-and-conquer che combina i match parziali.
- **Cross-producting:** pre-combina sottoinsiemi di risultati per accelerare la fase di combinazione.
- **Decision Trees:** visita guidata dei campi/regioni per ridurre lo spazio di ricerca.

Per estendere le prestazioni su più dimensioni si impiegano anche strutture come i **Set-Pruning Tries**, che potano lo spazio dei candidati attraversando i set validi per ciascun campo.

## 5.12 Approccio Bit Vector Linear Search

Il **Bit Vector Linear Search** è un metodo efficace per l’ottimizzazione della ricerca di regole. In questo approccio, ogni campo del pacchetto viene confrontato con le regole utilizzando una rappresentazione binaria (bitmask). Il risultato della ricerca di un match viene quindi combinato attraverso l’operazione logica AND, riducendo significativamente il numero di confronti da effettuare.

Ogni regola di classificazione è associata a una bitmask che rappresenta i campi di intestazione. L'operazione di AND tra le bitmask riduce il numero di regole che devono essere controllate, migliorando l'efficienza del matching.

### 5.12.1 Funzionamento del Bit Vector Linear Search

Il sistema di bit vector funziona confrontando il pacchetto con i set di regole. Ad esempio, se un pacchetto ha una destinazione IP che corrisponde a un prefisso specificato in una regola, questa regola può essere "marcata" come un potenziale match. Successivamente, la ricerca avviene combinando i risultati di ciascun campo (destinazione IP, porta, ecc.) con un'operazione AND, per ridurre il numero di regole rimanenti da confrontare.

**Esempio pratico:** Nel caso di un pacchetto con destinazione IP ‘192.168.1.100‘, il sistema verifica quale regola ha il prefisso più lungo corrispondente. Se una regola ha un prefisso ‘192.168.1.0/24‘ e un'altra ha ‘192.168.1.100/32‘, il sistema selezionerà la regola con il prefisso più specifico (quella con ‘/32‘).

**5.12.1.1 Nota tecnica (integrazione e correzione).** Dal punto di vista pratico, l'uso di bitset porta a una complessità di combinazione tipica  $\mathcal{O}(N \cdot K/W)$  (con  $N$  = numero di regole,  $K$  = numero di campi,  $W$  = ampiezza di parola/word-size), che risulta molto efficiente grazie ad operazioni bitwise vettorializzabili su CPU/GPU/ASIC. Inoltre, la selezione della regola con priorità massima può essere accelerata con tecniche come *de Bruijn* per individuare rapidamente il primo bit significativo. Si noti infine che l'esempio LPM (/24 vs /32) riguarda il *lookup per campo* (indirizzo IP), mentre la scelta finale della regola avviene sulla combinazione dei bitset dei vari campi.

### 5.12.2 Vantaggi e Limiti del Bit Vector Linear Search

Il principale vantaggio di questo approccio è la sua capacità di ridurre il numero di regole da confrontare, migliorando l'efficienza della classificazione. Tuttavia, l'approccio ha alcuni limiti:

- **Espansione della Memoria:** Con l'aumento del numero di regole, la memoria necessaria per memorizzare le bitmask cresce esponenzialmente.
- **Efficacia Limitata in Caso di Ranges Dinamici:** Le regole che utilizzano intervalli dinamici di valori (ad esempio, intervalli di porte) possono complicare la gestione della bitmask.

**5.12.2.1 Precisazione sulla memoria.** In molti design reali l'uso di bitset resta gestibile grazie a compressione, partizionamento e rappresentazioni a parole larghe; più che una crescita “esponenziale”, il costo memoria è determinato dalla cardinalità  $N$ , dal numero di campi  $K$  e dal fattore di compressione/packing adottato.

## 5.13 Contenuto e Prestazioni della Classificazione dei Pacchetti

Per migliorare le prestazioni della classificazione dei pacchetti, è essenziale considerare l'uso di tecniche come la memorizzazione cache, che permette di ridurre i tempi di ricerca. Inoltre, la gestione dinamica delle regole (come l'inserimento automatico di nuove regole per le connessioni UDP) può contribuire a mantenere alta l'efficienza anche in ambienti complessi.

Nel caso di firewall ad alta velocità, è necessario bilanciare la velocità di classificazione con l'efficienza nella gestione della memoria. La combinazione di algoritmi di ricerca rapidi con hardware specializzato (come le CAM) può ridurre significativamente i costi computazionali.

### 5.13.1 Caching 5-tuple: benefici e limiti

Le cache (es. su 5-tuple) possono azzerare il costo medio in presenza di forte località, ma soffrono in scenari avversi (workload *elephant/mice*, scansioni di porta, miss storm) e richiedono politiche di rimpiazzo attente. Inoltre non sostituiscono il classificatore: servono percorsi “cold/miss” efficienti e meccanismi di invalidazione coerenti quando le regole cambiano.

### 5.13.2 TCAM/CAM: pro e contro

Le **TCAM** offrono match ternario in tempo quasi costante, molto adatte a wildcard/prefissi multipli. Pro: latenza prevedibile e parallelismo massivo. Contro: consumo energetico elevato, densità limitata, *rule multiplication* per rappresentare range, costi e integrazione non banali. Spesso si adottano design ibridi: pre-filtrati software + TCAM per hot paths.

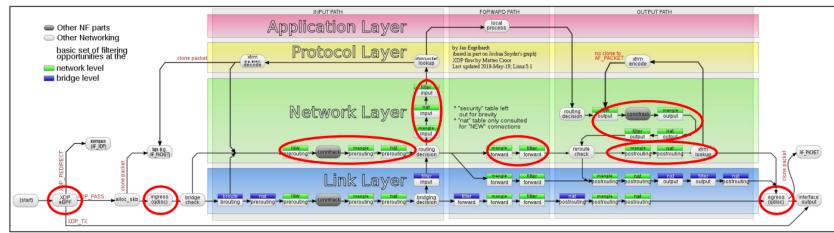
## 5.14 eBPF

eBPF è un framework (pensato in origine per Linux) che consente di scrivere piccoli programmi (tipicamente in C) che vengono compilati in *bytecode* e caricati nel kernel su specifici *hook*. L'esecuzione avviene in una sandbox con garanzie di sicurezza fornite dal **verifier** (controlli su accessi a memoria, loop non limitati, ecc.) e prestazioni vicine al nativo grazie al **JIT** che traduce il bytecode in ISA macchina. L'architettura eBPF è una VM RISC con 11 registri a 64 bit e stack fisso da 512 B.

In pratica consente di estendere funzionalità del sistema operativo *a runtime*, senza patch del kernel, interagendo tramite interfacce stabili: **helper** (API kernel) e **mappe** condivise kernel/user per configurazione e stato. Riduce anche i costi di contesto grazie all'accesso diretto alle risorse del kernel.

### 5.14.1 Hook eBPF (punti di aggancio)

I programmi eBPF sono *event-driven*: vengono eseguiti quando il kernel (o un'applicazione) attraversa un punto di aggancio (*hook*) e consegna un evento al programma. Il codice viene caricato come bytecode e validato dal verifier; l'esecuzione avviene in sandbox nel kernel, con eventuale compilazione JIT. Le interazioni runtime e la conservazione di stato avvengono tramite *mappe* condivise tra kernel e user space. :contentReference[oaicite:0]index=0



### 5.14.2 Focus sui percorsi di rete

**5.14.2.1 XDP** Tra i vari hook, quello di maggiore interesse per il networking a bassa latenza è **XDP**, posizionato molto in basso nel percorso di ricezione (nel driver della network interface card). Questo consente di intervenire precocemente sul pacchetto, riducendo al minimo la dipendenza dagli altri sottomoduli dello stack: si ottengono prestazioni elevate al prezzo di una flessibilità inferiore rispetto ad hook più “alti”. Programmi eBPF per XDP si scrivono in C ristretto, si compilano con LLVM e si caricano nel kernel (syscall `bpf`); il bytecode è verificato e poi JIT-traslatato nell'ISA nativa. :contentReference[oaicite:1]index=1

**5.14.2.2 JIT** JIT sta per Just-In-Time compilation. È una tecnica in cui il codice non viene eseguito interpretandolo riga per riga, né compilato “una volta per tutte” prima di girare (AOT), ma compilato in codice macchina al volo, poco prima dell’esecuzione—o alla prima esecuzione.

**5.14.2.3 TC** Oltre a XDP, gli appunti inquadra l’uso di eBPF come strumento generale per implementare logica di rete nel kernel; in questo contesto, hook a livello **tc** (ingress/egress) permettono classificazione e azioni su pacchetti a uno stadio superiore rispetto a XDP, con maggiore integrazione nel percorso standard dello stack.

**5.14.2.4 NETFILTER e Socket hooks** Nel percorso di rete “classico” il traffico attraversa i punti di aggancio del framework **NETFILTER**, che interfaccia tabelle e catene di filtraggio/-trasformazione; in parallelo, hook sui *socket* abilitano politiche vicino alle applicazioni. Questi punti consentono politiche più ricche e stateful (anche con ausilio di conntrack), a fronte di una maggior latenza rispetto a XDP perché più interni allo stack.

## 5.15 Scrivere e caricare programmi eBPF

### 5.15.0.1 Pipeline operativa

1. **Compilazione** — Scrivi il programma in C ristretto e compila con LLVM/Clang in *bytecode eBPF* destinato alla VM (11 registri a 64 bit, stack 512 B).
2. **Caricamento** — Usa la syscall `bpf()` per trasferire al kernel il bytecode e i metadati (tipo di programma, definizioni delle mappe, licenza).
3. **Verifica** — Il **verifier** controlla sicurezza e correttezza (accessi a memoria, bound dei loop, ecc.); se fallisce, il caricamento viene rifiutato.
4. **JIT** — Se la verifica passa, il kernel può *JIT-compilare* il bytecode in istruzioni native della CPU per ottenere performance prossime al codice nativo.
5. **Attacco all’hook & run-time** — Il programma viene *attaccato* all’hook (es. XDP/TC/-tracepoint) e viene eseguito sugli eventi/pacchetti in arrivo.

### 5.15.0.2 Componenti di supporto

- **Helper functions** — API stabili del kernel per operazioni su socket, checksum, aggiornamenti mappe, manipolazione dei pacchetti e integrazione con lo stack.
- **Mappe (key-value)** — Strutture condivise kernel/user per stato e controllo; tipi comuni: *hash*, *array*, *LPM-trie*, ecc.

# Laboratorio

## 5.16 Laboratorio X: Warm-up con eBPF e XDP

### 5.16.1 Scenario

Obiettivo: prendere confidenza con **XDP** (eBPF a livello driver NIC) implementando:

1. un *Basic PASS/DROP* con logging via `bpf_printk()`;
2. un *Packet Filtering* minimale (IPv4) con i **boundary checks** richiesti dal verifier;
3. un *Map Counter* (contatori per pacchetti/byte) con lettura lato user;
4. una semplice *ACL denylist* basata su mappa hash aggiornata in tempo reale.

### 5.16.2 Topologia del laboratorio

Topologia minima a due nodi: **host A** (dove carichiamo XDP) e **host B** (generatori di pacchetti di test). Collegamento L2 diretto o tramite bridge.

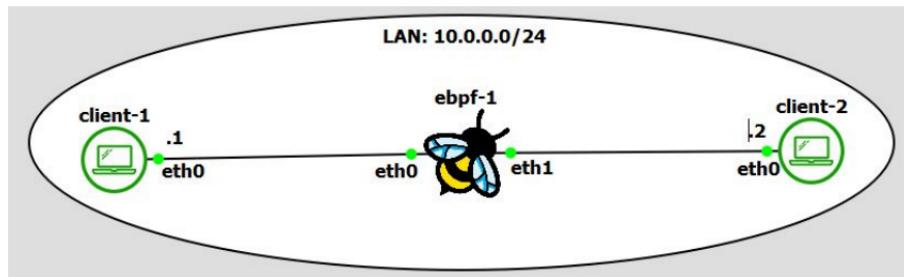


Figura 43: Lab X — Topologia: host A con XDP sul driver di `eth0`; host B genera traffico (ping/Scapy).

*Nota operativa.* Ambiente consigliato: VM o container predisposto per eBPF (`clang/llvm`, `libbpf`, `bpftool`) e privilegi per `bpf()`. Interfaccia target: `eth0` (adatta se differente).

### 5.16.3 Prerequisiti

- Kernel con supporto eBPF/XDP e JIT abilitabile (`/proc/sys/net/core/bpf_jit_enable`);
- toolchain `clang/llvm` e `bpftool`, libreria `libbpf`;
- permessi amministrativi (`root`) sull'host A.

### 5.16.4 Passi del laboratorio (step-by-step)

#### 5.16.4.1 Step 1 — Basic PASS con logging

1. Scrivere un programma XDP minimale che ritorna `XDP_PASS` e fa `bpf_printk("pkt len=`
2. Compilare in bytecode eBPF (`clang/llvm`) e caricare su `eth0`.
3. Verificare i log con `cat /sys/kernel/debug/tracing/trace_pipe`.

#### 5.16.4.2 Step 2 — Switch a DROP e detach

1. Modificare il verdetto a XDP\_DROP e ricaricare il programma.
2. Testare con ping da host B: i pacchetti vengono scartati.
3. Fare *detach*: ip link set dev eth0 xdp off.

#### 5.16.4.3 Step 3 — Packet Filtering (hardcoded) con boundary checks

1. Implementare i controlli di **bound** su header Ethernet e IPv4 (verifier-friendly).
2. Eseguire parsing IPv4 (includere <linux/ip.h>); leggere ip->saddr / ip->protocol.
3. Applicare filtro hardcoded (es. DROP se saddr == X o se proto == ICMP); altrimenti PASS.
4. Caricare, testare con ping e scapy per verificare i rami DROP/PASS.

#### 5.16.4.4 Step 4 — Map Counter (pkt/byte)

1. Definire una **mappa** per contatori (key = u32, ad es. per-CPU o globale; value con rx\_packets, rx\_bytes).
2. In XDP: aggiornare rx\_packets += 1 e rx\_bytes += (data\_end - data) (aggiorni atomici, es. lock\_xadd per byte).
3. Lato user: leggere periodicamente la mappa e stampare pps/mpbs.

#### 5.16.4.5 Step 5 — ACL denylist con mappa hash

1. Creare una mappa denylist (key = u32 indirizzo IPv4, value = u8 segnaposto).
2. In XDP: se lookup(denylist, ip->saddr) ⇒ XDP\_DROP, altrimenti proseguire con la logica (PASS/altro).
3. Aggiornare la denylist a runtime: bpftool map update inserendo/rimuovendo IP senza ricompilare.

### 5.16.5 Esecuzione pratica (comandi essenziali)

Listing 16: Abilitazione JIT, caricamento/isolamento log, attach/detach XDP

```
# Abilita JIT (se disabilitato)
echo 1 > /proc/sys/net/core/bpf_jit_enable

# (Esempio) Build di un programma XDP minimal (clang/libbpf) -> xdp_prog.o
# ... comandi di build specifici al tuo tree (Makefile o libbpf-bootstrap) ...

# Attach su eth0 (loader a scelta: ip link, xdp-loader, o tool custom)
ip link set dev eth0 xdp obj xdp_prog.o sec xdp

# Osserva i log del printk
cat /sys/kernel/debug/tracing/trace_pipe

# Detach
ip link set dev eth0 xdp off
```

Listing 17: Aggiornamento mappa denylist via bpftool

```
# Trovare l'ID della mappa (oppure usare pinning in bpffs)
bpftool map show
# Aggiungere un IP (u32 in endianness corretta, es. 192.0.2.10)
bpftool map update id <MAP_ID> key hex C0 00 02 0A value hex 01
# Rimuovere l'IP
bpftool map delete id <MAP_ID> key hex C0 00 02 0A
```

### 5.16.6 Cosa osservare con lo sniffer / tracing

- Con **XDP\_DROP**: assenza di echo reply; su `trace_pipe` compaiono le linee `bpf_printk()`.
- Con **filter hardcoded**: pacchetti che rispettano la condizione vengono scartati; gli altri passano.
- Con **Map Counter**: incremento coerente di `rx_packets/rx_bytes` in presenza di traffico.
- Con **denylist**: inserendo l'IP in mappa gli ICMP da quel sorgente diventano DROP immediatamente.

#### 5.16.6.1 Troubleshooting

- **Verifier errors**: controllare i boundary checks; evitare dereferenziazioni senza verifica di `data/data_end`.
- **JIT disabilitato**: verificare `/proc/sys/net/core/bpf_jit_enable`.
- **Permessi**: molte operazioni richiedono privilegi root e mount di `bpffs` (`/sys/fs/bpf`).
- **Detach mancato**: usare `ip link set dev eth0 xdp off` prima di ricaricare.

### 5.16.7 Limitazioni pratiche

- XDP è estremamente veloce ma opera *precoce* sul percorso: alcune funzionalità dello stack non sono disponibili.
- Filtri hardcoded servono solo per il warm-up: per policy reali serve gestione dinamica (mappe) e selezione regole robusta.
- Strumenti e offset dipendono dalla versione del kernel e dalla toolchain (compatibilità `libbpf`, pinning, ecc.).

### 5.16.8 Mitigazioni / buone pratiche

- Validare sempre gli header con **boundary checks** prima di accedere ai campi.
- Sfruttare le **mappe** per configurazione a caldo (niente ricompilazioni per cambiare policy).
- Tenere i programmi **piccoli e deterministici** (aiuta il verifier e la latenza).
- Loggare in modo **parsimonioso**: `bpf_printk()` è utile in debug ma impatta le performance.

## Conclusioni

Questo warm-up mostra come **XDP** consenta filtri e contatori ad altissima efficienza *prima* dello stack IP. Dalla base (PASS/DROP, logging) si passa a parsing sicuro, contatori con mappe e **ACL** aggiornabili in tempo reale: fondamenta per un firewall *eBPF/XDP* più evoluto.

## 6 NET\_06 - Secure Protocols and Overlay VPNs

Dopo aver visto che i meccanismi di base dei protocolli IP non forniscono alcuna garanzia di sicurezza — nessuna autenticazione, confidenzialità o integrità dei dati — si rende necessario introdurre protocolli che, a diversi livelli dello stack ISO/OSI, permettano di realizzare comunicazioni sicure. In precedenza abbiamo analizzato meccanismi di sicurezza *perimetrale*, come VLAN, 802.1X, ACL e firewall. Tuttavia, quando la comunicazione esce dal perimetro aziendale e si estende su Internet, tali meccanismi non bastano più. La protezione deve quindi avvenire tramite protocolli di sicurezza *end-to-end*, basati su strumenti crittografici: cifratura, firme digitali, scambio di chiavi e codici di autenticazione dei messaggi (MAC).

Le tre principali famiglie di protocolli di sicurezza di rete sono:

- **SSH (Secure Shell)** a livello applicativo;
- **TLS (Transport Layer Security)** a livello di trasporto;
- **IPsec (Internet Protocol Security)** a livello di rete.

A questi si aggiungono soluzioni di rete private sovrapposte, dette **Overlay VPNs**, che utilizzano tali protocolli per creare tunnel cifrati su reti pubbliche.

### 6.1 Sicurezza a livello applicativo: Secure Shell (SSH)

SSH nasce per sostituire protocolli insicuri come `telnet`, che trasmettevano in chiaro credenziali e dati sensibili. È un protocollo di *remote login* sicuro, ma nel tempo si è evoluto in una suite di strumenti per connessioni protette, trasferimento file (`scp`), inoltro di porte (port forwarding) e proxy TCP.

**6.1.0.1 Architettura e RFC.** SSH è definito da tre RFC principali:

- **RFC 4252:** Transport Layer Protocol – gestisce cifratura, scambio di chiavi e autenticazione del server;
- **RFC 4253:** User Authentication Protocol – autentica l’utente tramite password o chiavi pubbliche;
- **RFC 4254:** Connection Protocol – gestisce canali logici per servizi come shell remota o forwarding.

L’implementazione più diffusa è **OpenSSH**, disponibile su Linux e sistemi Unix-like.

**6.1.0.2 Obiettivi di sicurezza.** SSH fornisce:

- **Autenticazione** dell’host remoto;
- **Confidenzialità** dei dati trasmessi tramite cifratura simmetrica;
- **Integrità** mediante HMAC (Hash-based Message Authentication Code).

**6.1.0.3 Cifratura e handshake.** Durante la fase di handshake, client e server negoziano:

- algoritmi di cifratura simmetrica (es. AES, ChaCha20-Poly1305);
- algoritmi di scambio chiavi (Diffie–Hellman o ECDH);
- algoritmi di firma (RSA, ECDSA, EdDSA);
- funzioni hash (SHA-2, SHA-3).

Una volta scelti gli algoritmi, viene stabilita una *session key* usata per cifrare il traffico del canale.

## SSH handshake at a glance

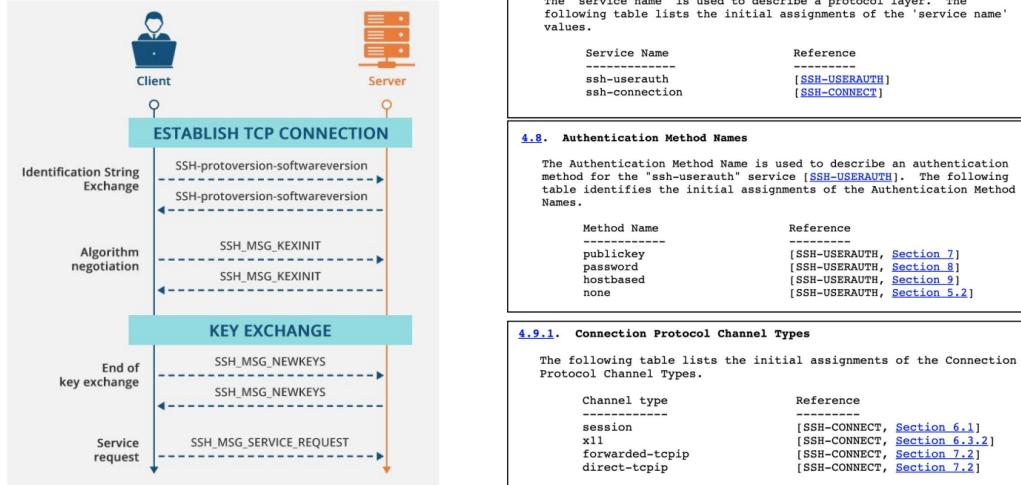


Figura 44: Esempio di handshake SSH

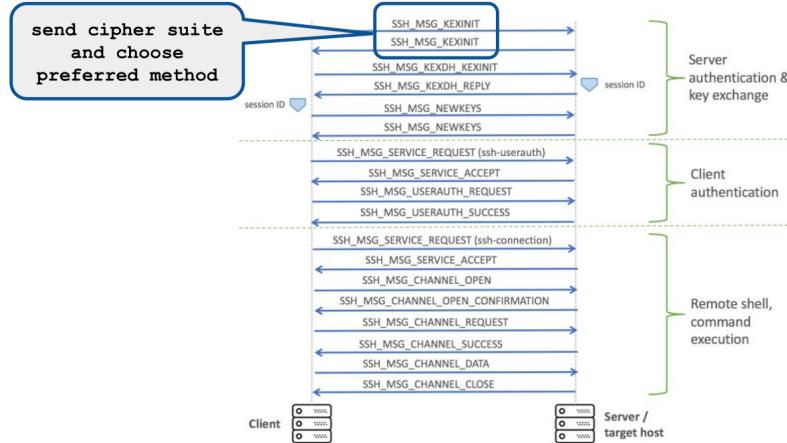


Figura 45: Sequenza di messaggi nel protocollo SSH durante la fase di connessione: negoziazione della *cipher suite*, autenticazione del server e del client, e apertura del canale sicuro per la sessione remota.

**6.1.0.4 Autenticazione del server.** Alla prima connessione, il server invia la propria chiave pubblica: l'utente deve confermare manualmente l'impronta (fingerprint) per legare l'identità del server alla chiave ricevuta. SSH non usa certificati X.509 né una PKI: la fiducia viene costruita progressivamente, salvando la chiave del server nel file `~/.ssh/known_hosts`.

to connect to an SSH server

```
$ ssh username@server
```

```
marlon@demons:~$ ssh pippo@demons.netgroup.uniroma2.it
The authenticity of host 'demons.netgroup.uniroma2.it (160.80.103.172)' can't be
established.
ECDSA key fingerprint is SHA256:sdFXXWU1x9mZjtHkoEz2hbM1XuzqtBTNbqe087fG9rU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'demons.netgroup.uniroma2.it,160.80.103.172' (ECDSA)
to the list of known hosts.
pippo@demons.netgroup.uniroma2.it's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-117-generic x86_64)

Last login: Wed Dec  9 08:05:34 2020 from 160.80.103.172
pippo@demons:~$
```

Figura 46: Esempio di connessione SSH e verifica dell'impronta del server.

**6.1.0.5 Autenticazione del client.** Il client, di default, si autentica con username e password, ma è possibile abilitare l'autenticazione a chiave pubblica. La coppia di chiavi è generata con:

```
$ ssh-keygen -t [rsa|dsa]
```

La chiave pubblica del client (`id_rsa.pub`) deve essere aggiunta sul server al file `~/.ssh/authorized_keys`. Una volta installata, l'utente potrà accedere senza password, basandosi sul meccanismo di *mutual authentication*.

**6.1.0.6 Servizi aggiuntivi.** Oltre all'accesso remoto sicuro, SSH offre numerose funzionalità aggiuntive che lo rendono uno strumento estremamente versatile per l'amministrazione di rete. Tra i principali utilizzi troviamo:

- **Trasferimento di file sicuro** mediante i comandi `scp` o `sftp`, che sostituiscono i protocolli FTP tradizionali trasmettendo i dati cifrati:

```
$ scp [-r] [[user@]host1:]file1 ... [[user@]host2:]file2
```

- **Esecuzione remota di comandi** su un server senza aprire una shell interattiva:

```
$ ssh username@server command
```

- **Inoltro di sessioni grafiche X11**, che consente di eseguire applicazioni grafiche remote:

```
$ ssh -X username@server
```

- **Port forwarding** per creare tunnel TCP cifrati, utili per instradare traffico verso servizi interni:

- *Locale*:

```
$ ssh -L lport:remote_addr:rport username@server
```

- *Remoto*:

```
$ ssh -R rport:local_addr:lport username@server
```

- **Proxy SOCKS5 cifrato**, per instradare connessioni TCP di un'applicazione attraverso il tunnel SSH:

```
$ ssh -ND 9999 username@server
```

- **Montaggio di filesystem remoti** come se fossero directory locali, grazie a `sshfs`:

```
$ sshfs user@host: mountpoint
```

Queste funzionalità dimostrano come SSH non sia soltanto un protocollo di autenticazione remota, ma un vero e proprio framework di sicurezza per la comunicazione cifrata, capace di adattarsi a diversi scenari operativi.

**6.1.0.7 Conclusione.** SSH implementa i propri meccanismi di sicurezza a livello applicativo. Tuttavia, estendere lo stesso approccio ad ogni protocollo applicativo (HTTP, FTP, SIP...) sarebbe inefficiente. È preferibile fornire i servizi di cifratura e autenticazione nei livelli inferiori, rendendoli disponibili come *servizi di sicurezza trasparenti* alle applicazioni. Il problema di ssh è che funziona solo per le applicazioni che lo supportano esplicitamente ed è oneroso. Il seguente diagramma (Figura 47) riassume i principali protocolli di sicurezza a vari livelli dello stack ISO/OSI.

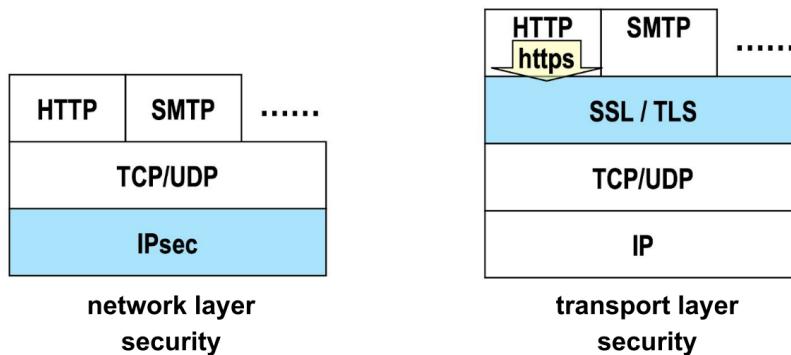


Figura 47: Tipi di sicurezza ai vari livelli

## 6.2 Sicurezza a livello di trasporto: Transport Layer Security (TLS)

Il protocollo **Transport Layer Security (TLS)**, erede diretto di **SSL (Secure Sockets Layer)**, costituisce oggi lo standard per la protezione delle comunicazioni a livello di trasporto. Il suo obiettivo principale è fornire **confidenzialità**, **integrità** e **autenticazione** tra due applicazioni che comunicano su una rete potenzialmente insicura, come Internet. TLS opera sopra il livello di trasporto TCP, fornendo un canale cifrato e autenticato che garantisce protezione end-to-end alle applicazioni, indipendentemente dal protocollo applicativo utilizzato (HTTP, SMTP, IMAP, FTP, ecc.).

**6.2.0.1 Obiettivi di TLS.** Le finalità del protocollo TLS possono essere riassunte in tre fasi principali:

- **Stabilire una sessione sicura (fase di Handshake).** In questa fase le due parti concordano gli algoritmi crittografici da utilizzare, scambiano le chiavi necessarie per la cifratura e si autenticano reciprocamente, solitamente tramite certificati digitali X.509. L’obiettivo è creare un canale cifrato condiviso e verificato, sul quale potranno transitare i dati applicativi.
- **Trasferire dati applicativi in modo protetto.** Una volta stabilita la sessione, la comunicazione avviene in modo privato e integro. La *confidenzialità* è garantita da algoritmi di cifratura simmetrica (ad esempio AES o ChaCha20), mentre l’*integrità* dei dati è assicurata da un codice di autenticazione dei messaggi (**HMAC**, Hash-based Message Authentication Code), che impedisce modifiche non autorizzate ai pacchetti in transito.
- **Un approccio unificato.** A differenza di altri protocolli di sicurezza (come IPsec), che separano il protocollo di negoziazione delle chiavi (IKE) da quello di trasporto dei dati (ESP/AH), TLS integra entrambe le funzioni in un’unica architettura. Ciò semplifica la gestione delle sessioni e consente alle applicazioni di utilizzare direttamente i servizi di sicurezza del livello di trasporto.

**6.2.0.2 Ruolo nello stack di rete.** TLS si colloca tra i livelli 4 e 7 dello stack ISO/OSI: è trasparente per il livello applicativo, che si limita a inviare e ricevere dati sul canale cifrato, e per il livello di trasporto sottostante (TCP), che gestisce solo la consegna affidabile dei segmenti. In tal modo, TLS agisce come un servizio di sicurezza “plug-in” per qualsiasi applicazione che necessiti di autenticazione e riservatezza, rendendo la comunicazione immune da intercettazioni e manipolazioni.

**6.2.0.3 Esempio pratico.** Quando un utente visita un sito web `https://`, il browser instaura una connessione TCP sulla porta 443 e avvia la procedura TLS. Durante l’handshake, client e server negoziano algoritmi, verificano i certificati e generano chiavi di sessione. Da quel momento, ogni messaggio scambiato tra browser e server web è cifrato, autenticato e garantito contro modifiche o intercettazioni.

**6.2.0.4 Storia e porte standard.** In origine, le versioni “sicure” dei protocolli applicativi venivano pubblicate su porte dedicate: HTTP (80) → HTTPS (443), SMTP (25) → SMTPS (465), POP3 (110) → POP3S (995). Questo approccio è stato poi sostituito da meccanismi di *upgrade in-band* (es. STARTTLS), che permettono di negoziare la sicurezza sulla stessa porta.

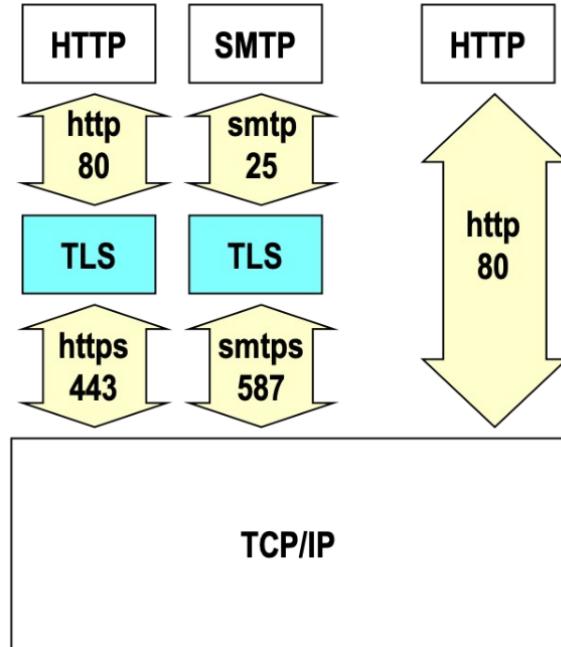


Figura 48: Come viene inserito TLS nello stack di protocolli.

**6.2.0.5 Funzionamento.** TLS stabilisce una **sessione sicura** in due fasi:

1. **Handshake:** negoziazione di algoritmi, scambio di chiavi, autenticazione reciproca;
2. **Data transfer:** cifratura simmetrica e autenticazione tramite MAC (Message Authentication Code).

L'handshake si basa su certificati X.509 e su un'**infrastruttura a chiave pubblica (PKI)** che consente di verificare la legittimità del server e, facoltativamente, del client.

**6.2.0.6 Evoluzione a TLS 1.3.** La versione 1.3 del protocollo introduce miglioramenti sostanziali:

- Rimozione di algoritmi deboli (MD5, RC4, DES, 3DES);
- Uso esclusivo di cifrature AEAD (Authenticated Encryption with Associated Data);
- Perfect Forward Secrecy (PFS) obbligatoria con Diffie–Hellman effimero;
- Handshake più rapido (1-RTT);
- Supporto per 0-RTT data per sessioni ripetute.

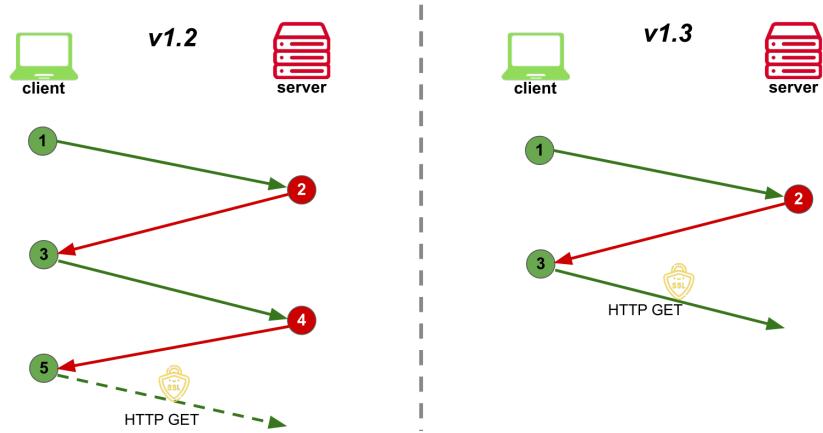


Figura 49: TLS 1.2 vs TLS 1.3: confronto tra le fasi di handshake e i miglioramenti introdotti nella versione 1.3.

#### 6.2.0.7 Attacchi noti.

Nel tempo, TLS ha subito diversi attacchi:

- **Downgrade attacks** (es. FREAK, Logjam): forzano l'uso di versioni o cifrari deboli;
- **Compression attacks** (CRIME, BREACH): sfruttano la compressione per estrarre dati sensibili;
- **Cipher attacks** (BEAST): attacchi ai blocchi CBC di TLS 1.0;
- **Implementation flaws** (Heartbleed): vulnerabilità nel codice OpenSSL.

L'elenco completo degli attacchi noti è riportato in RFC 7457.

# Laboratorio

## 6.3 Laboratorio: HTTPS con Apache2

Il laboratorio ha lo scopo di comprendere il funzionamento di un server HTTPS e la generazione dei certificati necessari per una comunicazione TLS.

**6.3.0.1 1. Creazione della CA.** Si realizza una gerarchia di certificati composta da:

- una **Root CA** autofirmata;
- una **Intermediate CA** firmata dalla root;
- un **certificato server** firmato dall'intermediate.

Le operazioni vengono eseguite con il tool `openssl`:

1. Generazione delle chiavi RSA per la Root CA:

```
openssl genrsa -out root.key 2048
openssl req -new -x509 -days 1460 -key root.key -out root.crt
```

2. Creazione della chiave e CSR per la Intermediate CA e firma da parte della Root CA;

3. Generazione della chiave e CSR del server web, firmata dalla Intermediate CA.

Infine, si concatenano i certificati per creare la *certificate chain*:

```
cat intermediate.crt root.crt > chain.crt
```

**6.3.0.2 2. Configurazione di Apache2.** Nel file `/etc/apache2/sites-available/testssl.conf`:

```
<VirtualHost _default_:443>
    DocumentRoot "/var/www/testssl"
    ServerName www.sito.it
    SSLEngine On
    SSLCertificateFile /path/server.crt
    SSLCertificateKeyFile /path/server.key
    SSLCertificateChainFile /path/chain.crt
</VirtualHost>
```

Si abilita il modulo SSL e il sito:

```
a2enmod ssl
a2ensite testssl
service apache2 restart
```

**6.3.0.3 3. Accesso e verifica.** Visitando `https://www.sito.it`, il browser mostra un avviso poiché la CA non è riconosciuta: importando la Root CA nel browser, la connessione risulterà completamente valida.

**6.3.0.4 Redirect da HTTP a HTTPS.** È possibile implementare il reindirizzamento automatico aggiungendo una regola di rewrite:

```

<VirtualHost _default_:80>
    ServerName www.sito.it
    RewriteEngine On
    RewriteRule (.*) https:// %{HTTP_HOST} %{REQUEST_URI}
</VirtualHost>

```

## 6.4 Attacchi di downgrade e contromisure

Un attacco comune è l'**HTTPS Downgrade Attack**, in cui l'attaccante si interpone fra client e server (Man-in-the-Middle) e forza la vittima a usare HTTP invece di HTTPS. Storicamente, molti siti erano misti (HTTP per la home e HTTPS per l'autenticazione). Oggi il rischio è mitigato da meccanismi come **HSTS (HTTP Strict Transport Security)**.

### 6.4.0.1 HSTS.

Il server invia un header HTTP:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Questo informa il browser che le connessioni future verso il dominio dovranno sempre usare HTTPS. HSTS protegge quindi da downgrade futuri, ma non dal primo accesso in chiaro. Alcuni browser integrano liste di domini pre-caricate (“HSTS preload list”) per evitare anche il rischio iniziale.

## 6.5 Overlay VPNs e OpenVPN

Le **Virtual Private Networks (VPN)** permettono di creare reti private sovrapposte a reti pubbliche, come Internet, garantendo cifratura e autenticazione del traffico. Le VPN si dividono in due categorie:

- **Intra-AS VPNs:** realizzate all'interno di un singolo dominio amministrativo (es. MPLS VPN);
- **Overlay VPNs:** costruite sopra una rete pubblica non fidata, tipicamente tramite tunneling cifrato.

**6.5.0.1 Concetto di tunneling.** Nel tunneling, un pacchetto IP “interno” viene incapsulato come payload in un altro pacchetto IP “esterno”. Il pacchetto esterno contiene gli indirizzi pubblici dei nodi del tunnel, mentre quello interno trasporta gli indirizzi della rete privata.

### 6.5.1 OpenVPN

**OpenVPN** è una soluzione open source per la creazione di VPN a livello utente, basata su TLS e su un modello client/server. Supporta due modalità operative:

- **TUN mode:** tunnel punto-punto di livello 3 (IP);
- **TAP mode:** tunnel di livello 2, utile per trasmettere frame Ethernet e traffico broadcast.

**6.5.1.1 Architettura.** OpenVPN utilizza due canali:

- un **canale di controllo**, che usa TLS per negoziare chiavi e autenticazione;
- un **canale dati**, cifrato simmetricamente e incapsulato in UDP o TCP.

Il meccanismo di autenticazione si basa su certificati X.509 generati da una CA locale (facilmente realizzabile con **easy-rsa**).

### 6.5.1.2 Configurazione di base.

- Server:

```
port 1194
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh.pem
server 10.8.0.0 255.255.255.0
```

- Client:

```
client
dev tun
proto udp
remote vpn.server.it 1194
ca ca.crt
cert client.crt
key client.key
```

**6.5.1.3 Routing e NAT.** Per permettere ai client di raggiungere la rete interna dietro il server VPN, è necessario abilitare il **IP forwarding** e applicare una regola di masquerade:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
```

**6.5.1.4 Autenticazione e cifratura.** Durante l'handshake TLS, server e client si autenticano con i propri certificati e negoziano i parametri di sicurezza (algoritmi di cifratura, HMAC e DH). OpenVPN può usare sia chiavi statiche (*pre-shared*) che dinamiche (via TLS), ma la seconda opzione garantisce PFS e sicurezza end-to-end.

### 6.5.1.5 Vantaggi.

- Elevata compatibilità (lavora in user space);
- Supporto a NAT e firewall;
- Configurazione flessibile e integrabile con iptables;
- Sicurezza basata su standard TLS.

## 6.6 Conclusione

I protocolli di sicurezza a vari livelli dello stack cooperano per garantire comunicazioni affidabili su reti non sicure. SSH protegge accessi e sessioni remote, TLS assicura la riservatezza delle applicazioni Internet e IPsec offre protezione nativa per il traffico IP. Le Overlay VPNs, come OpenVPN, uniscono tali principi in un contesto pratico, creando reti private virtuali su infrastrutture pubbliche. Questi meccanismi costituiscono i mattoni fondamentali della **difesa di rete moderna**, integrando autenticazione, cifratura e controllo dell'integrità in ogni fase della comunicazione.

## 7 SYS\_01 — Introduzione ai Security Frameworks

### 7.1 Quadro generale e obiettivo

La sicurezza informatica non è uno stato binario ma un processo continuo: si riduce il *costo d'attacco* mantenendo usabilità e disponibilità accettabili. Ogni strato (hardware, firmware, kernel, librerie, applicazioni) introduce superfici d'attacco e dipendenze; perciò servono principi comuni e *framework* che rendano il miglioramento *ripetibile*, *misurabile* e *auditabile*. L'obiettivo del modulo è collegare i **principi fondativi** (autenticità, autorizzazione, accountability) con i **framework operativi** (FIPS 200, CIS Controls, ISO/IEC 27000) e mostrare come si traducano in piani d'azione.

### 7.2 Principi fondamentali

#### 7.2.0.1 Confidenzialità

##### Che cos'è:

È la proprietà per cui l'informazione è accessibile solo ai soggetti autorizzati, impedendo letture, copie o divulgazioni non consentite.

##### Perché serve:

Senza confidenzialità, qualsiasi misura di sicurezza perde significato: un'informazione rubata o intercettata può compromettere l'intero sistema.

##### Come si ottiene:

Attraverso cifratura in transito (TLS, SSH) e a riposo (Full Disk Encryption, database encryption), una gestione sicura delle chiavi (KMS, HSM), segmentazione di rete, principio del need-to-know e riduzione dei dati trattati al minimo necessario.

I Modelli sono: *simmetrica* (AES: veloce, chiave condivisa) e *asimmetrica* (RSA/EC: coppie pub/priv, abilita anche firme). Sicurezza “computazionale”: l'obiettivo è rendere impraticabile la ricerca esaustiva della chiave. La gestione delle chiavi (KMS, rotazione, *separation of duties*) è parte integrante del controllo.

##### Trappole comuni:

Chiavi salvate in chiaro o condivise tra utenti, backup e log non cifrati, assenza di cifratura interna su link “fidati”.

#### 7.2.0.2 Integrità

##### Che cos'è:

È la garanzia che i dati o i componenti software non siano stati modificati in modo non autorizzato, oppure che ogni modifica sia tracciabile e verificabile.

##### Come si ottiene:

Con hash e HMAC, firme digitali, controlli di modifica (change management e principi 4-eyes), log e backup immutabili (WORM), validazioni in pipeline CI/CD.

##### Esempi:

Checksum end-to-end, code signing, manifest firmati, blockchain privata per log e supply-chain verification.

##### Attenzioni:

CRC o hash non firmati non offrono sicurezza reale; le chiavi di firma vanno custodite e ruotate; occorre distinguere tra collisione e preimage per valutare la robustezza di un algoritmo.

#### 7.2.0.3 Disponibilità

##### Che cos'è:

È la capacità di mantenere servizi e dati accessibili con prestazioni accettabili, anche in presenza

di guasti o attacchi.

**Come si ottiene:**

Con ridondanza ( $N+1$ , active-active), strategie di degrado controllato (graceful degradation), meccanismi di protezione come circuit breaker, capacity planning, autoscaling e piani di continuità (DR/BCP).

**Minacce tipiche:**

Attacchi DoS/DDoS, saturazione delle risorse, errori di configurazione, catene di dipendenze non ridondante, aggiornamenti mal pianificati.

#### 7.2.0.4 Autenticità e Autorizzazione

**Autenticità — che cos’è:**

È la garanzia che l’identità dichiarata da un soggetto corrisponda effettivamente a chi afferma di essere.

**Autenticità — come si ottiene:**

Tramite meccanismi di autenticazione a più fattori (MFA), gestione sicura delle credenziali, certificati digitali, attestazione hardware e protocolli di handshake sicuro.

**Autorizzazione — che cos’è:**

È il processo di assegnare o negare risorse a un soggetto autenticato in base al suo ruolo, contesto o policy.

**Autorizzazione — come si ottiene:**

Applicando il principio del least privilege, modelli RBAC (Role-Based Access Control) o ABAC (Attribute-Based), segregazione dei doveri, accessi JIT/JEA e revisioni periodiche dei permessi.

**Pattern utili:**

SSO centralizzato, token scadibili, sessioni brevi, meccanismi di break-glass tracciato per accessi di emergenza.

#### 7.2.0.5 Accountability

**Che cos’è:**

È la possibilità di attribuire in modo affidabile ogni azione compiuta in un sistema a un soggetto identificabile.

**Come si ottiene:**

Attraverso logging centralizzato e immutabile (SIEM), sincronizzazione oraria precisa (NTP/PTP), integrità dei log (HMAC, firme o catene di hash), correlation ID end-to-end e una catena di custodia documentata per la forensica.

**Attenzioni:**

Bilanciare privacy e obblighi di tracciabilità; proteggere sia il canale di logging sia il repository; stabilire tempi di retention coerenti con GDPR e compliance interna.

#### 7.2.0.6 Dipendibilità

**Che cos’è:**

È la qualità complessiva di un sistema nel fornire un servizio affidabile, sicuro, mantenibile e resiliente nel tempo.

**Come si ottiene:**

Attraverso osservabilità (metriche, log, tracing), deploy sicuri (feature flags, canary release, rollback), chaos engineering, e post-incident review per il miglioramento continuo.

**Connessioni:**

La CIA tutela dati e servizi, autenticità e autorizzazione controllano chi accede, accountability ne permette la verifica, e la dipendibilità orchestra tutto il ciclo di vita — dalla progettazione, all’esercizio, fino alla risposta agli incidenti.

## 7.3 Perché servono i Security Frameworks

La sicurezza non può più basarsi solo su “buone pratiche” isolate: i sistemi moderni sono troppo complessi, distribuiti e interdipendenti. Senza una struttura comune, ogni reparto o fornitore valuterebbe i rischi in modo diverso, rendendo impossibile il confronto e la gestione coerente della sicurezza.

I **Security Frameworks** servono proprio a questo: forniscono un linguaggio condiviso, un insieme ordinato di controlli riusabili e una base di confronto per misurare maturità e copertura. Ogni framework traduce i principi astratti (come confidenzialità, integrità e disponibilità) in **controlli pratici**, assegnando priorità e responsabilità operative. Inoltre, introducono un metodo sistematico di miglioramento continuo basato sul ciclo *Plan–Do–Check–Act* (PDCA): pianificare i controlli, attuarli, verificarne l’efficacia e correggere le lacune.

In sintesi, i framework rendono la sicurezza:

- **misurabile**, grazie a controlli e metriche standard;
- **ripetibile**, perché le procedure possono essere replicate e auditabili;
- **comunicabile**, poiché tecnologia, processi e persone usano la stessa terminologia;
- **migliorabile**, grazie al monitoraggio continuo del rischio residuo.

## 7.4 Tre famiglie a confronto

### 7.4.1 FIPS 200 — Requisiti minimi per i sistemi federali

Il **FIPS 200** (Federal Information Processing Standard) definisce i requisiti minimi di sicurezza per i sistemi informativi delle agenzie federali statunitensi. È un approccio prescrittivo e normativo: stabilisce quali controlli devono essere presenti e come verificarne l’attuazione.

Le principali famiglie di controllo comprendono: *Access Control, Audit and Accountability, Configuration Management, Contingency Planning, Identification and Authentication, Incident Response, Maintenance, Risk Assessment* e molte altre.

Ogni organizzazione deve:

- valutare periodicamente i controlli implementati;
- pianificare e attuare azioni correttive;
- ottenere l’autorizzazione all’operatività dei propri sistemi (*Authorization to Operate, ATO*);
- mantenere un monitoraggio continuo dell’efficacia delle misure.

Il FIPS 200 è quindi una **baseline di conformità** utile dove esistono vincoli regolatori o contrattuali (es. ambienti governativi o fornitori di enti pubblici). Fornisce un modello forte in termini di *assurance* e tracciabilità dei controlli, ma poco flessibile in contesti dinamici o non federali.

### 7.4.2 CIS Critical Security Controls — L’igiene prioritaria

I **CIS Controls** (Center for Internet Security) rappresentano un insieme pratico e priorizzato di buone pratiche operative, pensato per essere attuabile rapidamente anche da organizzazioni non grandi.

I controlli sono 20 (raggruppati in famiglie come *Inventory and Control, Secure Configuration, Continuous Vulnerability Management, Controlled Use of Administrative Privileges, Malware Defense, Incident Response...*), ciascuno articolato in sotto-controlli concreti.

I primi cinque sono considerati la “**cyber hygiene di base**”, cioè le difese fondamentali che riducono fino all’85% delle minacce comuni:

1. inventario di dispositivi autorizzati e non autorizzati;
2. inventario del software installato;
3. configurazioni sicure di host e apparati;
4. gestione continua delle vulnerabilità e delle patch;
5. uso controllato dei privilegi amministrativi.

La forza dei CIS Controls sta nella loro concretezza: sono pensati per rispondere alla domanda “*da dove cominciare domani mattina?*”. Sono verificabili, misurabili e fortemente orientati all’operatività quotidiana.

Per questo motivo vengono spesso adottati come base operativa su cui innestare framework più ampi (ISO, NIST, ecc.), fungendo da **checklist pragmatica** per il miglioramento rapido della postura di sicurezza.

#### 7.4.3 ISO/IEC 27000 — ISMS e gestione del rischio

La famiglia **ISO/IEC 27000** fornisce un modello internazionale per la gestione della sicurezza delle informazioni basato sull’approccio **ISMS** (*Information Security Management System*).

Il cuore della famiglia è la **ISO/IEC 27001**, che definisce i requisiti per stabilire, mantenere e migliorare un ISMS certificabile. La **27002** fornisce linee guida e buone pratiche per l’attuazione dei controlli, mentre la **27005** approfondisce la gestione del rischio.

Esistono inoltre estensioni verticali per contesti specifici:

- **27017** e **27018** per la sicurezza e privacy nel cloud;
- **27019** per i sistemi industriali (ICS/SCADA);
- **27011** (ITU-T X.1051) per operatori telco.

Il modello ISO si basa su un approccio **risk-based** e adattabile: ogni organizzazione identifica e valuta i propri rischi, seleziona i controlli più adatti (dall’*Annex A*) e li integra nei processi aziendali secondo il ciclo PDCA.

L’obiettivo non è la “sicurezza assoluta”, ma la **gestione consapevole del rischio**, bilanciando protezione, costi e conformità. Il valore aggiunto della serie ISO 27000 è la sua scalabilità e certificabilità: consente audit esterni, comparabilità e riconoscimento internazionale, fungendo da ponte tra *compliance* e *governance*.

#### Sintesi comparativa

- **FIPS 200**: prescrittivo, orientato alla conformità e all’accreditamento dei sistemi federali. Ottimo per la *baseline* normativa, poco flessibile fuori da contesti regolati.
- **CIS Controls**: operativo e immediato, fornisce priorità chiare e controlli attuabili. Ideale per migliorare rapidamente l’igiene di sicurezza.
- **ISO/IEC 27000**: gestionale e certificabile, integra sicurezza, rischio e governance. Indicato per aziende che mirano a maturità e riconoscimento formale.

Molte organizzazioni combinano i tre approcci: i *CIS Controls* come base operativa, la *ISO 27001* come cornice di governance, e il *FIPS 200* come riferimento di conformità in ambiti regolati.

## 8 SYS\_02 — Hardware Primer

### 8.1 Moore's Law e l'evoluzione del calcolo

Nel 1965 Gordon Moore osservò che il numero di transistor per unità di area nei circuiti integrati tendeva a raddoppiare ogni due anni. Questa legge empirica — *Moore's Law* — ha guidato per decenni l'intera industria dei semiconduttori: più transistor significano clock più alti, parallelismo maggiore e costi inferiori per unità di potenza computazionale. Per quasi quarant'anni il trend è stato rispettato, permettendo un'esplosione di potenza e miniaturizzazione, ma a partire dai primi anni 2000 sono emersi vincoli fisici e termici noti come **power wall**: oltre i  $\sim 130$  W di dissipazione termica per CPU, i guadagni marginali diventano insostenibili.

### 8.2 Fine della crescita lineare e parallelismo

La stagnazione delle frequenze e la saturazione del numero di transistor utili (come mostrato da Hennessy e Patterson nella Turing Lecture 2018) hanno spinto verso architetture parallele: più core, pipeline più profonde e ottimizzazioni speculative. Amdahl, già nel 1967, formalizzò il limite teorico di tale approccio: il guadagno complessivo  $S_{latency}$  da un'ottimizzazione parziale dipende dalla frazione  $p$  di codice migliorato e dal relativo speedup  $s$ , secondo

$$S_{latency} = \frac{1}{(1-p) + \frac{p}{s}}$$

da cui risulta chiaro che, se solo una piccola parte del programma è parallelizzabile, il beneficio totale resta limitato.

### 8.3 Pipeline e architettura superscalare

Per superare i limiti dei primi core multicycle (lenti ma complessi), negli anni '80 si introdusse il **pipelining**: un'istruzione viene divisa in più fasi temporali (fetch, decode, execute, write-back), permettendo l'esecuzione "a catena". Idealmente il CPI (Cycles Per Instruction) diventa 1, ma solo se non vi sono conflitti o branch. Negli anni '90 arrivarono le **architetture superscalari**, capaci di eseguire più istruzioni contemporaneamente grazie a unità funzionali replicate e scheduling dinamico a runtime: il processore decide in tempo reale quali istruzioni emettere in parallelo, massimizzando il throughput.

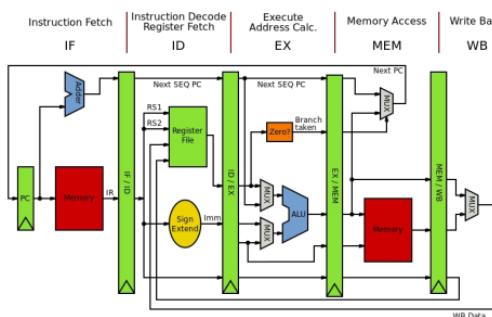


Figura 50: Architettura pipeline MIPS

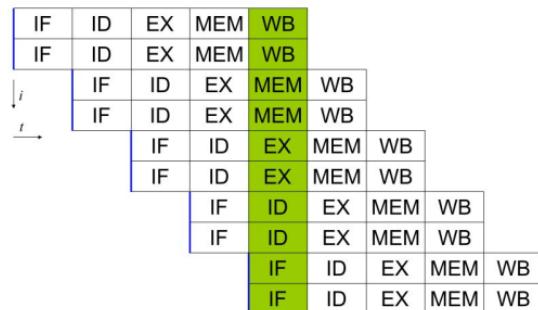


Figura 51: Esecuzione parallela di più istruzioni sulla stessa CPU

### 8.4 Branch Prediction: principi e strategie

La predizione dei salti è una delle ottimizzazioni più importanti per mantenere il *throughput* della pipeline: ogni salto condizionale interrompe il flusso sequenziale di istruzioni e può costringere a svuotare (flush) la pipeline se la direzione effettiva non è quella prevista. Con pipeline sempre

più profonde e architetture superscalari che eseguono molte istruzioni per ciclo, il costo di una **misprediction** può arrivare a decine di cicli.

**8.4.0.1 Perché serve una predizione** Quando la CPU incontra un’istruzione di salto (**branch**), non può sapere immediatamente se il salto sarà preso (*taken*) o meno (*not taken*) finché il confronto logico non viene risolto nello stadio di esecuzione. Attendere significherebbe fermare la pipeline: per questo, la CPU **indovina** il risultato e continua a caricare istruzioni dal percorso ipotizzato.

#### 8.4.0.2 Tipi di predizione

- **Statica:** la decisione è determinata a compile-time. È semplice e non richiede hardware dedicato: ad esempio, si assume che i salti in avanti (come in un **if**) non siano presi, mentre quelli all’indietro (come nei loop) lo siano. Alcuni compilatori o kernel possono anche inserire “hint” (es. attributi **likely/unlikely** o prefissi nel microcodice) per guidare il processore.
- **Dinamica:** la CPU apprende dal comportamento passato del programma, usando **storia dei branch** e piccole strutture hardware che registrano se l’ultimo salto da un certo indirizzo è stato preso o meno. La predizione viene aggiornata a runtime, quindi si adatta a comportamenti ricorrenti del software.

**8.4.0.3 Contatore a 2 bit saturante** Il modello più classico di predittore dinamico è il **2-bit saturating counter**: ogni branch ha un contatore a due stati fortemente orientati (00 = “fortemente non preso”, 11 = “fortemente preso”). Il contatore si incrementa o decrementa in base all’esito reale, ma non passa da un estremo all’altro in un solo errore, riducendo così le oscillazioni casuali. Questo approccio funziona bene per branch regolari, ma fallisce con strutture di controllo più complesse come **loop annidati**, dove il contesto del branch esterno viene continuamente “sporcato” da quello interno.

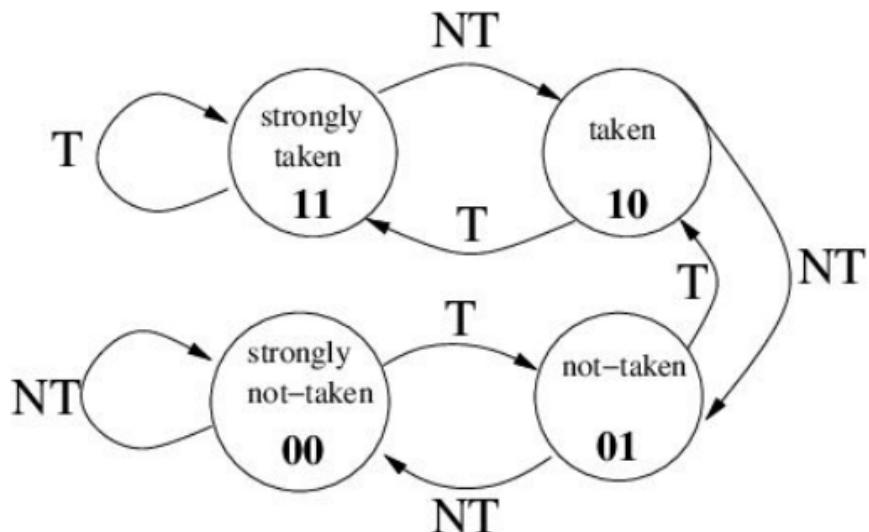


Figura 52: 2 bit saturating counter

**8.4.0.4 Predittori correlati e multilivello** Per migliorare l’accuratezza, i processori moderni impiegano predittori che tengono conto non solo della storia di un singolo branch, ma anche delle correlazioni tra branch consecutivi:

- **Correlated (two-level) predictor:** mantiene una **Global History Register (GHR)** che codifica l'esito degli ultimi  $m$  branch (presa = 1, non presa = 0). Il valore di questo registro viene usato come indice in una **Pattern History Table (PHT)**, che contiene i contatori a 2 bit. In questo modo la CPU “riconosce” sequenze ricorrenti di salti.

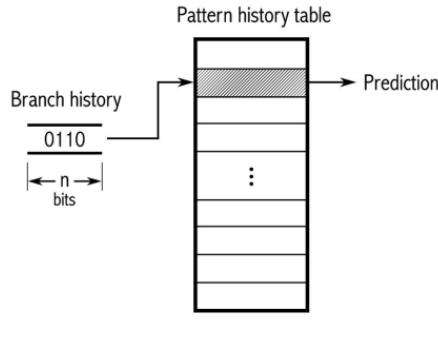


Figura 53: Pattern History Table

- **Hybrid o Tournament predictor:** combina più predittori (ad esempio uno locale e uno globale) e sceglie dinamicamente quale usare in base alla precisione recente. Una piccola tabella di selezione con contatori a 2 bit tiene traccia di quale predittore ha vinto più spesso per un certo branch.

Un esempio celebre è quello del processore **DEC Alpha 21264**, che impiegava un algoritmo di predizione “tournament” con meno dell’1% di mispredizioni totali, occupando appena il 2% dell’area del chip.

**8.4.0.5 Ottimizzazioni hardware** Oltre ai predittori veri e propri, esistono altre strutture dedicate a ridurre il tempo di risoluzione del salto:

- **Branch Target Buffer (BTB):** una piccola cache che associa a ciascun indirizzo di branch il suo target previsto. Se il salto è preddetto “taken”, il BTB fornisce immediatamente il nuovo indirizzo di fetch, senza attendere l’ALU.
- **Return Address Stack (RAS):** uno stack hardware che salva l’indirizzo di ritorno delle chiamate a funzione (`call/return`). Poiché la maggior parte dei salti indiretti sono proprio ritorni da funzione, questo migliora drasticamente la predizione nei linguaggi ad oggetti e nei programmi modulari.
- **Fetch both targets:** alcune architetture ad alte prestazioni (es. mainframe IBM) prelevano simultaneamente le istruzioni da entrambi i possibili percorsi (taken e not-taken), per poi scartare quello errato. È una soluzione costosa in termini di banda e cache, ma riduce quasi a zero la penalità di mispredizione.

**8.4.0.6 Importanza crescente della predizione** Con pipeline sempre più profonde e front-end superscalari, ogni ciclo perso per un salto errato può comportare la perdita di decine di istruzioni. La branch prediction moderna è quindi un componente centrale della microarchitettura: la precisione di questi predittori incide direttamente sul consumo energetico, sull’IPC (*Instructions Per Cycle*) e sulla sicurezza — come dimostrato dalle vulnerabilità speculative (*Spectre, Meltdown*) che sfruttano proprio l’esecuzione speculativa e il caching dei risultati predetti.

## 8.5 Simultaneous Multithreading (SMT)

Con la stagnazione del clock e i limiti di dissipazione, si è puntato a sfruttare meglio le risorse interne del core. Lo **SMT** consente a un singolo core fisico di apparire come più core logici: ogni thread mantiene il proprio stato architettonico (registri, flag, PC), ma condivide le unità funzionali. In caso di stallo di un thread (es. cache miss), un altro può utilizzare immediatamente le pipeline inutilizzate. Intel implementa SMT come **Hyper-Threading**: due thread logici per core, con alternanza in caso di contesa.

## 8.6 Pipeline interna nei processori Intel Xeon

La pipeline dei Xeon è divisa in due macro-stadi:

- **Front End:** decodifica le istruzioni CISC x86 in micro-operazioni RISC ( $\muops$ ) tramite un microcode ROM e le memorizza nella **Trace Cache (TC)**. La TC conserva le  $\muops$  nel loro ordine di esecuzione effettiva, evitando di ridurre in continuazione istruzioni già viste.
- **Out-of-Order Engine:** gestisce l'esecuzione delle micro-operazioni ( $\muops$ ) fuori ordine rispetto al flusso originale, massimizzando il parallelismo interno. Dopo il  **$\muop$  Queue**, le istruzioni entrano nello stadio di **Register Rename**, dove i registri logici vengono mappati su registri fisici per eliminare conflitti (false dipendenze). Le  $\muops$  vengono poi inserite in una **coda di scheduling** che decide, a runtime, quali operazioni emettere in base alla disponibilità delle risorse e dei dati. Una volta pronti gli operandi, il blocco **Execute** effettua le operazioni reali, accedendo alla **L1 Data Cache** per letture e scritture. I risultati vengono scritti nei registri fisici e infine ritirati in ordine logico dal **Reorder Buffer (ROB)**, che ricostruisce la sequenza originale del programma garantendo coerenza architettonica. La figura 54 mostra la sequenza interna del motore Out-of-Order nei processori Xeon.

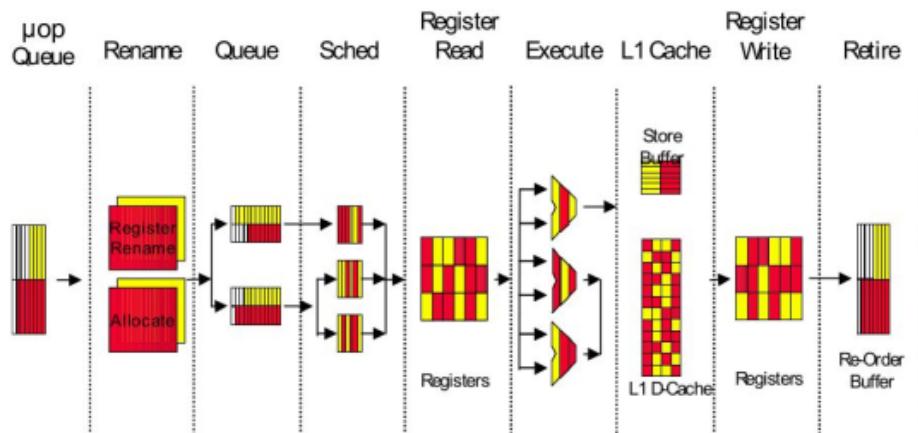


Figura 54: Pipeline interna Out-of-Order nei processori Xeon: le  $\muops$  attraversano gli stadi di rinomina, scheduling, esecuzione e *retirement* nel **Reorder Buffer**, che garantisce il commit in ordine architettonico.

Un *Trace Cache hit* consente al core di servire  $\muops$  senza passare per la decodifica, mentre un miss richiede fetch e decode da L2 (ovvero il livello 2 della cache). Questo modello riduce il collo

di bottiglia di decodifica tipico dell'ISA x86, ottenendo parallelismo interno senza moltiplicare il numero di core.

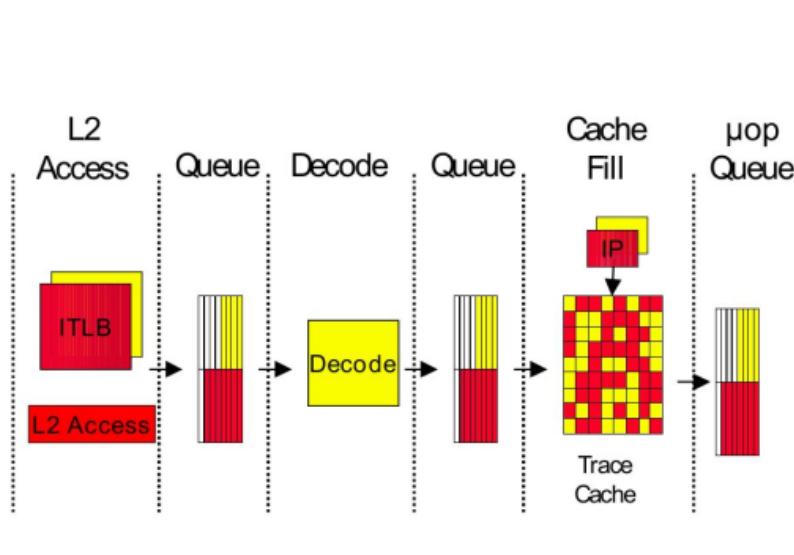


Figura 55: Flusso del **front-end** nei processori Intel: le istruzioni x86 vengono prelevate dalla cache L2, decodificate in micro-operazioni (*μops*) e memorizzate nella **Trace Cache**. In caso di *hit*, le *μops* vengono fornite direttamente alla pipeline senza ripetere il ciclo di fetch e decode.

## 8.7 Gerarchia di memoria

La distanza prestazionale tra CPU e memoria principale (DRAM) cresce di circa il 50% l'anno: per colmare il gap si adotta una **memoria gerarchica** (L1, L2, L3, RAM, disco). Ogni livello inferiore è più capiente ma più lento e meno costoso per bit. Il processore interagisce direttamente solo con L1: un **cache hit** consente accesso immediato, mentre un **miss** causa richieste ricorsive ai livelli successivi fino alla RAM (da L1 vado a L2, se c'è hit copio dato in L1, altrimenti vado in L3 e vedo hit/miss e così via).

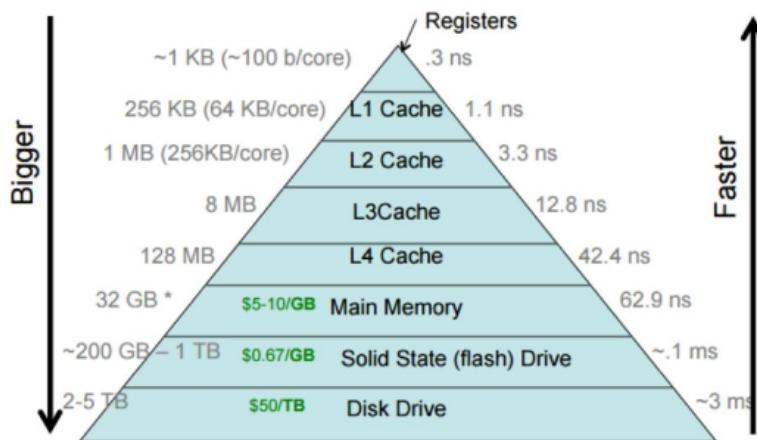


Figura 56: Schema piramidale della gerarchia di memoria

**8.7.0.1 Inclusività e mappatura** **Inclusività tra livelli di cache.** Nelle gerarchie moderne (L1, L2, L3) ogni livello può decidere se mantenere o meno una copia dei dati già

presenti nei livelli inferiori. Si distinguono tre politiche principali:

- **Inclusive:** tutti i blocchi presenti in L1 devono esistere anche in L2 (e, se presente, in L3). Ciò semplifica la **coerenza**: se un core invalida un dato in L3, si sa che anche le sue copie nelle cache inferiori devono essere invalidate. Tuttavia, l'inclusività riduce la capacità effettiva complessiva (L2 “spreca” spazio per dati già duplicati).
- **Exclusive:** i blocchi si trovano in un solo livello alla volta — L1 e L2 contengono set disgiunti. Aumenta la capacità totale utile, ma complica la gestione: un miss in L1 può richiedere di “spostare” un blocco da L2 a L1, generando più traffico interno.
- **Non-inclusive / Non-exclusive (NINE):** via di mezzo più flessibile; non impone regole rigide di duplicazione ma lascia la scelta all’hardware. È usata nei processori recenti (es. AMD Ryzen) per bilanciare prestazioni e flessibilità.

**Politiche di mappatura.** Quando la CPU deve memorizzare un blocco di memoria nella cache, serve una regola per stabilire in quale linea (entry) posizionarlo. Questa scelta influenza direttamente le prestazioni e il tasso di *miss*.

- **Direct-mapped cache.**

Ogni indirizzo di memoria corrisponde a una sola posizione possibile in cache, determinata da alcune bit dell’indirizzo (funzione modulo del numero di linee). È la struttura più semplice e veloce: un solo comparatore e accesso deterministico. Tuttavia, se due blocchi diversi mappano sulla stessa linea, si generano **conflitti ripetuti** (cache thrashing). È adatta a cache piccole e a livello L1, dove si privilegia la latenza minima.

- *Pro:* accesso veloce, logica semplice, basso consumo.
- *Contro:* tasso di miss elevato in presenza di conflitti.

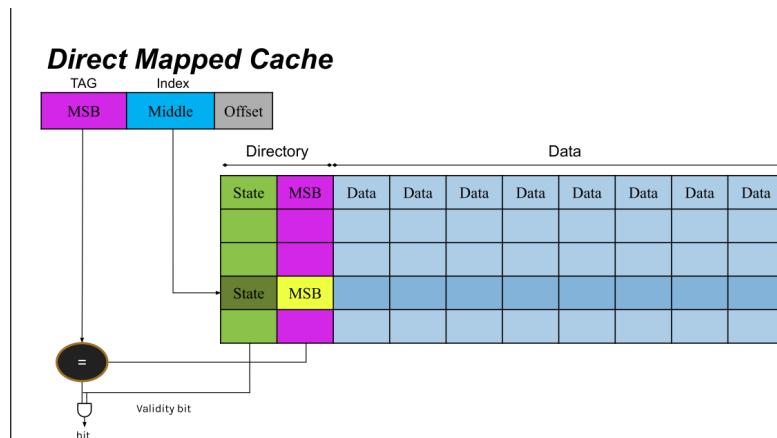


Figura 57: Direct mapped cache

- **Fully associative cache.**

Un blocco di memoria può essere collocato in *qualsiasi* linea della cache. L’indirizzo viene confrontato con tutte le etichette (tag) in parallelo per verificare un hit. Offre la massima flessibilità e il minor numero di conflitti, ma richiede **comparazioni parallele costose** in termini di area e potenza. È usata tipicamente per cache molto piccole o buffer speciali (es. TLB).

- *Pro:* minimi miss da conflitto, utilizzo ottimale dello spazio.
- *Contro:* costosa e lenta da scalare, elevato consumo energetico.

### Fully Associative Cache: read access

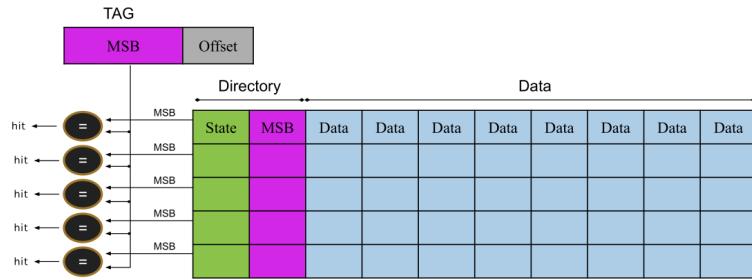


Figura 58: Fully associative cache

- **Set-associative cache.**

È il compromesso più comune. La cache è divisa in gruppi (set) di  $n$  linee ciascuno; un indirizzo mappa su un set specifico (tramite parte dell'indirizzo), ma può occupare *una qualsiasi* delle  $n$  linee di quel set. Il grado di associazione (2-way, 4-way, 8-way, ...) determina quanti blocchi possono coesistere nello stesso set. L'hardware esegue fino a  $n$  confronti paralleli, mantenendo un buon bilancio tra velocità e flessibilità.

- *Pro:* riduce fortemente i miss da conflitto rispetto al direct-mapped, con costo contenuto.
- *Contro:* leggermente più lenta e più complessa; aumento di consumo con l'associatività.

### 4-way Set Associative Cache: read access

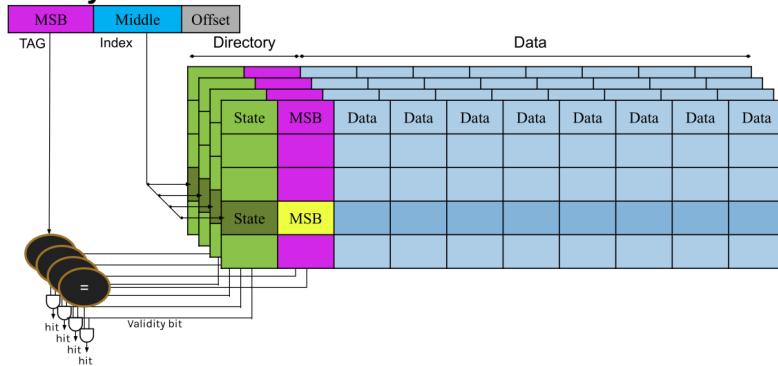


Figura 59: 4way set Associative cache

### Sintesi pratica.

- L1 è spesso **direct-mapped o 2-way** per minimizzare la latenza.
- L2 e L3 sono di solito **8-way o più** per massimizzare la capacità.
- I processori moderni (Intel/AMD) adottano strutture **inclusive o NINE** per semplificare la coerenza cache tra core.

**8.7.0.2 Sostituzione ed aggiornamento** Le politiche di rimpiazzamento (cambio dei blocchi di cache) determinano quale blocco deve essere espulso quando lo spazio di memoria nella cache è pieno. Le politiche più comuni sono:

- **LRU (Least Recently Used):** il blocco meno recentemente utilizzato viene espulso. Questo approccio è basato sull'assunzione che i blocchi recentemente usati abbiano più probabilità di essere riutilizzati rispetto a quelli che non sono stati usati di recente. È uno dei metodi più efficaci, ma richiede la gestione di un registro dei tempi di accesso per ogni blocco, il che può aumentare la complessità hardware.
- **FIFO (First In, First Out):** il blocco che è stato caricato per primo nella cache è quello che viene espulso. È semplice da implementare, ma meno ottimizzato rispetto a LRU, poiché non tiene conto dell'effettivo utilizzo recente dei dati.
- **Random:** espelle un blocco scelto a caso. Questo metodo è semplice da implementare e veloce, ma può comportare una maggiore probabilità di espulsione di blocchi che saranno riutilizzati di frequente, riducendo l'efficienza della cache.
- **Round Robin:** utilizza un ciclo continuo per scegliere quale blocco espellere, utilizzando un puntatore che viene aggiornato a ogni rimpiazzo. Questo approccio è meno comune ma può essere utilizzato in alcune cache completamente associative.

Per quanto riguarda le scritture nelle cache, esistono due principali strategie di gestione:

- **Write-through:** ogni volta che viene eseguita una scrittura nella cache, la stessa scrittura viene immediatamente propagata nella memoria principale (o nella cache di livello superiore). Questo approccio è semplice da implementare, ma comporta un rallentamento delle prestazioni, in quanto ogni scrittura nella cache necessita anche di una scrittura nella memoria principale, aumentando il traffico sulla memoria.
- **Write-back:** le scritture vengono effettuate solo nella cache, e i dati vengono trasferiti alla memoria principale solo quando il blocco viene rimosso dalla cache, cioè al momento della sostituzione. Questo approccio riduce il traffico di scrittura nella memoria principale e quindi migliora le prestazioni, ma è più complesso da gestire, in quanto è necessario tenere traccia dei blocchi modificati nella cache.

La scelta tra queste due strategie dipende dalle specifiche esigenze di prestazione e dalla complessità del sistema: **write-through** è più semplice e garantisce la coerenza immediata tra cache e memoria, mentre **write-back** è più efficiente ma richiede un meccanismo di aggiornamento ritardato, aumentando la complessità nella gestione della coerenza dei dati.

## 8.8 Cache coherence nei multicore

Nei sistemi multicore, ogni core dispone di una cache privata: per evitare inconsistenze sui dati condivisi serve un **protocollo di coerenza**. La coerenza (CC, Cache Coherence) garantisce che tutti i core osservino la stessa sequenza logica di aggiornamenti per una cella di memoria condivisa. Il caso ideale — **strong coherence** — impone che tutte le letture e scritture siano viste nello stesso ordine da tutti i core. Al contrario, con **weaker coherence**, le operazioni di scrittura possono non essere immediatamente visibili dagli altri core. Questo approccio riduce il carico di gestione della coerenza, permettendo al processore di eseguire altre operazioni in parallelo senza dover aspettare che tutte le scritture siano propagate e visibili a tutti i core.

Condizioni sufficienti:

1. **Single-Writer/Multiple-Readers (SWMR):** in un dato istante un solo core può scrivere su una cella, ma più core possono leggerla.
2. **Data-Value (DV):** il valore della cella è identico per tutti i core all'inizio di ogni epoca.

Molti sistemi reali adottano varianti **weaker coherence**, sacrificando temporaneamente la visibilità per migliorare prestazioni e parallelismo.

**8.8.0.1 Protocolli di coerenza** In un sistema multi-core, i dati condivisi devono essere coerenti tra le varie cache locali (L1, L2) per evitare letture di valori obsoleti o incoerenti. Esistono due principali categorie di protocolli per garantire la coerenza:

- **Invalidation:** quando un core scrive in un blocco di memoria, invalida tutte le copie di quel blocco nelle altre cache. Questo significa che ogni altro core che ha una copia del blocco dovrà ricaricarlo dalla memoria principale o dal livello superiore della cache prima di poterlo usare di nuovo. L'Invalidation è un approccio semplice, che riduce il traffico di coerenza, ma può aumentare la latenza se i dati invalidati sono frequentemente richiesti.
  - *Pro:* riduce il traffico di coerenza, adatto per carichi di lavoro con bassa contesa.
  - *Contro:* può causare elevata latenza in caso di **cache miss**, poiché le copie devono essere ricaricate dalla memoria.
- **Update:** quando un core scrive un dato, aggiorna tutte le copie di quel blocco nelle altre cache. Questo approccio garantisce che ogni copia in cache sia aggiornata in tempo reale, riducendo il rischio di letture obsolete, ma comporta un maggiore traffico di coerenza tra i core.
  - *Pro:* garantisce che tutti i core vedano subito i dati più recenti.
  - *Contro:* aumenta il traffico di coerenza e può ridurre le prestazioni, specialmente in sistemi con molteplici core che scrivono frequentemente.

**Implementazioni dei protocolli:** I protocolli di coerenza possono essere implementati in due principali modalità:

- **Snooping:** ogni controller di cache “sente” (snoops) le richieste di accesso sulla linea di comunicazione condivisa tra i core, e reagisce di conseguenza. Ogni core monitora il bus di comunicazione (ad esempio il **system bus**) per rilevare accessi a blocchi condivisi e determinare se deve aggiornare o invalidare una cache. Questo approccio è tipico nelle architetture con un bus condiviso, come quelle usate nei sistemi a singolo socket o con numero limitato di core.
  - *Pro:* semplice da implementare, non richiede una struttura centralizzata.
  - *Contro:* inefficiente per sistemi con molti core, poiché ogni cache deve monitorare l'intero bus, aumentando il carico e il traffico.

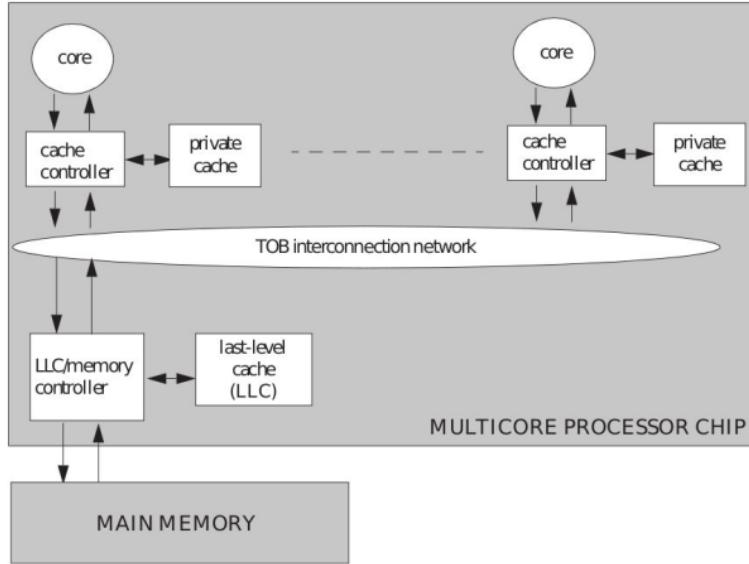


Figura 60: Snooping cache system model

- **Directory-based:** in questo modello esiste un nodo centrale, una **directory**, che tiene traccia di quali core possiedono una copia di ogni blocco di memoria. La directory riceve richieste di accesso da parte dei core e, quando un core scrive su un blocco, essa aggiorna o invalida le copie nelle altre cache. Questo modello è più scalabile rispetto allo snooping, poiché centralizza la gestione della coerenza, evitando che ogni cache debba monitorare il bus.
  - *Pro:* più scalabile in sistemi con molti core, riduce il traffico di coerenza.
  - *Contro:* maggiore complessità nell’implementazione e potenziale punto di congestione nella directory.

**Sintesi:**

- Il modello **Invalidate** è efficace in scenari con bassa contesa ma può rallentare l’accesso in caso di cache miss.
- Il modello **Update** è più costoso in termini di traffico di coerenza ma offre una visione più coerente dei dati.
- Il modello **Snooping** è semplice da implementare ma non scala bene per sistemi a molti core, mentre **Directory-based** è scalabile ma più complesso da gestire.

## 8.9 Protocolli MOESI e VI

I protocolli di coerenza sono descritti come **macchine a stati finiti** (FSM), dove ciascun blocco in cache può trovarsi in diversi stati di validità. Il protocollo più semplice è il **VI**, con due soli stati (Valid/Invalid).

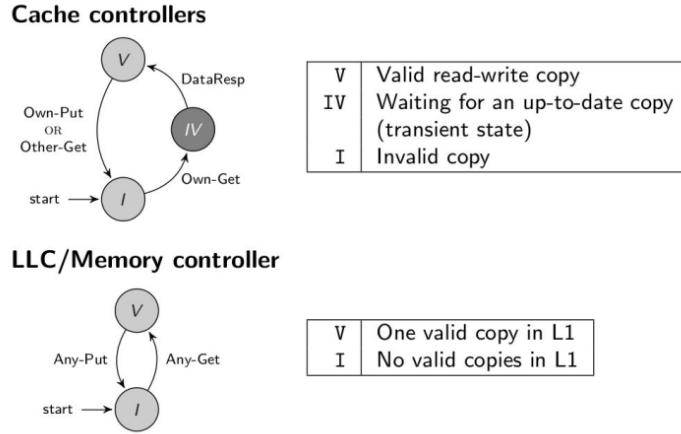


Figura 61: The vi protocol

Tuttavia, i sistemi moderni adottano versioni più espressive come il **MOESI**, che introduce:

- **M (Modified):** il blocco è valido, esclusivo, potenzialmente sporco; solo questa cache lo possiede.
- **O (Owned):** valido e potenzialmente sporco ma condiviso; la cache risponde alle richieste di altri core.
- **E (Exclusive):** valido e pulito; nessun'altra cache lo possiede.
- **S (Shared):** valido e condiviso, non modificabile.
- **I (Invalid):** non contiene dati validi.

Il protocollo MOESI consente di minimizzare traffico e latenza mantenendo comunque una visione coerente dei dati, sfruttando il concetto di *ownership* per coordinare chi deve rispondere a richieste remote.

**8.9.0.1 Virtual vs. Physical Cache Indexing** Le cache possono essere **indicizzate** usando l'indirizzo virtuale o quello fisico, a seconda del livello e della microarchitettura. La scelta influisce su latenza, complessità hardware e rischi di inconsistenza tra TLB e cache.

- **Virtual-indexed caches:** usano l'indirizzo virtuale generato dalla CPU, prima della traduzione tramite il **Translation Lookaside Buffer (TLB)**. Questo approccio è tipico della **L1 cache**, dove la priorità è la *bassa latenza* e la velocità di accesso supera i rischi di incoerenza. Tuttavia, poiché la traduzione virtuale→fisica non è ancora disponibile, il sistema deve gestire eventuali aliasing (lo stesso indirizzo fisico visto come virtuale diverso da più processi).
- **Physically-indexed caches:** usano invece l'indirizzo fisico, cioè quello tradotto dal TLB. Questo evita i problemi di aliasing e garantisce coerenza tra processi e livelli di cache diversi, ma richiede che la traduzione sia già avvenuta, aumentando leggermente la latenza. Le **L2 e L3 cache** usano quasi sempre indicizzazione fisica.
- **TLB (Translation Lookaside Buffer):** memorizza le traduzioni virtuale→fisico più recenti, riducendo il costo di accesso alla memoria principale. Quando il TLB fornisce rapidamente la traduzione, l'indicizzazione fisica non penalizza eccessivamente le prestazioni.

In sintesi, la cache L1 privilegia l'accesso immediato e può usare indirizzi virtuali, mentre i livelli esterni (L2, L3) scelgono la coerenza globale e usano indirizzi fisici. Questo bilanciamento permette di minimizzare la latenza mantenendo consistenza tra processi, TLB e memoria principale.

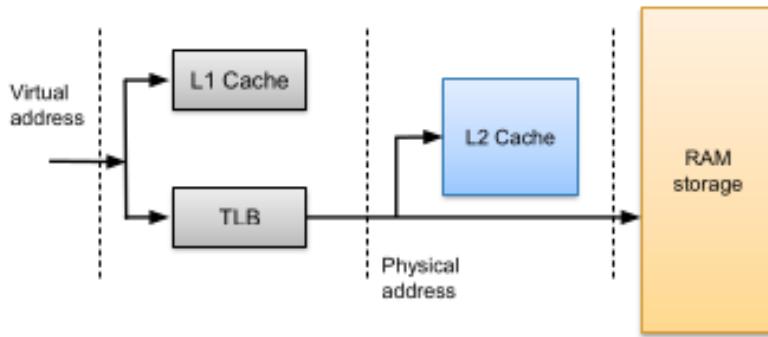


Figura 62: Virtual Vs Physical cache indexing

## 8.10 Hardware Transactional Memory (HTM)

Per ridurre la complessità della sincronizzazione esplicita tra thread (lock, mutex), le CPU moderne supportano la **Transactional Memory** hardware, introdotta da Intel nel 2013 (TSX). Un blocco di codice viene eseguito come transazione: tutte le operazioni sono provvisorie fino al commit, e se si verifica un conflitto con altri thread, la transazione viene annullata automaticamente. Questo approccio fornisce concorrenza “lock-free” e semplifica il codice multithreaded.

```

1 retry:
2     or eax, OFFFFFFFFh
3     xbegin LO
4 LO:
5     cmp eax, OFFFFFFFFh
6     jne L1 ;jump to 'else' branch
7     ;transaction body starts here
8     ....
9     ;transaction body ends here
10    xend
11    ;transaction body ends here
12    jmp L2
13    ;actions on aborts
14 L1: ;'else' branch
15     ;actions on aborts
16     ....
17     ....
18     jmp retry
19 L2:
20     ....

```

Esempio in C con `_xbegin()` e `_xend()`

Equivalenti in assembly con `xbegin/xend`

Figura 63: Confronto tra implementazione C e assembly di una transazione TSX

**8.10.0.1 Principio di funzionamento** Le transazioni si appoggiano alla coerenza cache: le scritture restano nella L1 finché non si effettua il commit. Durante l'esecuzione, l'hardware tiene traccia di due insiemi:

- **Read set:** blocchi letti dalla transazione.
- **Write set:** blocchi modificati (marcati nella cache come tentativi).

Un conflitto si verifica se un altro core tenta di accedere a un blocco presente in questi insiemi; il core rileva la violazione e *aborts* la transazione, ripristinando lo stato precedente. Le istruzioni principali sono `xbegin`, `xend`, `xabort`, `xtest`. Un'implementazione comune è la **RTM (Restricted Transactional Memory)**, che richiede fallback non transazionale in caso di fallimenti ripetuti.

**8.10.0.2 Casi di abort e codici di stato** Quando una transazione fallisce, l'istruzione `_xbegin()` restituisce in `EAX` un **codice di stato** che indica la causa dell'abort. Le ragioni più comuni sono:

- **Conflitti di accesso:** un altro core accede (in lettura o scrittura) a una linea di cache inclusa nel *write set* della transazione corrente.
- **Interruzioni o eccezioni asincrone:** interrupt, context switch, page fault o eventi che invalidano il contesto transazionale.
- **Overflow o limiti hardware:** superamento della capacità dei buffer interni o del numero massimo di linee L1 monitorabili (tipicamente poche decine di KB).
- **Abort esplicito:** uso dell'istruzione `XABORT`, che forza la terminazione e può passare un argomento nei bit 24–31 del registro `EAX`.
- **Assenza di supporto TSX:** il bit `CPUID.07H.EBX.RTM` non è impostato, quindi le istruzioni transazionali vengono ignorate o causano eccezione.

I bit del registro `EAX` restituiti in caso di abort sono così interpretati:

EAX Bit	Significato
0	Abort causato da istruzione <code>XABORT</code>
1	Transazione potenzialmente ripetibile al retry
2	Conflitto con altro processore su indirizzo monitorato
3	Overflow del buffer interno
4	Hit di breakpoint o debug event
5	Abort durante una transazione annidata
6–23	Riservati
24–31	Argomento passato a <code>XABORT</code>

**Tabella 1:** Codici di stato restituiti nel registro `EAX` in caso di abort transazionale

Se nessun bit è impostato (`EAX = 0`), significa che la transazione è stata abortita senza una causa specifica riconoscibile (abort generico).

## 8.11 DRAM e Refresh

La **Dynamic RAM (DRAM)** conserva ciascun bit come carica elettrica in un piccolo condensatore, controllato da un transistor. Col tempo la carica si disperde attraverso resistenze parassite, quindi ogni cella deve essere periodicamente **rinfrescata** per non perdere il dato. Il tempo caratteristico di scarica è  $\tau = RC$ , e ogni riga viene tipicamente refreshata ogni 64 ms.

### 8.11.0.1 Effetti del refresh

- **Energia:** ogni ciclo di refresh consuma energia addizionale.
- **Prestazioni:** durante il refresh le righe interessate non sono accessibili.
- **Scalabilità:** più capacità significa più righe da aggiornare e quindi pause più lunghe.

Per mitigare l'impatto si adottano tecniche di **distributed refresh**: il controller rinfresca blocchi separati in tempi diversi, riducendo le pause globali. Altre strategie (es. *retention-aware refresh*) regolano dinamicamente la frequenza in base alla stabilità elettrica delle celle, riducendo consumi senza compromettere l'integrità dei dati.

## Conclusione

Questo modulo mostra come l'evoluzione hardware — dal parallelismo a livello di pipeline e thread, alla gestione speculativa e coerente delle cache, fino alle transazioni e alla DRAM — influenzino direttamente i modelli di programmazione e le garanzie di sicurezza dei sistemi moderni. Ogni livello della gerarchia hardware (CPU, cache, memoria, controller) contribuisce a bilanciare potenza, latenza e coerenza, delineando le fondamenta su cui si costruiscono le architetture di sicurezza e difesa dei sistemi.

## 9 SYS\_04 - Hardware attacks and countermeasures

### 9.1 Introduzione agli Attacchi Hardware

Gli attacchi hardware sfruttano vulnerabilità fisiche e architetturali delle componenti elettroniche dei sistemi, come CPU, memoria e cache. Tali attacchi sono difficili da rilevare dai software di sicurezza e sfruttano caratteristiche intrinseche dell'hardware, come la temporizzazione, l'uso della cache e l'esecuzione speculativa. Gli attacchi hardware possono includere:

- Attacchi di temporizzazione (Timing Attacks)
- Attacchi a canale laterale (Side-Channel Attacks)
- Manipolazioni della cache (Cache Attacks)
- Attacchi alle funzioni speculative (Speculative Execution Attacks)
- Attacchi basati su memoria (Memory-based Attacks)
- Attacchi di RowHammer (attacchi a memoria dinamica)

### 9.2 Timing e Sicurezza

Gli attacchi di temporizzazione sfruttano la misurazione dei tempi di esecuzione di operazioni su dati sensibili per ottenere informazioni sui dati stessi. Ad esempio, se un algoritmo impiega più tempo per elaborare determinati dati, è possibile dedurre informazioni su di essi, come nel caso delle password o delle chiavi di crittografia.

**9.2.0.1 Uso della Funzione strcmp** La funzione `strcmp()` confronta due stringhe carattere per carattere. Se le stringhe sono uguali, la funzione restituisce un valore pari a zero. Tuttavia, se le stringhe differiscono, il tempo impiegato per confrontarle aumenta progressivamente, con il risultato che, più a fondo si arriva nel confronto, maggiore sarà il tempo impiegato. Ciò consente a un attaccante di dedurre quale carattere della stringa è corretto, basandosi sul tempo impiegato per completare il confronto.

```
int strcmp(char *t, char *s)
{
    for( ; *t == *s ; s++, t++)
        if(*t == '\0')
            return 0;
    return *t - *s;
}
```

Figura 64: esempio di utilizzo di funzione strcmp

**9.2.0.2 Timing attack in un Programma di Autenticazione** Supponiamo che un programma di autenticazione confronti un input (`t`) con la password corretta (`s`) usando `strcmp()`. Se `t = "password"` e `s = "password"`, il programma impiega più tempo per confrontare i caratteri quando l'input è corretto, rispetto a quando c'è un errore, permettendo all'attaccante di inferire gradualmente i caratteri della password.

#### 9.2.1 Attacchi di Temporizzazione nel Mondo Reale: RSA

Nel contesto della crittografia, gli attacchi di temporizzazione possono essere utilizzati per ottenere informazioni sulla chiave privata di un sistema come RSA. La decriptazione RSA implica una serie di operazioni di esponentiazione modulare, che dipendono dal valore della chiave privata.

**9.2.1.1 Attacchi alla Decrittazione RSA** Gli attacchi alla decrittazione RSA si basano sul rilevamento delle differenze nei tempi di esecuzione durante il calcolo dell'esponentiazione modulare. Ad esempio, l'attaccante può inviare cifrati progettati per creare tempi di risposta differenti a causa dei passaggi computazionali coinvolti. Questo fenomeno, noto come il *gap 0-1*; si verifica quando la decrittazione impiega meno tempo (0) quando certe condizioni sono soddisfatte e più tempo (1) quando altre condizioni richiedono più passaggi computazionali. Osservando queste differenze nei tempi di esecuzione, l'attaccante può inferire parti della chiave privata. Ad esempio, nel caso dell'esponente modulare, la modifica di uno dei bit della chiave può accelerare il calcolo, rivelando informazioni sul valore della chiave.

**9.2.1.2 Algoritmi a Tempo Costante** Per mitigare i Timing attacks, è essenziale utilizzare algoritmi che impiegano lo stesso tempo per ogni confronto, indipendentemente dai dati in ingresso. Un esempio di implementazione sicura di confronto di stringhe potrebbe essere la funzione `constantTimeStringCompare()` che confronta ogni carattere delle stringhe senza fermarsi fino a quando tutti i caratteri sono stati verificati, impedendo così che l'attaccante possa ottenere informazioni tramite differenze nei tempi di esecuzione. Ovviamente questo porta ad un peggioramento delle prestazioni, ma aumenta la sicurezza del sistema.

```
int constantTimeStringCompare(const char *t, const char *s, size_t length)
{
    int result = 1;

    for (size_t i = 0; i < length; i++, t++, s++) {
        result &= (*t == *s);
    }

    return result ? 0 : 1;
}
```

Figura 65: esempio di funzione `constantTimeStringCompare`

### 9.3 Preambolo: Side-Channel Attacks e Considerazioni Tecniche

I **side-channel attacks** sfruttano effetti micro-architetturali osservabili (tempi di accesso, stato delle cache, aborti transazionali, ecc.) per *inferire* dati che dovrebbero restare isolati tra processi/VM. L'idea chiave è: portare il sottosistema (in primis la gerarchia di cache) in uno *stato noto*, lasciare che la vittima esegua, quindi *misurare* un segnale (latenze, aborti, rumore) che rivela pattern di accesso o contenuti. Tecniche tipiche includono *Prime+Probe*, *Flush+Reload*, *Flush+Flush*, *Evict+Time* e *Prime+Abort*. Questi attacchi possono bypassare l'isolamento del sistema operativo sfruttando cache condivise (L3) o l'SMT per L1.

#### 9.3.0.1 Fasi operative

1. **Pre-attack:** selezione del target (set/linea di cache), calibrazione delle *soglie temporali* per distinguere *hit* (veloce) da *miss* (lento).
2. **Active attack:** inizializzazione (flush/prime), attesa dell'accesso vittima, misure e analisi degli effetti (tempo di reload/probe, aborti TSX), iterazione finché il segreto (es. chiave, nonce) viene ricostruito.

**9.3.0.2 Scoprire i percorsi di codice** Anche algoritmi con conteggio operazioni costante (es. *Montgomery Ladder*) possono rivelare *quale ramo* viene seguito, osservando quali linee di cache vengono toccate: il *timing della cache* permette di inferire il cammino senza cronometrare il numero di operazioni.

### 9.3.1 Technical Considerations

- **Precisione temporale:** servono timer ad alta risoluzione (`rdtsc`, `clock_gettime`); drift e disallineamento tra core introducono rumore.
- **Rumore e variabilità:** contesa CPU e *cache miss* non correlati disturbano le misure; aumentare i campionamenti e usare statistiche robuste (mediana) aiuta.
- **Ottimizzazioni HW/SW:** *branch prediction* ed *speculative execution* alterano i tempi; l'*Hyper-Threading (SMT)* introduce interferenze tra thread sullo stesso core.
- **Scheduling di processo:** l'interleaving del SO riduce la stabilità delle misure; *CPU affinity/pinning* migliora accuratezza e ripetibilità.
- **Gerarchie di cache e prefetch:** L1/L2/L3 hanno latenze diverse che plasmano il segnale; il *prefetching* riduce la predicitività dei miss e va considerato nel disegno dell'esperimento.

**9.3.1.1 Nota pratica (TSX e detection)** Varianti come *Prime+Abort* sfruttano aborti transazionali come *callback hardware* (senza timing esplicito), ma possono essere limitate disabilitando TSX; in generale la costruzione del canale produce *rumore* misurabile con *Hardware Performance Counters* (HPC), utile per profili di detection—sebbene i contatori differiscano tra vendor e generazioni.

## 9.4 Attacchi side channel

Gli attacchi a canale laterale sfruttano informazioni che possono essere osservate indirettamente durante l'esecuzione di un programma, come il consumo di energia, la radiazione elettromagnetica, o l'accesso alla memoria. Un esempio di attacco a canale laterale è quello che sfrutta il comportamento della cache, che può rivelare informazioni su operazioni sensibili come le operazioni crittografiche.

### 9.4.1 Timing della Cache

Quando si leggono i dati dalla memoria, la CPU li legge effettivamente dalla cache, che cerca di servire una richiesta di memoria il più velocemente possibile. Se un attaccante può controllare lo stato della cache, può sfruttare i tempi di accesso per ottenere informazioni sui dati memorizzati. Gli attacchi alla cache si possono dividere in varie tecniche, come *Prime + Probe* e *Flush + Reload*, che mirano a identificare quali dati sono stati caricati nella cache.

**9.4.1.1 Attacchi Flush + Reload** Il metodo *Flush + Reload* sfrutta il fatto che le linee di cache possono essere “scaricate” (flushed) e successivamente ricaricate. Misurando i tempi necessari per ricaricare una linea di cache, un attaccante può determinare se un altro processo ha accesso alla stessa linea di memoria, ottenendo così informazioni sensibili.

**9.4.1.2 Evict + Time** La tecnica *Evict + Time* combina l'evizione mirata di una linea/set di cache con la misura del *tempo di esecuzione complessivo* della vittima. Prima si lascia girare la vittima per pre-caricare il proprio working set e si stabilisce una *baseline* temporale; poi si *evince* la linea di interesse accedendo ripetutamente a blocchi che mappano nello *stesso cache set* (cache set-associative), quindi si rilancia la vittima: una variazione del tempo indica che quella linea è stata usata e deve essere stata ricaricata dalla memoria. Non richiede memoria condivisa (a differenza di *Flush+Reload*), funziona su L1/L2/L3 e fornisce un segnale più *coarse* (livello di set) perché si osserva il tempo del programma vittima, non il reload del singolo indirizzo. È utile per inferire *code paths* dipendenti dai dati, ad esempio in implementazioni crittografiche con tabelle, dove il diverso accesso alle entry rivela bit della chiave. Operativamente è importante

poter *triggerare* la vittima più volte, usare timer ad alta risoluzione e barriere (preferire `rdtscp` o fence per rispettare il program order) e considerare rumore da scheduling, frequency scaling e prefetching, che alterano la stabilità della misura.

**9.4.1.3 Attacchi Prime + Probe** L'attacco *Prime + Probe* è uno degli attacchi più noti alla cache. L'attaccante accede alla cache riempiendola con i propri dati e successivamente fa probing sui dati osservando il tempo necessario per accedere ai dati inseriti. Se un dato è stato sostituito da un altro processo, l'accesso sarà più lento, rivelando informazioni sul comportamento del sistema. Questo attacco targetta la cache di livello L1 ma può essere esteso fino ad L3 e può funzionare anche su una VM. Avere successo con questa tipologia di attacchi non è così semplice visto che l'attaccante ha bisogno di un set di indirizzi che collidano. Una contromisura hardware implementata dalle cache moderne prevede la definizione del mapping della memoria di cache a runtime questo vuol dire che il mapping cambia nel tempo di conseguenza l'attaccante ha una finestra ridotta di tempo per creare il suo eviction set implementando questi step:

- scegliere randomicamente N indirizzi di memoria
- caricare nella cache accedendo agli indirizzi scelti
- gli indirizzi che self-collidono devono essere rimossi dal set

**9.4.1.4 Prime + Abort** *Prime + Abort* sfrutta le transazioni hardware (Intel TSX) come una sorta di *callback* micro-architetturale: si avvia una transazione, si accede *dall'interno* della transazione alla linea di cache bersaglio e poi si osserva se la transazione va in *abort*. L'*abort* segnala un conflitto nel *write set/read set* causato dall'attività della vittima su quella linea, per cui **non serve alcuna misura temporale** (il segnale è l'evento di abort), a differenza di Prime+Probe o Flush+Reload. Requisito chiave: conoscere l'indirizzo preciso da monitorare (granularità a livello di *cache line*). Nel caso di **L1**, la transazione include la linea nel *write set*: una scrittura concorrente o un'evizione durante la finestra transazionale provoca abort; essendo L1 privata per core, lo *spionaggio* è limitato a thread co-residente via SMT sullo *stesso* core. In **L3**, invece, l'accesso transazionale mette la linea nel *read set*: la coerenza cache su letture/scritture di altri core può indurre abort, consentendo osservazioni *cross-core*. In sintesi: nessuna memoria condivisa richiesta, segnale netto (abort), granularità fine (linea), ma dipendenza da TSX e dalla possibilità di ripetere il trigger della vittima. In molti ambienti difensivi TSX può essere disabilitato o limitato; inoltre l'attività di preparazione/sonda introduce rumore rilevabile con *Hardware Performance Counters*.

**9.4.1.5 Flush + Flush** *Flush + Flush* è una variante di *Flush + Reload* che usa come segnale non il tempo di *reload*, ma il tempo d'esecuzione dell'istruzione di flush (es. `clflush`) sulla *stessa* linea di cache ripetutamente. Poiché `clflush` impiega un tempo diverso a seconda che la linea sia presente o meno in cache, misurando la latenza del *flush* si può inferire se la vittima ha toccato quella linea. Il vantaggio è duplice: è più *stealthy* (non esegue accessi di reload "visibili") ed è spesso più veloce; inoltre non richiede necessariamente memoria condivisa. La granularità resta a livello di *cache line* e funziona su L1/L2/L3. Operativamente servono timer ad alta risoluzione e barriere per stabilizzare le misure, mentre rumore da scheduling, prefetching e frequenza CPU può influenzare la separazione delle soglie hit/miss; in ambienti difensivi, l'uso anomalo di `clflush` può essere monitorato o limitato.

## 9.4.2 Meeting Out-of-Order Pipelines: perché conta per la sicurezza

Nei processori moderni, **OOO + speculative execution** possono far eseguire alla pipeline *istruzioni fantasma* (transient) che verranno poi scartate a livello architettonico, ma che lasciano *effetti micro-architetturali* (es. stato della cache) osservabili con un canale laterale temporale.

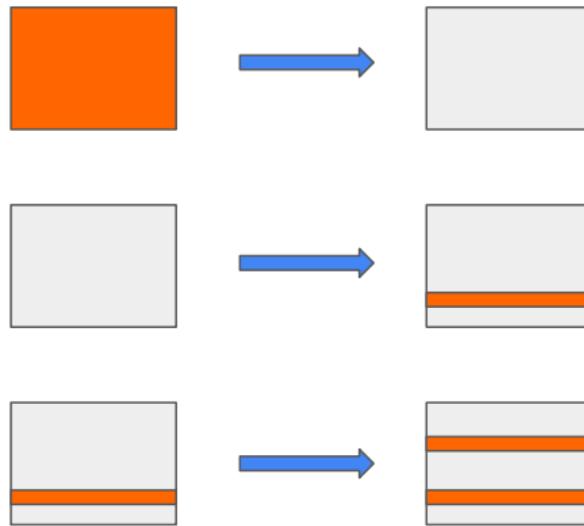
Questo è il razionale dei *Transient Execution Attacks* (Spectre/Meltdown): anche se il backend “flusha” la pipeline, le tracce rimangono abbastanza a lungo da essere misurate. Il fenomeno riguarda più famiglie di CPU (Intel, AMD, ARM).

### 9.4.3 Meltdown: come funziona (Primer)

**9.4.3.1 Idea chiave** Meltdown sfrutta il fatto che **il controllo di privilegio** sugli accessi memoria avviene *dopo* che la pipeline ha già iniziato ad eseguire speculativamente l'accesso. Durante la finestra *transient*, si può *dereferenziare* un indirizzo kernel e usare il valore letto per *indicare* (via stride di pagina) una linea di una *probe array* utente, portandola in cache. Anche se l'accesso privilegiato causa *segfault* e viene annullato architetturalmente, la cache resta “marcata” e il valore del byte si recupera misurando la latenza di reload sui 256 slot dell'array di probing.

#### 9.4.3.2 Schema operativo Meltdown

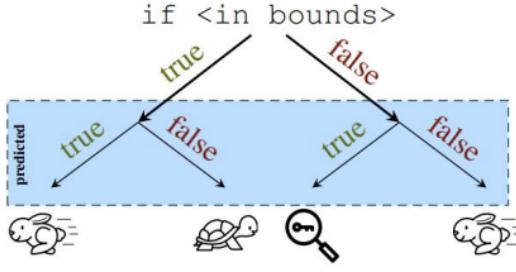
1. **Flush** della cache.
2. **Read** di un byte  $B$  in memoria kernel (vietato in user mode ma *transientmente* eseguito).
3. **Utilizzo** del byte  $B$  come offset: accesso a `probe_array[B * 4096]` per portare in cache una linea alla volta.
4. **Reload & Time**: si misura quale fra i 256 slot presenti nell'array di probing è in cache.



**9.4.3.3 Perché è possibile** Storicamente, i kernel mappavano l'intero spazio kernel nello *address space* dei processi utente (bit di privilegio nei PTE impediva l'uso lecito, ma l'entry era presente). La finestra *transient* tra *address translation/data fetch* e *privilege check* consente l'effetto in cache. Misurare la latenza di accesso alla probe array a fine sequenza rivela il valore reale del byte kernel.

### 9.4.4 Fooling the Branch Prediction Unit (BPU)

La **Branch Prediction Unit (BPU)** apprende dagli esiti recenti e tenta di anticipare il percorso di esecuzione. Un attaccante può *avvelenare* (miss-train) la BPU eseguendo ripetutamente un pattern di codice così che la previsione diventi stabile; cambiando poi il comportamento del codice, la BPU continua a predire il ramo “allenato”, consentendo l'esecuzione *transient* di istruzioni non autorizzate. Anche se la pipeline viene poi *flushata*, gli effetti micro-architetturali (es. cache) restano osservabili via side-channel.



**9.4.4.1 Dove si colpisce e perché funziona** Il *branch target buffer* (BTB) e i predittori (locali/globali, *tournament*) memorizzano storia e bersagli dei salti; l'attaccante può correlare rami, sfruttare la storia globale e indurre la predizione di un *target* controllato (anche per salti indiretti). Questa architettura è descritta negli appunti: predittori correlati, tournament predictor, BTB come piccola cache indicizzata dal *program counter*.

#### 9.4.5 Spectre Primer v1 (Bounds Check Bypass)

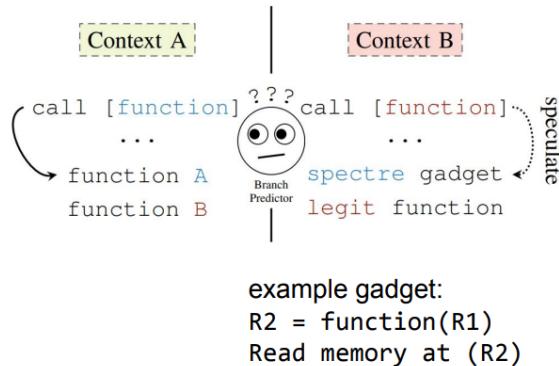
L'idea classica: si forza la speculazione oltre un controllo di bounds, usando l'indice per selezionare una linea di cache in una *probe array*; a fine speculazione si misura quale linea è calda e si ricostruisce il dato.

```
if (x < array1_size) {
    y = array2[array1[x] * 4096];
}
```

Anche se la condizione è falsa, durante la finestra speculativa la CPU può accedere a `array1[x]` fuori limite e toccare `array2[ ... ]`, lasciando un'impronta in cache che rivela `array1[x]`.

#### 9.4.6 Spectre Primer v2 (BTB Poisoning e Gadget)

Qui si mira ai **salti indiretti**: l'attaccante sceglie un *gadget* nell'*executable address space* della vittima (anche in eBPF), poi **avvelena il BTB** per far *mispredict* il salto indiretto verso quel gadget, che viene eseguito speculativamente. Il codice della vittima non è “buggato”: è sfruttato come gadget per eseguire accessi che *encodano* il dato in cache. Questo è concettualmente vicino alla ROP.



#### 9.4.6.1 Esempio di gadget dalle slide (Windows 10)

```
adc edi, dword ptr [ebx+edx+13BE13BDh]
adc dl, byte ptr [edi]
```

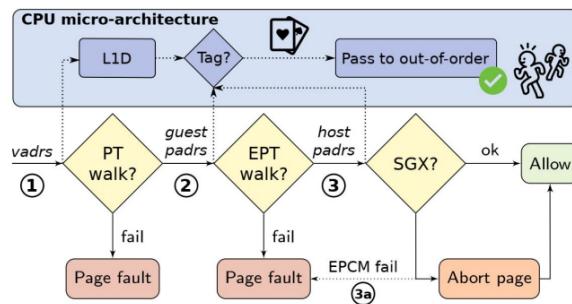
Impostando `edi` alla base di una *probe array* e `ebx = m - 0x13BE13BD - edx`, la prima istruzione legge 32 bit da `m` e li somma a `edi`, la seconda *tocca* `probe[m]` portandone la linea in cache. Così si codifica l'indice `m` nel canale; l'attaccante controlla i registri e sfrutta la finestra speculativa per far eseguire il gadget.

#### 9.4.6.2 Requisiti e note

Il miss-training della BPU deve avvenire sullo stesso core; la lettura del canale (timing di cache) può accadere altrove.

#### 9.4.7 L1TF (Foreshadow): principio e flusso

Durante la traduzione degli indirizzi virtuali in indirizzi fisici, i processori Intel eseguono in *parallelo* sia il flusso di traduzione (consultazione della **TLB**, eventuale **EPT walk** — *Extended Page Tables* in presenza di **VM**, e la verifica **SGX** — *Software Guard eXtensions*) sia la ricerca nella **cache L1 VIPT** — *Virtual-Indexed, Physically-Tagged*. In questa finestra *transient*, una entry della tabella delle pagine *non valida* può essere *speculativamente propagata* alla CPU: l'effetto risultante nella L1 rimane osservabile con un side-channel nello *stile Meltdown* (tracce in cache nonostante il flush architetturale). Questo consente a **L1TF** di *bypassare* le verifiche di protezione del **SO** durante la navigazione nella tabella delle pagine (caso VM->EPT walk).



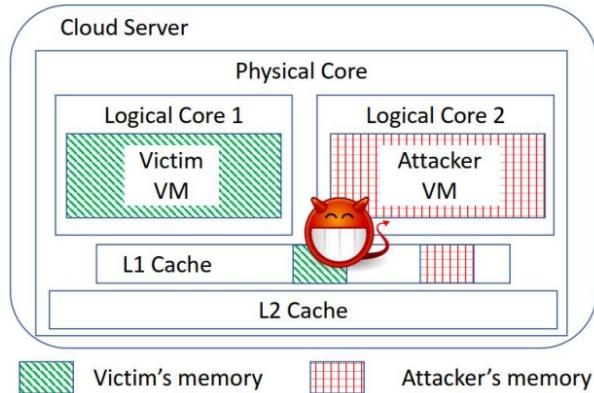
#### 9.4.7.1 Perché funziona

Un accesso a una pagina che dovrebbe generare un *terminal fault* (entry non presente o con bit riservati) può avanzare speculativamente abbastanza da lasciare effetti micro-architetturali nella L1. Poiché la ricerca in L1 procede mentre proseguono i controlli di privilegio/validità, l'indirizzo fisico speculativo può pilotare un'operazione di *encode* nel canale (es. accessi a una *probe array*); a fine finestra, il dato si ricostruisce misurando i tempi (hit/miss). In questo modo L1TF **aggira** i tre livelli di controllo: SO (PT walk), VMM/hypervisor (EPT walk) e SGX.

#### 9.4.7.2 Schema operativo (dalle slide)

- Preparazione PTE/EPT.** Creare una voce per l'indirizzo d'interesse e marcarla *invalida* (bit *present/reserved* a 0) così da generare un *terminal fault*. *Nota:* il kernel di norma non consente tali manipolazioni; però un attaccante con un *guest OS* dentro una **VM** può modificare le proprie **PT** — *Page Tables*. :contentReference[oaicite:3]index=3
- Trigger transiente.** Causare l'accesso che, pur faultando a livello architetturale, porta temporaneamente il dato in **L1**.
- Encode/Leak.** Usare un gadget di encodifica basato su cache (stile Meltdown) e misurare il canale per ricostruire il contenuto protetto. :contentReference[oaicite:4]index=4

Nel caso **VMM**, manipolando le **PT** del guest si può arrivare a *osservare* memoria dell'host o di altri guest co-locati, sfruttando la finestra transiente e la condivisione delle strutture di traduzione. :contentReference[oaicite:5]index=5



#### 9.4.8 Mitigating Side-Channel Attacks

Mitigare gli attacchi a canale laterale è una sfida complessa, poiché molti di questi attacchi si basano su effetti micro-architetturali che sono intrinseci nelle architetture moderne delle CPU. Tuttavia, diverse strategie possono essere adottate per ridurre il rischio o rendere più difficili gli attacchi.

**9.4.8.1 Possible Mitigations and Detection** Gli attacchi di temporizzazione sono difficili da mitigare completamente, poiché il comportamento delle cache dovrebbe essere modificato in modo sostanziale. Una soluzione teorica sarebbe quella di utilizzare cache a tempo costante, in cui non vi siano differenze nei tempi di accesso alle linee di cache tra hit e miss. Tuttavia, ciò richiederebbe una modifica fondamentale dell'architettura della cache, riducendo drasticamente le prestazioni, rendendo tale soluzione impraticabile per la maggior parte dei sistemi.

**9.4.8.2 Mitigations: the hard way** Le mitigazioni più efficaci richiedono cambiamenti significativi nell'architettura e nel software:

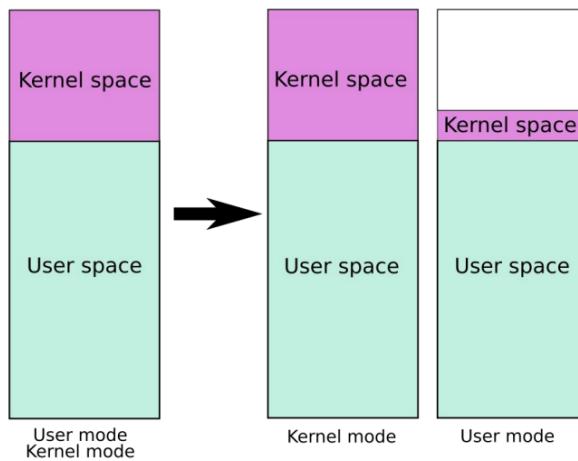
- Rimuovere o limitare l'accesso a timer ad alta risoluzione, come `rdtsc`, che vengono spesso utilizzati nei side-channel attacks. Questo è difficile, poiché tali timer sono necessari anche per il benchmark delle proprietà hardware.
- Consentire che alcune aree di memoria siano marcate come non cacheabili, ma ciò rappresenta una sfida hardware significativa.
- Utilizzare le istruzioni **AES-NI** (Advanced Encryption Standard New Instructions) nelle CPU Intel per eseguire AES, ma la stessa logica non si applica facilmente ad altri algoritmi di crittografia.
- Implementare tecniche come **scatter-gather**, dove i dati segreti non dovrebbero influenzare l'accesso alla memoria su una granularità maggiore di una linea di cache.
- Disabilitare le **TSX** (*Transactional Synchronization Extensions*), che sono utilizzate in alcune operazioni atomiche e che potrebbero essere sfruttate negli attacchi di tipo Prime + Abort.
- Disabilitare le pipeline *out-of-order* (OOO), sebbene ciò abbia un impatto significativo sulle prestazioni.

**9.4.8.3 Mitigations to L1 Attacks** Gli attacchi cross-process sulla cache L1 sono possibili solo quando due thread separati condividono la stessa L1. Poiché la cache L1 è privata per ogni core, gli attacchi a L1 sono praticabili solo se si fa affidamento sulla funzionalità **Simultaneous Multi-Threading (SMT)**. La soluzione più semplice per mitigare questi attacchi è **disabilitare SMT**, limitando la possibilità di attacchi tra thread di diversi processi che condividono lo stesso core.

**9.4.8.4 Mitigations: the detection way** L'allestimento di un canale laterale genera sempre un certo rumore nella gerarchia delle cache, soprattutto durante la fase di preparazione di alcuni attacchi. I moderni processori sono dotati di **Hardware Performance Counters (HPC)**, che consentono di tracciare eventi micro-architetturali e contarli. Questi contatori possono essere utilizzati per osservare il comportamento delle applicazioni in esecuzione nel sistema e attivare contromisure o mitigazioni selettive. Tuttavia, i contatori delle prestazioni non sono stabili tra diversi vendor e generazioni di CPU, quindi questa soluzione potrebbe non essere completamente affidabile.



**9.4.8.5 Meltdown Mitigation: Kernel Page Table Isolation** Una delle principali mitigazioni per Meltdown è l'implementazione della **Kernel Page Table Isolation (KPTI)**. Senza KPTI, lo spazio di memoria utente e kernel sono mappati nello stesso spazio di indirizzamento virtuale, con il kernel che è protetto dai livelli di privilegio. Con KPTI, i due spazi di indirizzamento sono separati, impedendo l'accesso al kernel quando la CPU è in modalità utente. Questo riduce il rischio che un attaccante possa sfruttare Meltdown per accedere ai dati del kernel.



#### 9.4.9 `cpu_entry_area`: Per-CPU Isolation

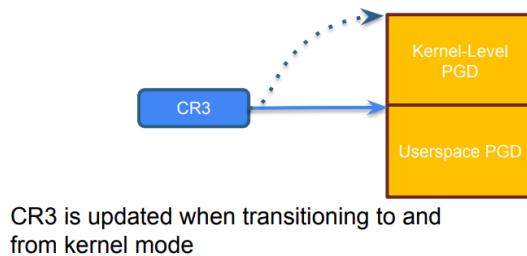
La struttura `cpu_entry_area` è utilizzata per isolare i dati specifici del processore, garantendo che le informazioni critiche, come la *Global Descriptor Table (GDT)*, le *Task State Segments (TSS)* e gli stack di eccezione, siano mantenute separate per ciascun core. Questo isolamento è

fondamentale per proteggere i dati tra i core e per impedire che informazioni sensibili vengano intercettate o compromesse durante i cambi di contesto. La **cpu\_entry\_area** è utilizzata per gestire in modo sicuro le transizioni durante gli interrupt e le eccezioni. Quando un processo cambia core, il sistema opera correttamente separando i dati relativi ai diversi core, riducendo il rischio di fuga di dati attraverso le transizioni tra i contesti di esecuzione.

#### 9.4.10 Double Page General Directory e Switch to CR3

La Double Page General Directory è una tecnica che separa le tabelle delle pagine tra spazio utente e spazio kernel, utilizzando due pagine di memoria da 4 KB ciascuna per mappare rispettivamente la memoria del kernel e quella utente. Questo approccio riduce il rischio di accessi non autorizzati alla memoria del kernel, come nel caso degli attacchi di tipo Meltdown. Il registro **CR3** è cruciale in questo contesto, poiché contiene il puntatore alla tabella delle pagine attualmente in uso. La transizione tra modalità kernel e utente richiede un aggiornamento del valore di CR3, garantendo l'isolamento tra i due spazi di memoria.

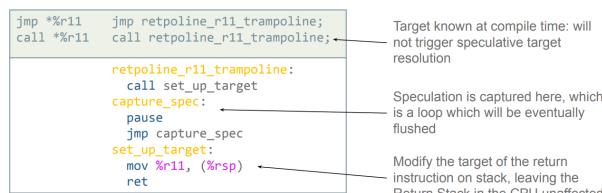
Le transizioni tra modalità si gestiscono tramite macro come **SWITCH\_TO\_KERNEL\_CR3** e **SWITCH\_TO\_USER\_CR3\_STACK**, che aggiornano CR3 durante il passaggio tra i due ambienti, assicurando che la separazione tra spazio utente e kernel venga mantenuta. Questo meccanismo è fondamentale per prevenire l'accesso non autorizzato alle risorse di sistema da parte di processi in esecuzione in modalità utente.



#### 9.4.11 Retpoline: Protezione contro gli Attacchi di Branch Target Injection e Retpoline Thunks

Il **Retpoline** è una tecnica software progettata per prevenire gli attacchi di *branch target injection*, tipici di Spectre, che sfruttano la predizione dei salti indiretti per manipolare l'esecuzione del codice. Il Retpoline impedisce che la CPU predica i salti indiretti, isolando queste istruzioni dall'esecuzione speculativa e forzando un salto controllato. Questo evita che gli attaccanti manipolino il flusso di esecuzione attraverso la speculazione e l'iniezione di indirizzi di salto non autorizzati. :contentReference[oaicite:0]index=0

I **Retpoline Thunks** sono funzioni che rimandano l'esecuzione di un calcolo fino a quando non è effettivamente necessario, evitando che il processore risolva speculativamente il target di salto. Questo previene la speculazione su indirizzi di ritorno e protegge il flusso di controllo dagli attacchi basati sulla predizione dei salti indiretti. :contentReference[oaicite:1]index=1



#### 9.4.12 Prevent Branch Poisoning: IBRS, STIBP, IBPB

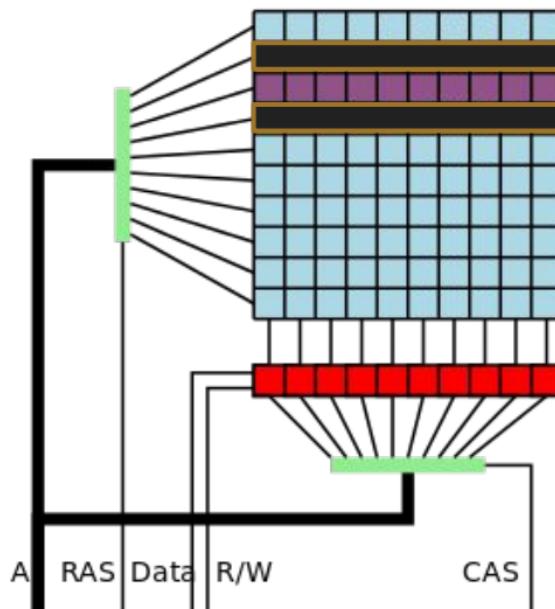
Le vulnerabilità legate alla *branch prediction* possono essere mitigate mediante tecniche come **IBRS**, **STIBP** e **IBPB**, che impediscono la contaminazione della predizione dei rami da parte di software non autorizzato.

- **IBRS** (Indirect Branch Restricted Speculation) è una modalità speciale che viene attivata nei processori per evitare che i predittori dei rami vengano influenzati da istruzioni eseguite prima dell'attivazione di IBRS. Questo garantisce che la predizione del ramo venga isolata, prevenendo gli attacchi di *branch target injection*.
- **STIBP** (Single Thread Indirect Branch Prediction) impedisce la condivisione della predizione dei salti indiretti tra i thread che girano sullo stesso core, limitando le possibilità per un attaccante di sfruttare la predizione dei salti indiretti tra thread diversi.
- **IBPB** (Indirect Branch Predictor Barrier) crea una barriera tra il software che viene eseguito prima e dopo l'impostazione della barriera. Questo impedisce al software che viene eseguito prima della barriera di influenzare la predizione del ramo per il software che viene eseguito dopo.

Queste tecniche contribuiscono a ridurre il rischio di attacchi come Spectre, migliorando la sicurezza dei processori moderni contro gli attacchi di predizione dei salti indiretti. :contentReference[oaicite:5]index=5

#### 9.5 RowHammer: Attacco alla Memoria e le sue Mitigazioni

Il **RowHammer** è una vulnerabilità fisica che riguarda la memoria DRAM e sfrutta l'effetto di interferenza elettromagnetica tra celle di memoria adiacenti. Quando una riga di memoria viene attivata ripetutamente in modo intensivo, la fluttuazione di tensione risultante può causare un "flip" nei bit della riga vicina, anche se non è direttamente coinvolta nell'operazione. Questo fenomeno, noto come bit flipping, può essere sfruttato da un attaccante per compromettere la sicurezza dei dati memorizzati, alterando il contenuto di celle di memoria vicine e, quindi, manipolando informazioni sensibili.



**Meccanismo dell'attacco RowHammer** L'attacco RowHammer si basa su un principio semplice ma efficace: l'attaccante cerca di attivare ripetutamente una riga di memoria specifica, in modo che le fluttuazioni elettromagnetiche causino una modifica dei dati nelle righe adiacenti. Questo accade perché, durante l'attivazione di una riga, il segnale elettrico può propagarsi nelle righe vicine, causando una carica indotta in queste celle di memoria.

**Il RowHammer è particolarmente potente** perché sfrutta una caratteristica della memoria DRAM: l'alta densità e la vicinanza fisica delle celle di memoria. Le celle vicine possono essere influenzate da un'operazione di scrittura su una riga diversa, specialmente quando quest'ultima viene attivata ripetutamente. L'attacco non richiede l'accesso privilegiato alla memoria, ma può essere eseguito da un utente malintenzionato che sfrutta la capacità di manipolare l'accesso alle righe di memoria.

**Impatti di RowHammer** L'attacco RowHammer può avere gravi implicazioni per la sicurezza dei sistemi, in particolare in ambienti condivisi, come i sistemi cloud, dove più macchine virtuali (VM) sono eseguite sulla stessa architettura fisica. Se un attaccante è in grado di sfruttare RowHammer, potrebbe corrompere la memoria di un altro processo, rubare chiavi crittografiche o alterare i dati senza che il sistema operativo o le applicazioni possano rilevare l'intrusione.

**Mitigazioni contro RowHammer** Fortunatamente, esistono alcune contromisure per contrastare l'attacco RowHammer, sebbene nessuna di esse offra una protezione perfetta. Le soluzioni più comuni includono:

- **Error Correction Codes (ECC)**: Tecnica che rileva e corregge i bit flip causati dal RowHammer, efficace solo per pochi bit flip per parola di memoria. Non sufficiente se i bit flip sono multipli.
- **Memoria a bassa densità**: Le memorie DRAM con bassa densità o maggiore distanza tra le righe riducono il rischio di interferenze, rendendo più difficile per l'attaccante manipolare più righe contemporaneamente.
- **Refresh rate più alto**: Aumentare la frequenza di aggiornamento della memoria per ridurre il rischio di perdita di carica nelle celle di memoria, ma con impatti sui consumi energetici e le prestazioni.
- **Pseudo-Target Row Refresh (pTRR)**: Tecnica che rinfresca le righe vulnerabili della memoria per evitare bit flip, ma può aumentare i costi e la latenza.
- **Tecniche di partizionamento della memoria**: Il *Memory Partitioning* separa le righe di memoria, riducendo l'accesso non autorizzato, e può essere implementato hardware o software.
- **Virtualizzazione e isolamento hardware**: L'isolamento tra macchine virtuali (VM) riduce il rischio che un attaccante acceda alla memoria di altre VM.

## 9.6 Memory Performance Attacks: Attacchi alle Prestazioni della Memoria

Gli **attacchi alle prestazioni della memoria** mirano a sfruttare il comportamento non ottimale dei sistemi DRAM, in particolare durante la gestione della memoria in scenari multi-core. Questi attacchi sfruttano le caratteristiche di programmazione e accesso alla memoria, come la gestione dei buffer di riga e la concorrenza tra processi, per compromettere le prestazioni o, in alcuni casi, causare un Denial of Service (DoS) alla memoria.

**9.6.0.1 Denial of Service nei Sistemi Multi-Core** Nei sistemi multi-core, i pianificatori di memoria DRAM sono progettati per chip a singolo core, il che comporta inefficienze nelle operazioni di memoria quando più core tentano di accedere a risorse condivise. Ogni banco di memoria ha un buffer di riga e l'accesso alla memoria avviene attraverso questo buffer. Se due

o più thread accedono simultaneamente a diverse righe di memoria, il sistema potrebbe dover risolvere conflitti, causando ritardi nei tempi di accesso alla memoria. Questo tipo di attacco può portare a rallentamenti significativi e persino al blocco dei processi, creando un attacco DoS a livello di memoria.

**9.6.0.2 Funzionamento degli Attacchi a Memoria Multi-Banca** Le memorie DRAM moderne hanno più banchi, ciascuno con un buffer di riga. Il **piano di accesso alla memoria** si occupa di risolvere le richieste in base alla disponibilità di ciascun banco. In uno scenario ottimale, la schedulazione della memoria cerca di minimizzare i conflitti tra le righe, ma nei sistemi multi-core il conflitto tra i processi che accedono simultaneamente agli stessi banchi o righe può compromettere gravemente le prestazioni. Gli attacchi in questo contesto sfruttano l'accesso a righe di memoria che non sono ottimizzate per l'esecuzione parallela, causando inefficienze nel sistema.

**9.6.0.3 Scheduling della Memoria e Contesa tra Banchi** Il piano di accesso **FR-FCFS** (First-Ready, First-Come, First-Serve) ottimizza le richieste che si allineano con la riga già disponibile nel buffer, riducendo i conflitti. Tuttavia, non gestisce bene i conflitti tra banchi diversi, causando inefficienze, specialmente quando un thread con alta località di riga accede a righe vicine. Inoltre, poiché i banchi di memoria condividono un bus centrale, un thread che accede frequentemente alla memoria può ridurre le risorse disponibili per altri thread, creando condizioni di contesa che rallentano l'accesso e, in alcuni casi, possono portare a un Denial of Service (DoS). Gli attaccanti possono sfruttare queste inefficienze con tecniche di scheduling avanzate.

**9.6.0.4 Mitigazioni e Strategie di Ottimizzazione** La protezione contro gli attacchi di prestazione della memoria dipende principalmente da tecniche di ottimizzazione a livello hardware e software. Alcuni approcci includono:

- **Piani di accesso a memoria più avanzati**, come quelli che minimizzano i conflitti di accesso tra banchi.
- **Affinità CPU** (CPU affinity), che assicura che i thread che accedono frequentemente a determinati dati vengano eseguiti sullo stesso core per migliorare la località di riferimento.
- **Schedulazione intelligente delle richieste di memoria**, che ottimizza l'allocazione delle risorse di memoria per ridurre il rischio di conflitti tra i thread.
- **Memoria di tipo HBM (High Bandwidth Memory)** che riduce la latenza e le collisioni nell'accesso alla memoria.

Tuttavia, come per la maggior parte delle soluzioni hardware, esiste un compromesso tra prestazioni e sicurezza, quindi è necessario un bilanciamento tra l'implementazione di difese contro gli attacchi a prestazione della memoria e l'ottimizzazione per le prestazioni generali del sistema.

# 10 SYS\_05 - OS Security Principles

## 10.1 Introduzione

La sicurezza dei sistemi operativi rappresenta uno dei pilastri fondamentali della difesa informatica. L'obiettivo di questo modulo è comprendere come i sistemi operativi moderni — in particolare Unix e Linux — gestiscano la sicurezza a livello di utenti, processi, permessi e politiche di controllo degli accessi. Un sistema operativo sicuro deve assicurare che solo utenti legittimi possano accedere alle risorse e che ogni processo operi entro i limiti dei privilegi assegnati, prevenendo così l'abuso di poteri amministrativi o l'escalation di privilegi.

Il principio guida è quello di ridurre la superficie d'attacco, garantendo al contempo la piena funzionalità del sistema per gli utenti autorizzati. Questo implica la corretta gestione dell'autenticazione, dei permessi, dei privilegi, e l'adozione di politiche di controllo (DAC e MAC) e meccanismi modulari come i Linux Security Modules (LSM).

## 10.2 Identificazione e Autenticazione degli Utenti

L'accesso a un sistema operativo avviene tramite un meccanismo di autenticazione, tipicamente basato su password. Nel mondo Unix/Linux, le informazioni sugli utenti e le password sono conservate in due file distinti: `/etc/passwd` e `/etc/shadow`.

**10.2.0.1 /etc/passwd** Il file `/etc/passwd` contiene i dati pubblici degli utenti con il seguente formato:

```
username:passwd:UID:GID:full_name:directory:shell
```

Ad esempio:

```
user1:Npge08pfz4wuk:503:100:TheUser:/home/user1:/bin/sh
```

Qui `Np` rappresenta il salt e `ge08pfz4wuk` la password cifrata. Nei sistemi moderni, tuttavia, il campo della password è sostituito da una semplice `x`, e il valore cifrato è spostato in `/etc/shadow`, inaccessibile agli utenti non privilegiati.

**10.2.0.2 /etc/shadow** Il file `/etc/shadow` contiene le informazioni sensibili relative alle password e alle politiche di scadenza degli account. Per ragioni di sicurezza questo file è accessibile soltanto all'utente `root` (tipicamente con permessi 600 e proprietà `root:root`), mentre il corrispondente `/etc/passwd` rimane leggibile da tutti.

Il formato standard di una riga di `/etc/shadow` è il seguente:

```
username:password:lastchg:min:max:warn:inactive:expire:reserved
```

- **username:** nome dell'account utente.
- **password:** hash della password (ad es. `$6$salt$...` per SHA-512). Se contiene un carattere speciale come “!” o “\*” il login con password è disabilitato per quell'account.
- **lastchg** (spesso indicato come `ult`): numero di giorni trascorsi dal 1/1/1970 (Unix epoch) dall'ultimo cambio password. Viene usato come riferimento per le politiche di aging.
- **min** (nella tua versione `can`): intervallo minimo (in giorni) che deve trascorrere prima che l'utente possa cambiare di nuovo la password. Valore comune: 0 (nessuna restrizione) o 1.
- **max** (nella tua versione `must`): intervallo massimo (in giorni) di validità della password; dopo questo periodo l'utente deve cambiare password.

- **warn** (nella tua versione `note`): numero di giorni prima della scadenza in cui il sistema avvisa l'utente che la password sta per scadere.
- **inactive** (nella tua versione `exp`): numero di giorni che intercorrono dopo la scadenza della password dopo i quali l'account viene reso inattivo (disabilitato) se la password non viene aggiornata.
- **expire** (nella tua versione `disab`): data (in giorni dall'epoch) dopo la quale l'account è definitivamente disabilitato, indipendentemente dallo stato della password. Spesso lasciato vuoto se non si intende disabilitare l'account in una data fissa.
- **reserved**: campo riservato per usi futuri (di solito vuoto).

### 10.3 Gestione degli UID e dei Privilegi

In Unix, il nome utente è un'etichetta simbolica, mentre le vere autorizzazioni sono legate al numero identificativo (**UID**). Ogni processo è associato a tre differenti UID:

- **Real UID (RUID)**: identifica chi è l'utente effettivo;
- **Effective UID (EUID)**: determina cosa può effettivamente fare;
- **Saved UID (SUID)**: rappresenta chi può tornare ad essere.

Questo modello consente di cambiare temporaneamente i privilegi di un processo. Le chiamate di sistema `setuid()` e `seteuid()` permettono a processi privilegiati di modificare gli identificatori utente. La prima è irreversibile e sovrascrive tutti e tre gli UID, mentre la seconda consente un cambio temporaneo e reversibile del contesto di esecuzione.

**10.3.0.1 su e sudo** I comandi `su` e `sudo` sfruttano il meccanismo SUID per assumere privilegi elevati. Entrambi sono contrassegnati con il bit “setuid-root”, che permette loro di eseguire operazioni con i permessi del superutente. Mentre `su` cambia completamente l'identità utente, `sudo` esegue un singolo comando come `root`, dopo aver verificato l'autorizzazione nel file `/etc/sudoers`.

### 10.4 Principio del Minimo Privilegio

Il **Principle of Least Privilege** stabilisce che ogni entità del sistema — utente, processo o applicazione — debba poter accedere solo alle risorse strettamente necessarie alla sua funzione. Questo principio aumenta la stabilità, riduce la possibilità di sfruttamento di vulnerabilità e semplifica la verifica del comportamento delle applicazioni.

In pratica, ciò significa che un servizio non deve mai essere eseguito come `root` se non strettamente necessario, e che le operazioni privilegiate devono essere segmentate in processi specifici e confinati.

### 10.5 Controllo degli Accessi

Il controllo degli accessi definisce quali soggetti (utenti o processi) possano eseguire determinate azioni sugli oggetti del sistema (file, socket, dispositivi). L'obiettivo è limitare i danni derivanti da errori o comportamenti malevoli, applicando una politica di sicurezza che stabilisca chi può fare cosa.

Component	Description
Subject	The entity making the request (a process or user)
Action	The requested operation (read, write, execute)
Object	The target of the action (a file, network connection, hardware)

Figura 66: Access Control Fundamentals

### 10.5.1 Politiche di Sicurezza: DAC e MAC

Due principali modelli di sicurezza regolano il controllo degli accessi:

- **Discretionary Access Control (DAC)**: l'accesso è deciso dal proprietario della risorsa, che può modificare liberamente i permessi (classico modello Unix con **rwx**);
- **Mandatory Access Control (MAC)**: l'accesso è imposto dal sistema in base a politiche centrali e non può essere cambiato dagli utenti.

Il primo offre flessibilità, il secondo garantisce rigore e isolamento, essenziale in ambienti ad alta sicurezza.

### 10.5.2 POSIX ACL e Access Control Fine-Grained

Il modello **POSIX ACL (Access Control List)** rappresenta un'estensione del tradizionale schema **DAC (Discretionary Access Control)** tipico di Unix, introducendo un livello di granularità più fine nella gestione dei permessi. In un sistema Unix classico, i permessi di accesso ai file sono definiti da tre soli insiemi di diritti: **owner**, **group** e **others**. Con le ACL, invece, è possibile definire regole specifiche per più utenti e più gruppi, andando oltre la rigidità della tripla **rwx** standard.

Le ACL consentono quindi di specificare quali utenti o gruppi possano leggere, scrivere o eseguire un file, anche se non sono il proprietario o non appartengono al gruppo principale del file. Il sistema continua a basarsi su un modello **DAC** (cioè le decisioni d'accesso dipendono dalle scelte del proprietario), ma con la possibilità di definire eccezioni molto più precise.

#### 10.5.2.1 Esempio di ACL POSIX

Un esempio reale di ACL può essere il seguente:

```
# file: audio
# owner: gwurster
# group: audio
user::rwx
group::r-x
group:powerdev:r-x
mask::r-x
other::---
```

In questo esempio:

- Il proprietario del file (**gwurster**) ha tutti i permessi (**rwx**);
- Il gruppo principale **audio** può leggere ed eseguire (**r-x**);
- È stato aggiunto un gruppo aggiuntivo **powerdev** con gli stessi permessi;

- La **mask** (`r-x`) definisce il livello massimo di permesso che gli altri utenti e gruppi possono effettivamente ottenere;
- Gli utenti “other” non hanno alcun permesso (–).

**10.5.2.2 Meccanismo di valutazione delle ACL** Quando un processo tenta di accedere a un file protetto da ACL, il kernel segue una precisa sequenza di controlli per determinare se concedere o meno il permesso. La decisione si basa sulla corrispondenza tra l’identità (UID/GID) del processo e le voci presenti nella lista di controllo.

#### 10.5.2.2.1 Fase (i): Selezione della voce corrispondente

1. Se l’UID del processo coincide con quello del proprietario, si applicano i permessi dell’entry `user::`.
2. Altrimenti, se l’UID del processo corrisponde a uno degli utenti elencati in una `user:<nome>` specifica, quella entry determina l’accesso.
3. Se nessuna delle precedenti condizioni è vera, si verifica se uno dei GID del processo corrisponde al gruppo proprietario del file e la relativa entry `group::` contiene i permessi richiesti.
4. Se non è così, ma uno dei GID del processo corrisponde a un gruppo elencato in una entry `group:<nome>`, e quella entry concede i permessi richiesti, l’accesso viene determinato da quella voce.
5. Se invece uno dei gruppi del processo corrisponde a un gruppo menzionato, ma né la entry del gruppo proprietario né le altre entry di gruppo concedono i permessi richiesti, l’accesso è negato.
6. In tutti gli altri casi, l’accesso viene determinato dalla entry `other::`.

**10.5.2.2.2 Fase (ii): Decisione finale sull’accesso** Una volta individuata la entry corrispondente:

- Se la voce trovata è `user::` o `other::` e include i permessi richiesti, l’accesso è **concesso**.
- Se la voce è una `user:<nome>` o `group:<nome>` e include i permessi richiesti, allora il sistema controlla anche la **mask**: solo se la `mask` include gli stessi permessi, l’accesso è effettivamente **concesso**.
- In caso contrario, l’accesso è **negato**.

**10.5.2.3 Ruolo della Mask** La voce `mask::` ha un ruolo cruciale: essa rappresenta un limite massimo di permessi che può essere effettivamente applicato a tutte le entry di tipo gruppo o utente nominato. Anche se una voce ACL concede permessi più ampi, questi vengono ridotti al livello della maschera. In pratica, la **mask** agisce come un filtro: se la mask non contiene il bit di lettura, anche un gruppo che possiede `rwx` non potrà leggere il file.

## 10.6 Linux Capabilities

Nel modello Unix tradizionale, un processo è considerato **privilegiato** se ha `EUID = 0`, ossia se viene eseguito come `root`. Questo modello binario, seppur semplice, presenta un limite significativo: qualsiasi processo con `EUID = 0` ha accesso completo al sistema, potendo bypassare tutti i controlli di sicurezza del kernel. Le **Linux Capabilities** sono state introdotte per risolvere questo problema, frammentando i privilegi del superutente in unità indipendenti, chiamate

*capabilities*. Ogni capability rappresenta un potere specifico (ad esempio gestire la rete, montare filesystem, modificare l'orario di sistema, ecc.), e può essere assegnata o rimossa in modo granulare.

In sostanza, le capabilities trasformano l'amministrazione dei privilegi da un modello “tutto o niente” a un modello modulare, dove un processo può avere solo i permessi strettamente necessari.

**10.6.0.1 Esempi di capability** Alcune delle capabilities più comuni sono definite nel file sorgente del kernel `linux/include/linux/capability.h`:

Nome	Descrizione
CAP_CHOWN	Permette di cambiare la proprietà di un file, ignorando i controlli DAC.
CAP_KILL	Consente di inviare segnali ad altri processi.
CAP_NET_RAW	Autorizza l'uso di socket raw (es. per <code>ping</code> ).
CAP_NET_BIND_SERVICE	Permette di bindare porte TCP/UDP inferiori a 1024.
CAP_SYS_NICE	Consente di modificare la priorità dei processi.
CAP_SYS_TIME	Permette di impostare l'orologio di sistema.
CAP_SYS_ADMIN	È la capability più ampia, spesso considerata “il nuovo root”, poiché copre molte di queste.

**10.6.0.2 Implementazione nel kernel** Durante l'esecuzione di un'operazione privilegiata, il kernel verifica se il thread richiedente possiede la capability necessaria nel proprio insieme di capabilities attive. Le capabilities vengono gestite tramite apposite system call:

```
int capget(cap_user_header_t hdrp, cap_user_data_t datap);
int capset(cap_user_header_t hdrp, const cap_user_data_t datap);
```

Inoltre, la libreria `libcap` fornisce funzioni di più alto livello per la gestione da spazio utente.

Per funzionare correttamente, il filesystem deve supportare gli **attributi estesi** (`xattr`), poiché le capabilities possono essere associate direttamente agli eseguibili. Quando un processo esegue un file dotato di capability, eredita automaticamente quei privilegi al momento dell'`execve()`.

Tutti questi meccanismi sono pienamente supportati a partire dal kernel Linux 2.6.24.

**10.6.0.3 Insiemi di capability di un thread** Ogni thread in Linux mantiene quattro insiemi distinti di capabilities, che determinano i privilegi effettivi e potenziali del processo:

- **Permitted set (P)**: è il superset dei privilegi che il thread può potenzialmente utilizzare. Se una capability viene rimossa da questo insieme, non potrà mai più essere recuperata (nemmeno da un processo privilegiato).
- **Inheritable set (I)**: definisce le capabilities che possono essere ereditate dai nuovi processi tramite `execve()`, purché anche il file eseguibile consenta l'ereditarietà.
- **Effective set (E)**: contiene le capabilities attualmente attive e utilizzate dal kernel per i controlli di autorizzazione. Solo le capabilities in questo set sono operative in un dato momento.
- **Ambient set (A)**: introdotto nelle versioni più recenti del kernel, consente di mantenere alcune capabilities anche attraverso l'esecuzione di un programma non privilegiato. È utile per preservare privilegi minimi in catene di processi.

Quando un processo genera un figlio con `fork()`, quest'ultimo eredita una copia di tutti gli insiemi di capability del padre.

**10.6.0.4 File capabilities** Le capabilities possono essere associate anche ai file eseguibili, mediante attributi estesi del filesystem. Ciò consente a un programma di ottenere automaticamente specifici privilegi al momento dell'esecuzione, senza dover essere eseguito come `root`. Sono previsti due insiemi di capabilities associate al file:

- **Permitted**: capabilities che vengono sempre concesse al thread che esegue il file, indipendentemente dall'insieme ereditabile del processo.
- **Inheritable**: capabilities che vengono combinate (tramite un AND logico) con l'insieme ereditabile del processo per determinare quali privilegi passano nel nuovo contesto.

È inoltre presente un flag denominato **Effective**, che indica se le capabilities ottenute dal file devono essere immediatamente attive nel set **Effective** dopo l'`execve()`.

**10.6.0.5 Capability Bounding Set** Il **bounding set** agisce come un filtro di sicurezza globale per il sistema. Serve a limitare le capabilities che possono essere acquisite da un processo anche se il file eseguibile ne possiede di più. In pratica:

- durante l'esecuzione, il bounding set viene intersecato con le capabilities del file per determinare quali possono essere aggiunte al set **Permitted**;
- un processo non può mai aggiungere a sé stesso una capability che non sia contenuta nel proprio bounding set;
- rimuovere una capability dal bounding set la rende definitivamente inaccessibile fino al riavvio.

## 10.7 Linux Security Modules (LSM)

Il framework **Linux Security Modules (LSM)** rappresenta la base architetturale su cui si fondano le estensioni di sicurezza del kernel Linux. È stato introdotto per fornire un'infrastruttura modulare e generica che permetta l'integrazione di diversi modelli di controllo d'accesso, in particolare di tipo **MAC (Mandatory Access Control)**, senza dover modificare in modo invasivo il codice sorgente del kernel.

L'idea alla base di LSM nasce dall'esigenza di colmare una lacuna del modello tradizionale Linux, che nativamente implementa solo meccanismi di tipo **DAC (Discretionary Access Control)** e capabilities. Nel tempo, la comunità di sicurezza ha sviluppato molte proposte su come introdurre politiche MAC nel kernel, ma senza una struttura comune: LSM nasce proprio per fornire un punto di estensione unificato e standard.

**10.7.0.1 Concetto di base** Il framework LSM fornisce una serie di **hook di sicurezza**, ossia punti di intersezione nel codice del kernel dove è possibile eseguire controlli di accesso aggiuntivi. Questi hook sono chiamati ogni volta che un processo tenta di compiere un'operazione sensibile (ad esempio aprire un file, creare un socket, eseguire un processo o accedere a una memoria condivisa). In questo modo, i moduli LSM possono decidere se consentire o negare l'operazione in base a politiche definite dall'amministratore di sistema.

**10.7.0.2 Struttura generale** Il comportamento del framework è ispirato al modello classico del **Reference Monitor**, un concetto cardine della sicurezza dei sistemi operativi. Il Reference Monitor agisce come un "guardiano centrale" che controlla tutte le operazioni di accesso alle risorse del sistema secondo tre principi:

- **Tamper-resistant**: non deve essere aggirabile o modificabile da utenti o processi non autorizzati;

- **Always invoked:** deve essere sempre invocato per ogni richiesta di accesso, senza eccezioni;
- **Simple and analyzable:** deve essere progettato in modo semplice e verificabile, così da poter garantire la correttezza delle decisioni.

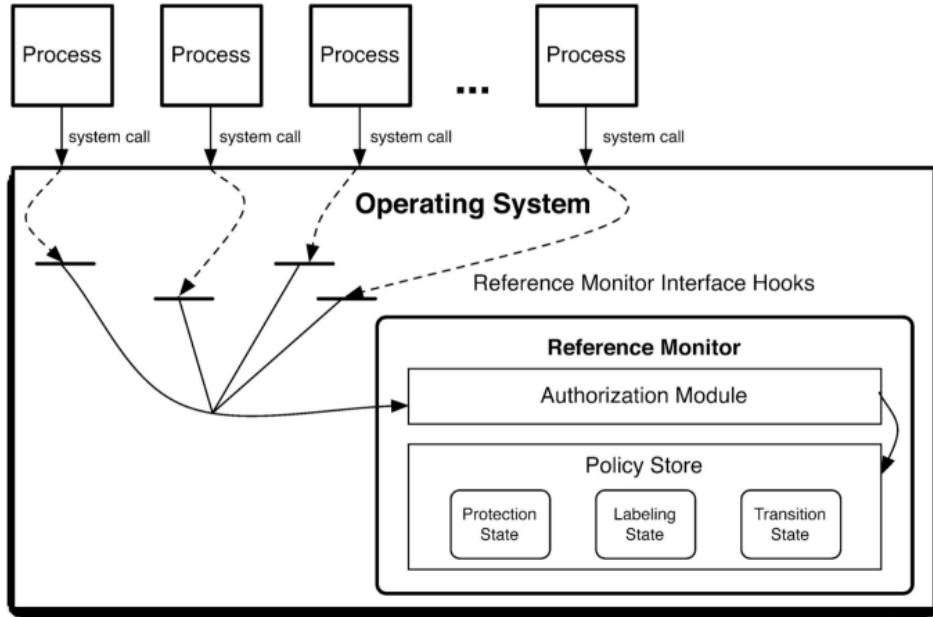


Figura 67: Reference Monitor Model

In Linux, LSM fornisce l'implementazione pratica di questo modello: tutte le richieste di accesso passano attraverso i suoi hook, che possono essere sovrascritti da moduli di sicurezza specifici (come SELinux, AppArmor, SMACK o Tomoyo).

**10.7.0.3 Funzionamento del framework** Durante la fase di boot, il kernel inizializza una struttura di funzione chiamata `security_ops`, che contiene i puntatori alle funzioni di sicurezza predefinite (i controlli DAC standard e le capabilities). Quando un modulo LSM viene caricato, può registrarsi tramite la funzione `register_security()` e sostituire i puntatori relativi alle operazioni che vuole controllare. Ad esempio:

```
int register_security(struct security_operations *ops) {
    if (verify(ops)) return -EINVAL;
    if (security_ops != &default_security_ops) return -EAGAIN;
    security_ops = ops;
    return 0;
}
```

In questo modo, il kernel rimpiazza le funzioni di sicurezza predefinite con quelle del modulo LSM caricato. Solo un modulo principale può essere attivo alla volta, anche se esistono meccanismi per la *stacking* (esecuzione in catena) introdotti nelle versioni più recenti.

**10.7.0.4 Hook di sicurezza** Gli hook LSM sono **restrittivi**: possono solo ridurre i privilegi concessi da DAC o capabilities, mai ampliare i diritti. In altre parole, un LSM può negare un accesso anche se DAC lo consentirebbe, ma non viceversa. Questo garantisce che i meccanismi

tradizionali rimangano sempre compatibili e che il framework mantenga un comportamento conservativo.

Gli hook sono definiti all'interno della struttura `security_operations`, che può essere personalizzata da ciascun modulo:

```
struct security_operations {
    int (*ptrace)(struct task_struct *parent, struct task_struct *child);
    int (*inode_setattr)(struct dentry *dentry, struct iattr *attr);
    ...
};
```

Ogni voce rappresenta un punto di controllo specifico nel flusso di esecuzione del kernel (ad esempio operazioni sui processi, sui file o sul networking).

**10.7.0.5 Categorie di hook** Gli hook LSM sono organizzati in diverse categorie, in base al tipo di risorsa o evento che controllano:

- **Task hooks:** controllano le operazioni legate ai processi (creazione, esecuzione, terminazione, segnalazioni, cambio UID/GID).
- **Program loading hooks:** gestiscono la sicurezza durante il caricamento di nuovi eseguibili (`execve()`, mapping di binari, interpreti, ecc.).
- **IPC hooks:** controllano l'accesso ai meccanismi di comunicazione interprocesso (code di messaggi, memoria condivisa, semafori).
- **Filesystem hooks:** verificano i permessi su file, directory, inode e attributi estesi.
- **Network hooks:** gestiscono la sicurezza a livello di rete (socket, pacchetti, dispositivi).

**10.7.0.6 Esempio: hook su un file system** Quando un processo tenta di creare una directory tramite una chiamata come:

```
mkdir("/home/user/docs", 0777);
```

il kernel attraversa diversi livelli:

1. Controlla i permessi DAC sul percorso;
2. Verifica le capabilities del processo;
3. Invoca l'hook LSM corrispondente, ad esempio `inode_mkdir()`, definito dal modulo attivo;
4. Se l'hook restituisce un errore (es. `-EPERM`), l'operazione viene negata e il file system non viene modificato.

Questo meccanismo consente di applicare controlli specifici e coerenti con le policy MAC, come nel caso di SELinux, che confronta le etichette di sicurezza di soggetto e oggetto prima di approvare l'operazione.

**10.7.0.7 Estensioni del kernel per LSM** Per integrare il framework nel kernel, molte strutture dati sono state estese con campi aggiuntivi che puntano a metadati di sicurezza. Tra queste troviamo:

Struttura kernel	Oggetto rappresentato
<code>task_struct</code>	Thread/processo
<code>linux_binprm</code>	Programma in fase di caricamento
<code>super_block</code>	Filesystem
<code>inode</code>	File, socket o pipe
<code>file</code>	File aperto
<code>sk_buff</code>	Buffer di rete (pacchetto)
<code>net_device</code>	Dispositivo di rete
<code>kern_ipc_perm</code>	Oggetti IPC (memoria condivisa, semafori, code)

Ogni modulo LSM può utilizzare questi puntatori per mantenere e consultare informazioni di sicurezza specifiche associate a ogni risorsa.

### 10.7.1 Esempi di Moduli MAC

**10.7.1.1 S.M.A.C.K. (Simplified Mandatory Access Control Kernel)** Il **Simplified Mandatory Access Control Kernel (SMACK)** è un modulo LSM che implementa un meccanismo di sicurezza di tipo **MAC** basato su etichette (*labels*) associate a soggetti e oggetti. Ogni processo (soggetto) e ogni risorsa (oggetto) del sistema possiede un'etichetta testuale — una semplice stringa — che rappresenta la sua identità di sicurezza. A differenza di SELinux, dove le etichette sono strettamente tipizzate e fanno parte di un modello complesso di domini e ruoli, in SMACK le etichette non hanno alcun significato semantico intrinseco: sono semplici stringhe confrontate tra loro secondo regole esplicite.

Un processo può accedere a un oggetto solo se esiste una regola che consente l'interazione tra le rispettive etichette. La forma generale di una regola di accesso è:

```
<subject-label> <object-label> <access>
```

dove `<access>` è espresso nel tradizionale formato Unix (es. `rwx`) e specifica i permessi consentiti per la coppia **soggetto-oggetto**. Se non esiste una regola corrispondente, l'accesso viene negato in modo predefinito, realizzando così un modello *default-deny*.

Le etichette sono memorizzate come **attributi estesi (xattr)** nei file e in altre risorse del sistema. Solo i processi dotati della capability `CAP_MAC_ADMIN` possono modificare queste etichette o le regole di accesso associate.

**10.7.1.2 Etichette di default** SMACK utilizza tre etichette predefinite, che semplificano la configurazione del sistema e ne definiscono il comportamento di base:

- `_ (floor)`: rappresenta il livello più basso; tutti i soggetti possono leggere o eseguire oggetti etichettati con `_`.
- `*` (*star*): indica un'entità completamente isolata; nessun soggetto può accedere a risorse etichettate con `*`, e tali soggetti non possono interagire con altri.
- `^ (hat)`: rappresenta un livello privilegiato; i soggetti etichettati con `^` possono accedere liberamente in lettura o esecuzione ad altre etichette.

Le tre etichette predefinite consentono di gestire in modo semplice i comportamenti più comuni:

- un oggetto etichettato con `*` è completamente protetto;
- un soggetto con etichetta `^` ha accesso ampio in lettura/esecuzione;
- un oggetto etichettato con `_` è leggibile o eseguibile da chiunque.

**10.7.1.3 SMACKFS** La configurazione del sistema SMACK avviene attraverso un **filesystem virtuale** dedicato, chiamato `smackfs`, montato nel percorso `/sys/fs/smack`. Questo pseudo-filesystem fornisce una serie di file speciali che consentono di consultare e modificare dinamicamente le regole di accesso e le etichette. Tra i più importanti troviamo:

- **access2**: permette di interrogare i permessi di accesso tra due etichette. Scrivendo una regola nel formato `<subject> <object> <access>` e leggendo la risposta, si può verificare se un'operazione è consentita.
- **change-rule**: consente di modificare i permessi associati a una specifica coppia di etichette.
- **load2**: serve per caricare nuove regole di accesso in blocco, tipicamente all'avvio del sistema.
- **onlycap**: elenca le etichette dei processi che possiedono la capability `CAP_MAC_ADMIN`; solo questi soggetti possono cambiare regole o etichette.
- **revoke-subject**: rimuove tutte le regole di accesso associate a un determinato soggetto.

Questi file rendono SMACK altamente configurabile senza la necessità di strumenti esterni: basta scrivere e leggere file di testo per modificare il comportamento di sicurezza del kernel.

**10.7.1.4 Caratteristiche e limiti** SMACK è apprezzato per la sua semplicità e leggerezza rispetto a sistemi più complessi come SELinux. Tuttavia, questa semplicità è anche la sua principale limitazione: il modello basato su stringhe e regole dirette non supporta costrutti avanzati come domini, ruoli o attributi contestuali. È quindi adatto a sistemi embedded, IoT o ambienti dove la configurazione deve essere essenziale ma efficace.

**10.7.1.5 Tomoyo Linux** Tomoyo adotta un approccio più leggibile, basato sui percorsi dei file anziché su etichette astratte. Ogni processo appartiene a un dominio definito dalla sequenza di eseguibili attraversati, e le ACL possono specificare permessi dettagliati per dominio, mantenendo un labeling sempre coerente.

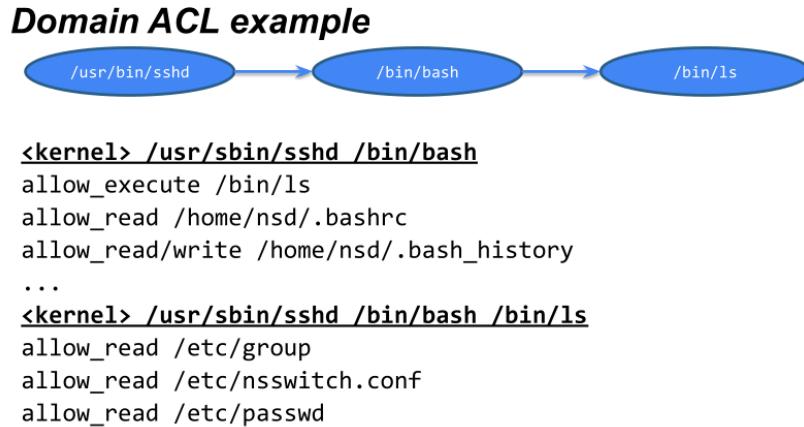


Figura 68: Domain ACL in Tomoyo Linux

**10.7.1.6 AppArmor** AppArmor applica una politica centrata sui task, dove ogni programma ha un profilo specifico che definisce i file, le capacità e le risorse accessibili. I profili sono scritti nel linguaggio AppArmor e caricati in `/etc/apparmor.d`. I processi non confinati seguono invece le regole DAC tradizionali.

**10.7.1.7 SELinux (Security-Enhanced Linux)** SELinux (Security-Enhanced Linux) è il modulo LSM più potente e complesso, nato da un progetto congiunto tra NSA e Red Hat per implementare un controllo d'accesso di tipo **MAC completo** all'interno del kernel Linux. Il suo obiettivo è garantire che ogni accesso a una risorsa (file, processo, socket, memoria, ecc.) avvenga in accordo a una **policy di sicurezza formale**, basata sul concetto di **security context**.

Ogni soggetto (processo) e oggetto (risorsa di sistema) è associato a un'etichetta di sicurezza, e ogni operazione tra due entità è autorizzata solo se esiste una regola esplicita che lo consente. SELinux, dunque, sostituisce il modello DAC predefinito (basato sul proprietario e sui permessi rwx) con un modello in cui le decisioni d'accesso sono dettate da regole globali definite da un amministratore di sistema di sicurezza.

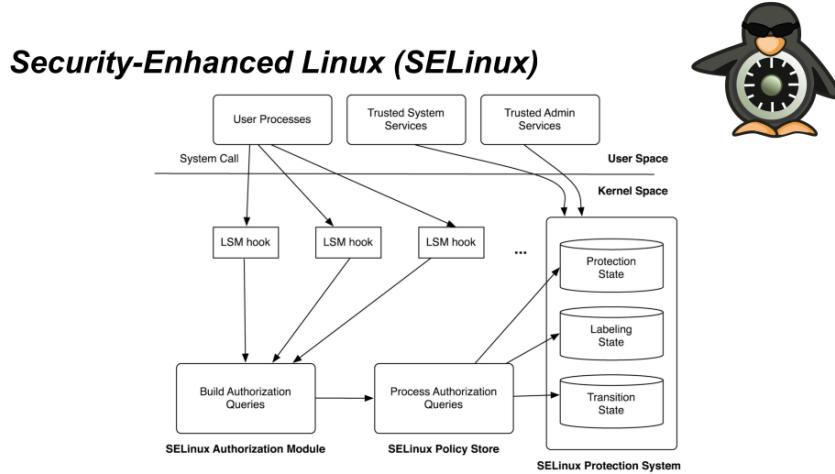


Figura 69: Security-Enhanced Linux (SELinux): mapping di soggetti e oggetti in base al loro contesto di sicurezza.

**10.7.1.8 Fase 1: dal kernel alla query di autorizzazione** Quando un processo effettua un'operazione (es. apre un file, crea un socket, o esegue un binario), il kernel intercetta la richiesta attraverso gli **hook LSM**. Questi parametri vengono convertiti in una *authorization query* che identifica:

- Il **soggetto** (processo) che effettua la richiesta, associato a un tipo di sicurezza (`subject_type`);
- L'**oggetto** (file, socket, ecc.) su cui si vuole agire, con un tipo e una classe (`object_type` e `object_class`);
- L'**operazione** richiesta (lettura, scrittura, esecuzione, ecc.).

A questo punto SELinux verifica se esiste una regola nella policy che consenta l'operazione richiesta.

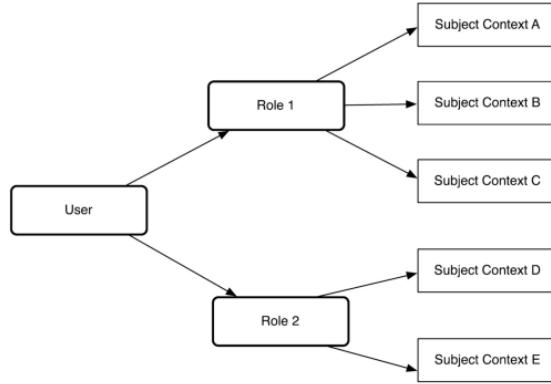


Figura 70: Security-Enhanced Linux (SELinux): fase 1 - generazione della query di autorizzazione.

**10.7.1.9 Fase 2: ricerca della policy SELinux** Le regole SELinux sono espresse con una sintassi formale del tipo:

```
allow <subject_type> <object_type>:<object_class> <operation_set>;
```

Ad esempio:

```
allow user_t passwd_exec_t:file execute;
allow passwd_t shadow_t:file { read write };
```

La prima regola permette a un processo con tipo `user_t` di eseguire un file etichettato come `passwd_exec_t`; la seconda consente a un processo `passwd_t` di leggere e scrivere file con etichetta `shadow_t`. In assenza di una regola esplicita, l'accesso è negato: SELinux opera quindi secondo una politica **default-deny**.

**10.7.1.10 Stato di etichettamento (Labeling State)** Il meccanismo di labeling in SELinux mappa tutte le risorse del sistema (file, dispositivi, processi, socket) su etichette di sicurezza permanenti, chiamate **security context**. Ogni contesto è composto da tre campi:

```
user:role:type[:level]
```

Esempio: `system_u:object_r:shadow_t:s0`

Il campo `user` rappresenta l'utente SELinux, `role` il ruolo assegnato (utile per il Role-Based Access Control), mentre il campo `type` è il più importante ed è quello utilizzato nel **Type Enforcement**, il principale meccanismo di policy. Il livello (`level`) è opzionale e viene usato nei sistemi multilevel (MLS).

Le policy di labeling specificano come assegnare automaticamente etichette a nuovi file e processi. Ad esempio:

```
/etc/shadow.*      system_u:object_r:shadow_t:s0
/etc/*.*           system_u:object_r:etc_t:s0
```

In questo modo, ogni nuovo file creato nel percorso `/etc` riceverà un tipo coerente con la destinazione, garantendo una separazione logica delle risorse di sistema.

**10.7.1.11 Stato di transizione (Transition State)** Oltre al labeling statico, SELinux gestisce anche le **transizioni di tipo**, che stabiliscono come devono cambiare le etichette dei soggetti o degli oggetti in seguito a determinate operazioni. Esistono due tipi principali di transizione:

**10.7.1.11.1 1. Transizione di file** Definisce come viene assegnata l'etichetta a un nuovo file creato da un processo. Ad esempio:

```
type_transition passwd_t etc_t:file shadow_t;
```

Con questa regola, un processo con etichetta `passwd_t` che crea un file in una directory etichettata come `etc_t` produrrà un file con tipo `shadow_t`. Questo comportamento è essenziale per mantenere la coerenza delle etichette nei file di sistema sensibili (es. `/etc/shadow`).

**10.7.1.11.2 2. Transizione di processo** Stabilisce come cambia l'etichetta di un processo quando esegue un certo programma. Ad esempio:

```
type_transition user_t passwd_exec_t:process passwd_t;
```

Quando un utente con tipo `user_t` esegue il binario `passwd` (etichettato come `passwd_exec_t`), il nuovo processo generato assume il tipo `passwd_t`. In questo modo, il programma viene confinato in un dominio sicuro che gli consente di accedere solo ai file strettamente necessari (ad esempio `/etc/shadow`) senza ereditare tutti i privilegi dell'utente che lo ha avviato.

## 10.8 Boot Time Security

La fase di avvio è estremamente critica poiché rappresenta il momento in cui il sistema carica kernel, moduli e servizi. Un attacco in questa fase può compromettere tutto il sistema prima dell'applicazione delle policy di sicurezza.

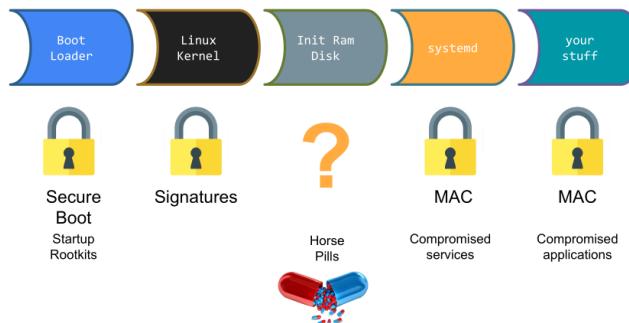


Figura 71: How your computer boots

**10.8.0.1 Funzionamento del Ramdisk durante il Boot** Durante la fase iniziale dell'avvio di un sistema Linux, il **ramdisk** (o *initramfs/initrd*) svolge un ruolo fondamentale per la preparazione dell'ambiente necessario al caricamento del sistema operativo vero e proprio. Il ramdisk è un piccolo filesystem temporaneo caricato in memoria dal bootloader, che contiene gli script e i moduli indispensabili per montare la root reale e avviare il processo `init`.

In un sistema sano e non compromesso, il flusso di esecuzione del ramdisk segue una sequenza ben definita di passaggi:

- Caricare i moduli necessari:** vengono caricati i moduli del kernel indispensabili per il riconoscimento dei dispositivi hardware (es. driver del filesystem, controller SATA, moduli per la rete o per i dischi NVMe).
- Gestire gli eventi di hotplug:** il sistema risponde agli eventi generati da dispositivi collegati dinamicamente durante la fase di boot (ad esempio rilevamento di periferiche USB o partizioni di disco).

3. **Eseguire cryptsetup (opzionale):** se il sistema adotta cifratura a livello di disco, il ramdisk gestisce la fase di sblocco dei volumi crittografati prima del montaggio della root.
4. **Trovare e montare la root filesystem:** viene identificato il dispositivo che ospita la partizione principale del sistema (/) e ne viene eseguito il montaggio. Questo è uno dei momenti più critici, poiché da qui il controllo passa al sistema reale.
5. **Pulire l'ambiente temporaneo:** una volta montata la root reale, il contenuto del ramdisk (inizialmente residente in memoria) viene smontato e liberato per ridurre il consumo di RAM.
6. **Eseguire init:** infine, viene lanciato il processo /sbin/init (o equivalenti come systemd), che avvia la vera sequenza di bootstrap del sistema operativo.

Questi sei passaggi rappresentano il comportamento atteso di un **ramdisk legittimo**. Qualsiasi deviazione da questa sequenza — ad esempio l'inserimento di codice non autorizzato o la modifica di script di inizializzazione — può essere sintomo di un compromesso del processo di boot. Ed è proprio su queste deviazioni che si basano attacchi come le *Horse Pills*, in grado di eseguire codice malevolo già nella fase di avvio, prima ancora che i meccanismi di sicurezza del kernel siano operativi.

**10.8.0.2 Horse Pills** L'attacco denominato **Horse Pills** rappresenta una forma particolarmente insidiosa di compromissione del sistema, poiché avviene durante la fase di **boot**, quando la catena di fiducia non è ancora completamente stabilita. In particolare, l'attacco prende di mira il **ramdisk** (o **initramfs/initrd**), che viene caricato in memoria prima del sistema operativo vero e proprio e che contiene gli script necessari per il montaggio del filesystem principale e l'avvio del processo **init**.

Un ramdisk compromesso può contenere moduli, script o binari alterati che si eseguono con privilegi totali, ottenendo così il controllo completo del sistema prima che vengano avviate le difese tradizionali (come SELinux, AppArmor o i meccanismi PAM). Da quel momento, l'attaccante può manipolare il comportamento del kernel, iniettare codice, installare backdoor o persistere in modo invisibile.

**10.8.0.3 Cosa può fare un ramdisk infetto** Una volta eseguito, un ramdisk malevolo può replicare tutte le operazioni di un initrd legittimo, ma con aggiunte o sostituzioni di comandi che modificano il comportamento del sistema. Tra le operazioni che un *Horse Pill* può eseguire:

- **Caricare moduli kernel** o librerie modificati, compromettendo i meccanismi di sicurezza a livello di spazio kernel;
- **Gestire la cifratura (cryptsetup)** in modo malevolo, esfiltrando password o chiavi prima di montare il filesystem cifrato;
- **Montare e manipolare il root filesystem (/)** per inserire file o alterare binari di sistema prima che il vero OS prenda il controllo;
- **Enumerare e modificare i thread del kernel** per inserire processi fittizi o mascherare moduli caricati;
- **Creare nuovi namespace e PID namespaces** usando chiamate come `clone(CLONE_NEWPID, CLONE_NEWNS)`, isolando i processi malevoli dal resto del sistema e rendendoli invisibili ai normali strumenti di analisi;
- **Eseguire fork() multipli** per installare hook o aprire **backdoor persistenti**, ad esempio intercettando aggiornamenti futuri dell'initrd o lanciando shell nascoste;

- **Rimontare filesystem critici** come `/proc` e creare **thread kernel falsi** per simulare processi legittimi;
- **Ripulire il ramdisk e rieseguire init**, completando il boot in modo apparentemente normale, così da non lasciare indizi immediati della compromissione.

In sostanza, l'attacco “Horse Pills” è una forma di **rootkit pre-boot**, che agisce prima della transizione dallo spazio d'inizializzazione a quello del sistema operativo vero e proprio, rendendo la rilevazione estremamente difficile.




---

Figura 72: Horse Pills: compromissione del ramdisk durante la fase di boot.

**10.8.0.4 Mitigazioni e strategie di difesa** Le contromisure contro gli attacchi Horse Pills si dividono in due categorie: **rilevazione** e **prevenzione**. Poiché il ramdisk è parte critica del processo di avvio, la sua compromissione può essere individuata solo tramite controlli di integrità e monitoraggio mirato.

#### 10.8.0.4.1 1. Rilevazione

- **Analisi dei namespace:** esaminare i link in `/proc/<pid>/ns` per identificare configurazioni di namespace anomale o isolate.
- **Verifica dei processi kernel:** analizzare i processi con `ppid != 0` che potrebbero mascherarsi come thread del kernel.
- **Audit e verifiche esterne:** effettuare controlli periodici e audit di sicurezza da fonti esterne al sistema (es. strumenti di forensics live).
- **Monitoraggio dell'integrità:** utilizzare strumenti come **AIDE** o **Tripwire** per rilevare modifiche al contenuto dell'initrd e ai file critici del sistema.
- **Verifica dei moduli kernel:** abilitare meccanismi di validazione come **dm-verity**, che consente al kernel di controllare la firma e l'integrità dei moduli caricati.
- **Analisi dei log:** usare sistemi di log analysis (es. **OSSEC**, **Splunk**) per identificare comportamenti sospetti nella sequenza di avvio.
- **Analisi della memoria:** strumenti come **Volatility** consentono di eseguire ispezioni in memoria viva per individuare moduli o processi iniettati.

- **Regole di auditd:** configurare **auditd** per monitorare l'accesso ai file critici e ai comandi di boot (es. `modprobe`, `insmod`).
- **Monitoraggio in tempo reale:** utilizzare strumenti come `sysdig` o `strace` per tracciare le chiamate di sistema durante il boot e identificare anomalie.

#### 10.8.0.4.2 2. Prevenzione

- **Limitare la creazione di ramdisk in produzione:** evitare che i sistemi operativi assemblino o rigenerino initrd direttamente in ambienti live, riducendo così il rischio di inserimento di codice malevolo.
- **Applicare la verifica delle firme digitali:** usare **Secure Boot** o meccanismi di firma delle immagini di initrd per assicurarsi che vengano caricati solo ramdisk autorizzati.
- **Controllare l'origine delle immagini di boot:** mantenere repository firmati e monitorare la catena di aggiornamento del kernel.
- **Separare la fase di build dalla fase di esecuzione:** costruire i ramdisk in ambienti controllati (build server) e distribuirli in sola lettura.

### 10.9 Sicurezza di Rete a Livello OS

A livello di sistema operativo, la sicurezza di rete si basa su meccanismi di controllo dell'accesso ai servizi in ascolto. Storicamente, Unix ha introdotto il concetto di “super-server” (`inetd` e `xinetd`), che gestiscono l'attivazione dei servizi su richiesta, e strumenti come `tcpd` per filtrare gli accessi in base all'indirizzo IP.

**10.9.0.1 inetd e xinetd** Il demone `inetd` ascolta su porte specifiche e, alla ricezione di una connessione, avvia il servizio corrispondente reindirizzando i socket standard (`stdin`, `stdout`, `stderr`). Il suo successore, `xinetd`, aggiunge controllo per indirizzo IP, finestre temporali di accesso, limitazioni sul numero di connessioni e log dettagliati. Le configurazioni sono in `/etc/inetd.conf` e `/etc/xinetd.conf`.

**10.9.0.2 TCP Wrappers e Reverse DNS** Il meccanismo noto come **TCP Wrappers** (programma comunemente chiamato `tcpd`) è una forma storica di controllo degli accessi a livello applicazione usata in congiunzione con i demoni gestiti da `inetd/xinetd`. Il suo obiettivo è semplice: intercettare una connessione in ingresso e decidere, in base a regole amministrative, se inoltrare la richiesta al servizio oppure rifiutarla.

**10.9.0.3 File di configurazione** TCP Wrappers utilizza due file di configurazione principali:

- `/etc/hosts.allow` — regole che consentono l'accesso (valutate per prime);
- `/etc/hosts.deny` — regole che negano l'accesso (valutate solo se nessuna regola in `hosts.allow` ha già concesso l'accesso).

La sintassi generale di una riga è:

```
daemon_list : client_list [ : shell_command ]
```

Dove:

- `daemon_list` indica il servizio o l'insieme di servizi (es. `sshd`, `ALL`);
- `client_list` indica gli host/client ammessi o negati (può essere un indirizzo IP, un network, un nome di host, un dominio con prefisso punto `.example.com`, un netgroup con `@`, ecc.);

- opzionalmente si può aggiungere un comando shell che viene eseguito quando la regola corrisponde.

#### 10.9.0.4 Esempi pratici

```
# permetti solo localhost per tutti i demoni
ALL: 127.0.0.1

# permetti sshd dal /24 di rete interna e dai nomi sotto example.com
sshd: 192.0.2.0/24, .example.com

# se nessuna allow ha concesso l'accesso, rifiuta tutto
ALL: ALL
```

**10.9.0.5 Ordine e logica di valutazione** La regola operativa è semplice ma importante:

1. il demone TCP wrapper consulta `/etc/hosts.allow`; se trova una voce che corrisponde, l'accesso è **concesso** e la valutazione termina;
2. se non c'è corrispondenza, viene consultato `/etc/hosts.deny`; se trova una voce corrispondente, l'accesso è **negato**;
3. in assenza di corrispondenze in entrambi i file l'accesso è **concesso** per default (da qui l'importanza della regola esplicita `ALL: ALL` in `hosts.deny` per un comportamento *deny-by-default*).

**10.9.0.6 Reverse DNS Tampering: il problema** Molte voci in `hosts.allow` e `hosts.deny` possono essere espresse con nomi host o domini (es. `.example.com`). Per risolvere questi nomi, `tcpd` fa ricorso al DNS: normalmente esegue una **reverse lookup** (PTR) sull'indirizzo IP del client per ottenere un nome d'host, e poi una **forward lookup** su quel nome per verificare che torni all'indirizzo IP originale. Questa doppia verifica (PTR → nome, poi nome → A/AAAA) è pensata per evitare falsificazioni semplici.

Il problema del *Reverse DNS Tampering* si presenta quando un attaccante riesce a manipolare le risposte DNS inverse (PTR) — o controlla la zona PTR del proprio blocco IP — e imposta un PTR che corrisponde a un nome autorizzato. Se la verifica forward/reverse non viene eseguita o se il DNS sottostante è compromesso (per esempio la zona PTR non è firmata o è sotto il controllo del malintenzionato), allora un host malevolo può farsi passare per un host autorizzato usando solo la manipolazione del PTR.

#### 10.9.0.7 Esempio di attacco semplificato

```
# l'amministratore permette .trusted.example.com su sshd
sshd: .trusted.example.com

# l'attaccante controlla la zona PTR del suo IP e imposta:
10.2.0.192.in-addr.arpa. PTR evil.trusted.example.com.

# se non viene effettuata una forward-check corretta, tcpd potrebbe accettare
# la connessione credendo che l'IP appartenga a trusted.example.com.
```

**10.9.0.8 Contromisure pratiche** Per ridurre il rischio derivante dal Reverse DNS Tampering e, più in generale, dalla fiducia nelle risposte DNS:

- **Eseguire la doppia verifica (reverse + forward):** assicurarsi che il software effettui la forward-check (nome ottenuto da PTR deve risolvere all'IP originale). TCP Wrappers storicamente lo fa; tuttavia, non è infallibile se l'attaccante controlla sia la PTR che la A record.
- **Preferire ACL basate su indirizzi IP o reti note** (meno dipendenti da DNS) quando possibile: usare reti, indirizzi pubblici o range ASN, non solo nomi host.
- **Usare firewall a livello kernel (iptables/nftables) o filtri a livello di rete** per bloccare connessioni indesiderate prima che raggiungano il demone applicativo.
- **Impostare policy di default-deny:** adottare ALL: ALL in /etc/hosts.deny e specificare esplicitamente solo ciò che è necessario in /etc/hosts.allow.
- **Ridurre la fiducia nel DNS non firmato:** quando possibile, utilizzare infrastrutture DNS con DNSSEC per le zone autoritative e per quelle inverse; la firma delle zone rende più difficile la manipolazione delle risposte.
- **Usare meccanismi di autenticazione addizionali:** ad esempio autenticazione a chiave (SSH keys), sistemi di autenticazione centralizzata (RADIUS/LDAP) o PAM, che non si basano esclusivamente sul nome restituito dal DNS.
- **Limitare le deleghe PTR:** essere cauti nel delegare la gestione delle zone inverse a terze parti; preferire la gestione diretta per quei blocchi IP sensibili.
- **Log e monitoraggio:** abilitare logging dettagliato per tcpd/xinetd e correlare con i log DNS per individuare incongruenze (PTR che cambiano frequentemente, mismatch forward/reverse).
- **Considerare alternative moderne:** TCP Wrappers è un meccanismo storico e presenta limiti (ad es. scarsa integrazione con IPv6 in vecchie implementazioni, gestione limitata delle policy). In molti contesti è preferibile usare strumenti di controllo accessi più moderni (firewall a livello host, SELinux/AppArmor per controllo applicativo, sistemi di autenticazione centralizzati).

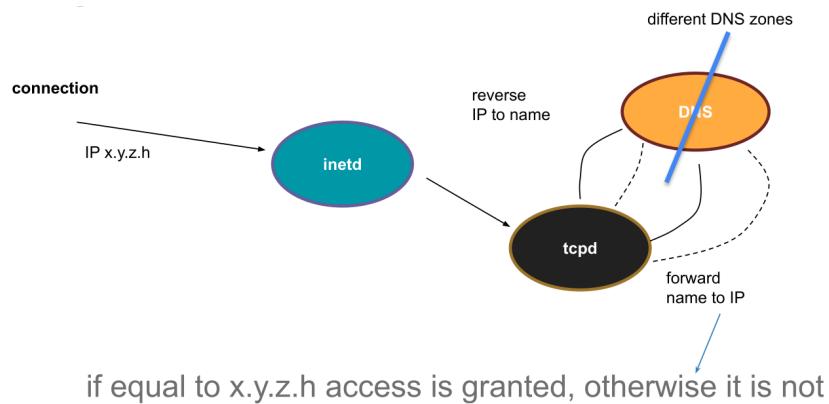


Figura 73: Esempio di reverne dns tampering

**10.9.0.9 Conclusione** La sicurezza del sistema operativo non è data da un unico meccanismo, ma dall'integrazione di più livelli di difesa: autenticazione robusta, controllo dei privilegi, politiche d'accesso rigorose e monitoraggio costante del comportamento del sistema. Solo un approccio multilivello, che rispetti il principio del minimo privilegio e l'integrità della catena di boot, può garantire la reale affidabilità di una piattaforma.

## 11 SYS\_06 - Antivirus

Un **antivirus** è un software che opera in background con l'obiettivo di **rilevare, prevenire e rimuovere** software malevolo. Il suo scopo principale è la *prevenzione*, ma nel caso in cui il malware riesca a penetrare nel sistema, deve essere in grado di **disinfettare i programmi infetti** e ripulire il computer.

Il primo esempio di virus informatico fu il **Creeper** (1971), creato da Bob Thomas alla BBN: si diffondeva nella rete ARPANET mostrando il messaggio “*I'm the Creeper, catch me if you can!*”. Poco dopo nacque **Reaper**, il primo programma antivirus, progettato per individuare e rimuovere Creeper: un esempio precoce del gioco “gatto e topo” tra virus e software di sicurezza.

### 11.1 Problemi principali degli antivirus

Gli antivirus affrontano diverse difficoltà:

- i malware cercano attivamente di nascondersi e auto-proteggersi, implementando tecniche di **evasione** e usando **funzionalità non documentate** dei sistemi operativi;
- la **superficie di attacco** dei sistemi moderni è enorme: include applicazioni, servizi, OS, email e rete;
- ogni giorno vengono rilasciate centinaia o migliaia di nuove varianti di software malevolo;
- l'obiettivo del malware è passato dal puro interesse tecnico al **profitto economico** o allo **spionaggio**.

### 11.2 Evoluzione degli antivirus

In origine gli antivirus erano semplici scanner a riga di comando basati su **pattern matching**. Oggi includono firewall, moduli di protezione web e componenti residenti che monitorano file e processi in tempo reale. A causa della quantità di nuovi campioni ricevuti giornalmente, le tecniche di sola firma non bastano più: vengono quindi impiegate **euristiche e analisi comportamentali**.

### 11.3 Componenti di un antivirus

Un antivirus moderno è composto da più moduli cooperanti:

- **Kernel dell'antivirus**: processo residente in memoria, avviato allo startup. È responsabile delle funzioni core e dei meccanismi di **self-protection** (es. intercetta segnali come SIGKILL per evitare la terminazione).
- **Scanner**: controlla i file creati o modificati, sfruttando i meccanismi di *publish/subscribe* del SO.
- **Filter driver per file system e rete**: analizzano il traffico prima che venga scritto o consegnato, spesso implementati con **eBPF**.
- **Database delle firme**: contiene hash, checksum o fingerprint di malware conosciuti; viene aggiornato frequentemente.
- **Unpacker**: analizza file compressi o cifrati, poiché molti malware sono “impacchettati”.
- **Interpreti di formati**: moduli in grado di comprendere strutture complesse (PE, ELF, PDF, DOCX, PNG, JPG), spesso basati su conoscenza parziale dei formati.
- **Emulatori**: ambienti sandbox per analizzare dinamicamente il comportamento di binari sospetti.

- **Meccanismi di autoprotezione:** come ASLR, DEP e anti-debugging.

## 11.4 Esempi di packing

I malware spesso ricorrono al **packing** per ostacolare l'analisi statica: il payload reale viene compresso, cifrato o trasformato in una rappresentazione intermedia e accompagnato da uno *stub/unpacker* che, a runtime, ripristina il codice originale in memoria. Di seguito sono spiegati con maggior dettaglio gli indicatori più utili per riconoscere un eseguibile impacchettato e come interpretarli.

**11.4.0.1 1. Poche o nessuna importazione nella IAT (Import Address Table)** Un eseguibile legittimo normalmente importa funzioni di librerie di sistema (es. kernel32.dll, libc, ecc.). I packer spesso creano un eseguibile con una IAT minimale (perché l'unpacker esegue la risoluzione dinamica delle API dopo lo *unpack*). Pertanto:

- **Indicatore:** IAT con poche voci o assente.
- **Strumenti:** `readelf -d`, `objdump -p` (ELF), `pefile` / `CFF Explorer` / `dumpbin` (PE).
- **Interpretazione:** sospetto di packing quando l'eseguibile non importa funzioni di libreria comuni o importa solamente poche funzioni “bootstrap”.

**11.4.0.2 2. Nomi di sezioni non standard o incoerenti** I packer tendono a rinominare o creare sezioni con nomi strani (es. `.upx`, `.vmp0`, sezioni con nomi casuali) oppure mettono tutto in poche sezioni generiche.

- **Indicatore:** sezioni con nomi sconosciuti o ripartizione non usuale di `.text`, `.data`, `.rdata`.
- **Strumenti:** `readelf -S`, `objdump -h`, `peheader/PEview`.
- **Interpretazione:** naming anomalo spesso correlate a packer noti; attenzione però ai compiler/linker proprietari che possono produrre nomi non standard.

**11.4.0.3 3. Sezioni piccole su disco ma grandi in memoria** I packer memorizzano il codice compresso in disco (raw size piccola) e lo espandono in memoria al momento dell'esecuzione (virtual size grande).

- **Indicatore:** grande differenza tra `raw_size` (su file) e `virtual_size` (in memoria) per una sezione.
- **Strumenti:** `readelf -S`, `pefile`, `radare2`.
- **Interpretazione:** forte indizio di packing / unpacking runtime.

**11.4.0.4 4. Poche stringhe leggibili** Poiché il payload è spesso compresso o cifrato, nell'eseguibile su disco ci sono poche stringhe ASCII/Unicode riconoscibili (URL, user-agent, messaggi di debug).

- **Indicatore:** basso numero di stringhe significative trovate con `strings`.
- **Strumenti:** `strings`, `floss` (per stringhe de-obfuscated), analisi con `rabin2` o `binwalk`.
- **Interpretazione:** packing o cifratura del contenuto; attenzione a false positive (ad esempio eseguibili minimizzati).

**11.4.0.5 5. Sezioni con permessi RWX** Sezioni marcate Read-Write-Execute (RWX) sono sospette perché permettono di scrivere codice in memoria e quindi eseguirlo — comportamento tipico dell'unpacker che scrive il payload in memoria e salta lì.

- **Indicatore:** presenza di sezioni con permessi RWX o pagine mappate RWX a runtime.
- **Strumenti:** `readelf -l` (ELF), **Process Explorer**, **VMMMap**, oppure con un debugger vedere le mappature di memoria.
- **Interpretazione:** RWX è raro in software benigno moderno (che preferisce W + X separati per mitigazioni), è quindi un forte campanello d'allarme.

**11.4.0.6 6. Salti o chiamate a registri/indirizzi anomali** I packer/unpacker spesso usano indirezioni (call/jmp su registri, valori calcolati, table-driven jumps) per trasferire il controllo al codice unpacked.

- **Indicatore:** molte istruzioni del tipo `jmp eax`, `call rax`, o calcoli di indirizzo complessi nelle prime regioni di codice.
- **Strumenti:** disassembler (IDA, Ghidra, radare2), `objdump -d`.
- **Interpretazione:** pattern tipico dello stub unpacker che risolve indirizzi e salta al codice decompresso.

## 11.5 Approccio operativo alla rilevazione

1. **Prima ispezione statica rapida:** `strings`, `readelf/objdump` o PE tools per IAT e sezioni anomale.
2. **Heuristics/ML:** n-grams di byte, istogrammi di opcode o grafi di chiamata per confrontare con dataset di benign/malware (utile quando le firme falliscono).
3. **Analisi dinamica controllata:** eseguire il sample in sandbox/emulatore per osservare unpacking runtime — monitorare la creazione di memoria RWX, le Write+Execute transition e le API di allocazione (es. `VirtualAlloc`, `mmap` con `PROT_EXEC`).
4. **Fingerprinting dei packer:** molti packer lasciano artefatti riconoscibili (sezioni .upx, firme nello stub). Usare DB di packer per matching.

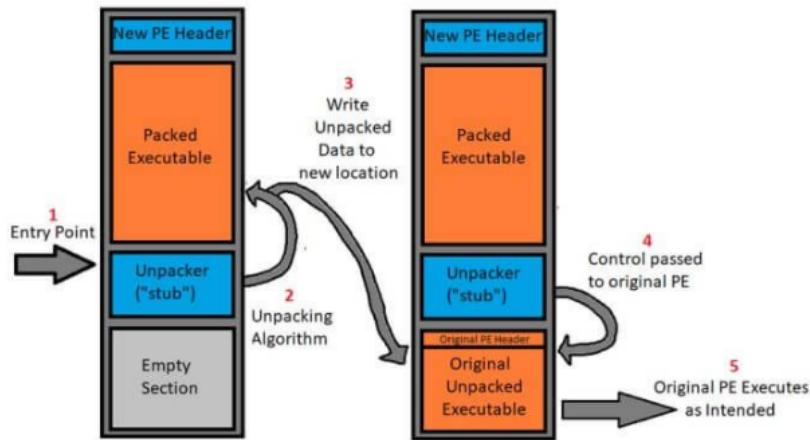


Figura 74: Processo di unpacking di un eseguibile: l'unpacker estraile il codice originale e trasferisce il controllo al programma legittimo.

## 11.6 Tecniche di evasione

Le tecniche di evasione mirano a rendere inefficace la firma dell'antivirus:

- aggiunta di **codice inutile o trasposto (spaghetti code)**;
- uso di **polimorfismo**, dove ogni infezione genera una versione leggermente diversa del codice;
- **metamorfismo**, in cui il virus si ricompila in una forma logicamente equivalente ma sintatticamente diversa, alterando anche il proprio motore di cifratura.

Figura 75: Evasion tramite **ineffective code sequences**: inserimento di codice inutile per alterare la firma.

Figura 76: Evasion tramite **code transposition**: riordino delle istruzioni per creare “spaghetti code”.

## 11.7 Analisi statica e comportamentale

La rilevazione del malware può essere condotta secondo due approcci principali: **analisi statica** (senza esecuzione) e **analisi comportamentale** (con esecuzione controllata o emulata). Le due tecniche sono complementari: la prima punta a riconoscere le firme del codice, la seconda ne studia il comportamento.

**11.7.0.1 Analisi statica** L’analisi statica si basa sull’esame del codice o dei binari senza eseguirli. È utile per individuare minacce note o pattern sospetti in modo rapido e sicuro. Le principali tecniche includono:

- **Disassembly:** conversione del codice binario in istruzioni assembly leggibili. Permette di osservare la struttura del programma (funzioni, loop, indirizzi di salto) e individuare pattern tipici di exploit o unpacker.
- **Analisi delle stringhe:** ricerca di sequenze ASCII o Unicode che possano rivelare domini di comando e controllo, URL, percorsi sospetti o messaggi diagnostici. Strumenti comuni sono **strings** o **floss** (per stringhe de-offuscate).
- **Ispezione degli header:** controllo dei metadati del file (es. intestazioni PE o ELF) per verificare incongruenze o campi modificati. Le intestazioni possono rivelare compilatori non standard o sezioni aggiuntive tipiche del packing.
- **Regole YARA:** rappresentano una forma evoluta di pattern matching. Ogni regola definisce un insieme di stringhe, espressioni regolari e condizioni logiche usate per identificare varianti di malware appartenenti alla stessa famiglia. YARA viene spesso definito il “coltellino svizzero” dell’analisi malware e può essere applicato a file, dump di memoria o interi filesystem.
- **Analisi della IAT (Import Address Table):** la IAT contiene i riferimenti alle funzioni importate da librerie dinamiche. La presenza simultanea di API come **OpenProcess**, **VirtualAllocEx**, **WriteProcessMemory** e **CreateRemoteThreadEx** è un forte indicatore di tentativo di **process injection**.

L’analisi statica è estremamente utile per malware conosciuti o con caratteristiche ricorrenti, ma risulta limitata quando il codice è offuscato o impacchettato: in questi casi diventa necessario il ricorso a tecniche dinamiche.

**11.7.0.2 Analisi comportamentale** L’analisi comportamentale, detta anche **behavioral analysis**, valuta ciò che un programma *fa* piuttosto che ciò che *contiene*. Il suo obiettivo è osservare il comportamento in un ambiente controllato (sandbox, emulatore o macchina virtuale) e costruire una **firma comportamentale** basata su sequenze di azioni.

- **Behavioral signatures:** si basano sull’osservazione delle azioni del programma (creazione di processi, accesso a registro o file system, comunicazioni di rete). Sistemi come quelli di Symantec utilizzano istogrammi che contano la frequenza delle istruzioni o coppie di istruzioni per classificare il codice in base al comportamento.
- **Heuristic engines:** introducono algoritmi di apprendimento automatico (*machine learning*) per distinguere malware da software benigno. Le feature più comuni sono:
  - **sequenze di chiamate API** (indicative delle interazioni con il sistema operativo);
  - **sequenze di opcode** (n-grammi di istruzioni macchina);
  - **grafo delle funzioni chiamate** (*Call Function Graph*), che rappresenta i blocchi base del programma e i collegamenti logici fra essi, indipendentemente da mutazioni o offuscamento.

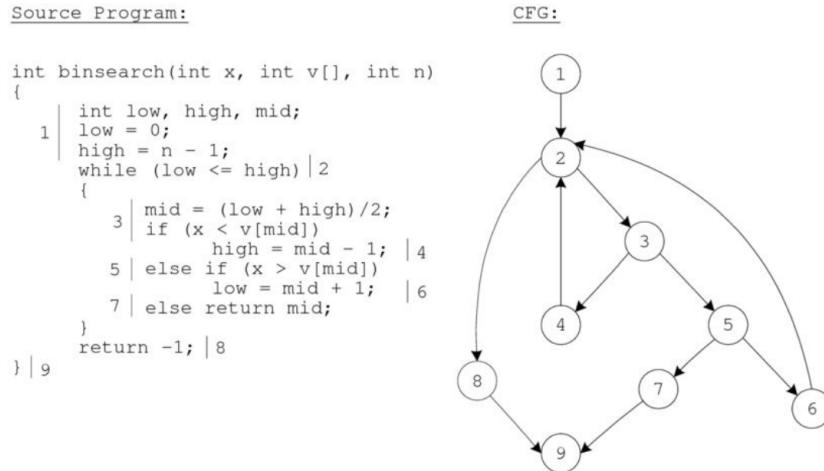


Figura 77: Esempio di Call Function Graph per estrarre blocchi di esecuzione

- **n-grams:** in questo contesto rappresentano sequenze contigue di byte o istruzioni, analoghe ai n-grammi linguistici nel testo. Servono per creare feature vettoriali che alimentano modelli di classificazione come *Naive Bayes*, *Support Vector Machine* o *Decision Tree*. Le varianti potenziate (*boosted decision tree*) forniscono spesso i risultati migliori.

**Example:** “*The cow jumps over the moon*”, with  $n=2$

- the cow
- cow jumps
- jumps over
- over the
- the moon

Figura 78: esempio di n-grams con  $n=2$

- **Byteplots:** tecnica visiva che converte i binari in immagini, confrontate con dataset di malware noti. Malware della stessa famiglia tendono a produrre pattern visivi simili, facilitando l’identificazione tramite analisi automatica.

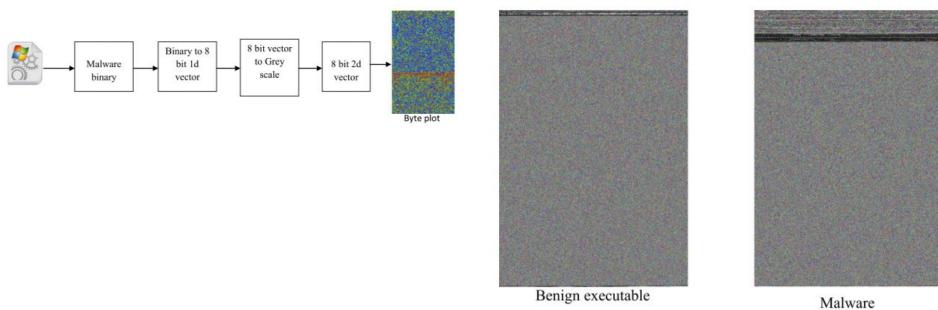


Figura 79: Esempio di byteplots

In generale, l’analisi comportamentale è più robusta contro malware polimorfici o metamorfici, ma è più costosa in termini di tempo e risorse, poiché richiede ambienti di esecuzione controllati. L’unione delle due tecniche — statica e dinamica — costituisce la base dei moderni motori antivirus, che combinano pattern matching, euristiche e modelli di apprendimento automatico per fornire una protezione completa.

## 11.8 Emulatori e Fingerprinting

Le tecniche di rilevamento basate su **firme statiche** o **euristiche** possono essere facilmente aggirate tramite offuscamento del codice o metamorfismo. Per questo motivo molti antivirus moderni includono **emulatori**, ovvero componenti che eseguono i campioni sospetti in un ambiente controllato (**sandbox**) capace di riprodurre in parte il comportamento del sistema reale.

### 11.8.0.1 Funzionamento degli emulatori

Gli emulatori simulano:

- l'**Instruction Set Architecture** (ISA) della CPU, interpretando le istruzioni del malware;
- le **API di sistema** principali, offrendo funzioni di sistema emulate (file I/O, rete, memoria, ecc.);
- un ambiente operativo virtuale in cui il programma può essere eseguito in sicurezza senza compromettere l'host.

Grazie a questa esecuzione controllata, l'antivirus può:

- osservare comportamenti sospetti che non emergono durante l'analisi statica;
- rilevare malware che si attiva solo dopo la decriptazione o il de-packing;
- superare molte tecniche di offuscamento del codice, poiché il malware viene eseguito e analizzato nel suo stato “reale”.

**11.8.0.2 Limiti e tecniche di evasione** Nonostante la loro utilità, gli emulatori non riescono a riprodurre fedelmente tutte le funzionalità del sistema operativo. Per semplicità o motivi di performance, molte API sono simulate tramite *stub*, ovvero funzioni che restituiscono valori fissi o predefiniti. Un esempio famoso è l'API `OpenMutexW` nel motore Comodo, che restituisce sempre il valore `0xB BBBB`. Poiché in un sistema reale è altamente improbabile ottenere tale valore, il malware può usarlo come indizio della presenza di un ambiente emulato.

**11.8.0.3 Fingerprinting dell'ambiente** Il **fingerprinting** consiste proprio nel testare l'ambiente circostante per capire se il programma sta girando in una sandbox o su una macchina reale. I malware più sofisticati verificano:

- valori anomali restituiti dalle API (come nel caso degli stub);
- presenza di processi, driver o nomi di dispositivi tipici delle VM (es. `VBoxService.exe`, `vmtoolsd.exe`);
- quantità di RAM o numero di core ridotti, parametri comuni nelle sandbox;
- assenza di interazione dell'utente (movimenti del mouse, eventi di input);
- tempi di esecuzione troppo rapidi (assenza di *sleep reali*).

Se il malware rileva la presenza di un ambiente virtualizzato o un emulatore, può:

- eseguire solo codice benigno per ingannare il motore antivirus;
- ritardare l'attivazione (time bomb) o attendere eventi specifici;
- disattivarsi completamente per evitare di essere analizzato.

**11.8.0.4 Contromisure adottate dagli antivirus** Per contrastare queste tecniche di evasione, gli antivirus implementano:

- emulatori più completi e realistici, capaci di restituire valori pseudo-casuali invece di costanti;
- ambienti sandbox che mascherano gli artefatti tipici delle macchine virtuali;
- **anti-anti-debugging**, per neutralizzare controlli sul debugger o sull'orologio di sistema;
- cifratura e offuscamento dei moduli interni, per impedire che il malware riconosca la firma dell'antivirus stesso.

In conclusione, gli emulatori rappresentano uno strumento essenziale per la difesa dinamica contro il malware, ma il loro successo dipende dal livello di realismo dell'ambiente simulato e dalla capacità di resistere ai test di fingerprinting condotti dai malware più avanzati.

## 11.9 Analisi del traffico cifrato

Alcuni antivirus analizzano anche traffico HTTPS, effettuando un **Man-in-the-Middle controllato**: installano un certificato CA sul sistema e generano certificati on-the-fly per ogni sito visitato. Questa tecnica può introdurre vulnerabilità:

- violazione di **HTTP Public Key Pinning (HPKP)**;
- esposizione a attacchi MITM se l'implementazione è errata;
- accettazione di **Diffie–Hellman a 8 bit**;
- indebolimento della sicurezza anche durante gli aggiornamenti AV.

È consigliabile **non abilitare l'ispezione HTTPS** sugli antivirus.

## 11.10 Fiducia e limiti del software antivirus

Gli antivirus restano strumenti essenziali ma non infallibili:

- sono scritti in linguaggi compilati e **memory-unsafe**, quindi vulnerabili a bug;
- possono essere aggirati da malware avanzato o configurazioni errate;
- inducono talvolta un **falso senso di sicurezza** negli utenti, riducendo la vigilanza.

Un computer completamente sicuro non esiste: l'antivirus deve essere visto come una **difesa a più livelli** all'interno di una strategia di sicurezza complessiva.

## 12 SYS\_07 - eBPF

### 12.1 Motivazioni: Perché osservare ciò che accade nel cluster

In un sistema moderno basato su container e orchestratori come Kubernetes, gli amministratori hanno la necessità di osservare in modo capillare cosa accade nel cluster. Le applicazioni risiedono nello spazio utente, mentre le operazioni critiche — gestione dei file, chiamate di rete, accesso alla memoria — transitano tutte attraverso il kernel. Questo rende il kernel il punto ideale per ottenere visibilità completa sul comportamento del sistema e per intercettare attività sospette, anche quando avvengono al di sotto dei servizi visibili all'utente.

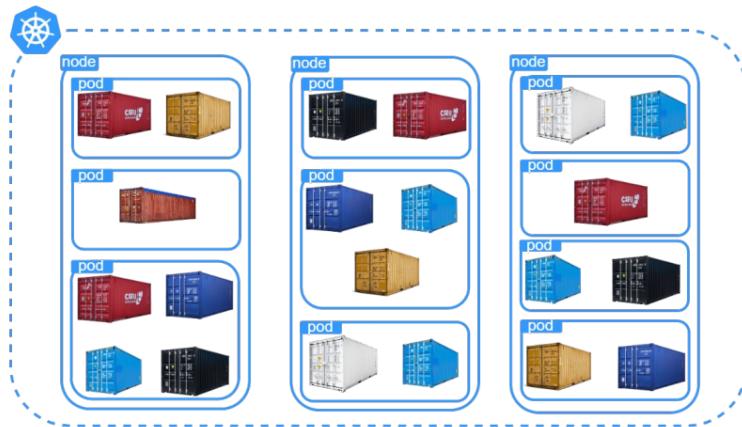


Figura 80: Struttura di un cluster Kubernetes: ogni node ospita più pod, ciascuno composto da uno o più container.

### 12.2 Il ruolo del kernel e la necessità di strumenti avanzati

Il kernel rappresenta il livello intermedio fondamentale tra le applicazioni e l'hardware fisico del sistema. Le applicazioni operano nello *user space* e non possono accedere direttamente alle risorse di basso livello come CPU, memoria, dispositivi o filesystem: ogni operazione sensibile deve necessariamente attraversare il *kernel space* tramite una *system call*.

Questo modello garantisce isolamento e sicurezza, ma implica anche che tutte le attività critiche — apertura di file, allocazioni di memoria, operazioni di rete, creazione e gestione dei processi — passino obbligatoriamente dal kernel. Di conseguenza, il kernel diventa il punto di osservazione più ricco e affidabile per comprendere il comportamento complessivo del sistema.

Tradizionalmente, ottenere visibilità a questo livello richiedeva strumenti complessi o intrusivi (modifiche al kernel, moduli caricati dinamicamente, debugger o tracer ad alto overhead). La necessità di soluzioni più sicure, flessibili ed efficienti ha portato all'evoluzione di tecnologie come eBPF, che consentono di monitorare ed estendere il kernel senza comprometterne la stabilità o le prestazioni.

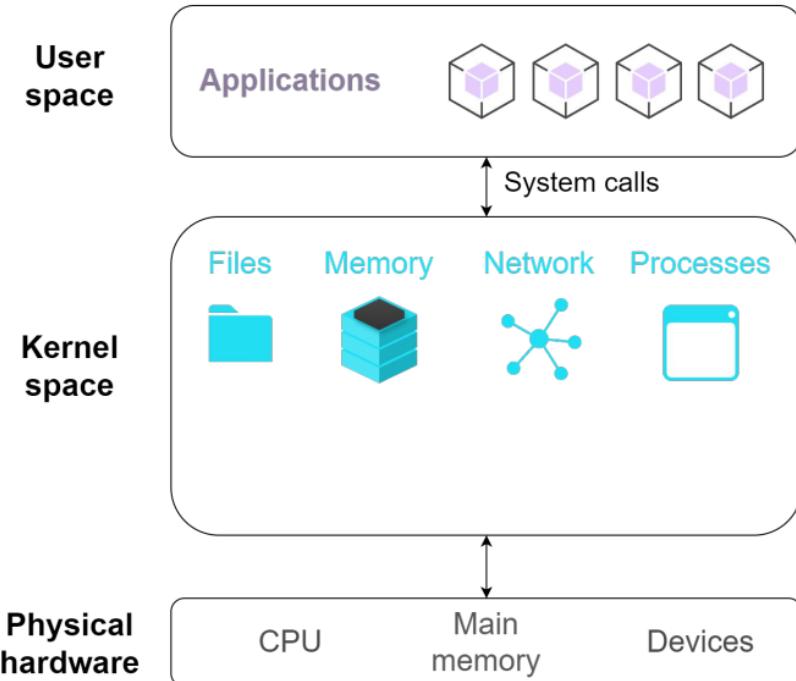


Figura 81: Relazione tra user space, kernel space e hardware: le applicazioni operano nello user space e non possono accedere direttamente alle risorse fisiche. Ogni operazione sensibile (file I/O, memoria, rete, gestione processi) passa attraverso il kernel tramite system call, che funge da intermediario unico verso CPU, memoria principale e dispositivi hardware.

### 12.3 eBPF: Hooks nel kernel e nello user space

eBPF risolve queste limitazioni consentendo di inserire “hook” in punti strategici del kernel e, optionalmente, anche in applicazioni user-space. Quando il flusso di esecuzione raggiunge uno di questi hook, il kernel invoca un programma eBPF, che può analizzare lo stato corrente, raccogliere metriche, verificare condizioni di sicurezza e reagire, il tutto con overhead trascurabile.

Gli hook sono quindi un meccanismo sicuro per estendere funzionalità del kernel o potenziare la capacità di osservazione senza modificare il codice sorgente del kernel stesso.

### 12.4 Sicurezza e osservabilità: un modello unificato

Gli eBPF program possono essere agganciati a eventi di varia natura (scheduler, syscall, eventi hardware, funzioni user-space, rete). Questo permette di:

- generare eventi personalizzati per l’osservabilità del sistema;
- aggregare metriche direttamente nel kernel senza passare per user space;
- individuare comportamenti anomali o sospetti;
- reagire immediatamente tramite segnali, blocchi o logging avanzato.

### 12.5 Tipologie di Hook eBPF

Gli hook si dividono in due categorie fondamentali: statici e dinamici, sia nel kernel che nello user space.

**12.5.0.1 Tracepoints Kernel (statici).** Sono punti di tracciamento predefiniti inseriti dagli sviluppatori del kernel mediante macro specifiche. Offrono un'API stabile attraverso le versioni del kernel e non hanno impatto prestazionale se non attivi. Sono ideali per monitoraggio continuo, analisi delle performance e debugging affidabile.

**12.5.0.2 Kprobes (dinamici).** Consentono di agganciarsi dinamicamente a quasi qualunque simbolo del kernel esportato. Sono estremamente flessibili, ma meno stabili dei tracepoint perché legati all'implementazione interna delle funzioni del kernel. Sono utili per analisi a bassa granularità e scenari di debug avanzato.

**12.5.0.3 USDT (User Statically Defined Tracing).** Sono tracepoint statici definiti dagli sviluppatori all'interno dell'applicazione. Permettono di osservare eventi logici, come fallimenti di login, errori applicativi o operazioni rilevanti per la business logic.

**12.5.0.4 Uprobes (dinamici).** Permettono di agganciare funzioni user-space a runtime, senza modificare il codice dell'applicazione. Sono l'equivalente user-space dei kprobes.

## 12.6 Tracepoints nel kernel

I tracepoint forniscono un'interfaccia stabile per analizzare eventi critici del kernel. Sono localizzati in ogni sottosistema: scheduler, filesystem, gestione memoria, system call, networking. Ogni tracepoint espone file speciali in `/sys/kernel/debug/tracing`, attraverso cui si possono:

- abilitarli o disabilitarli;
- definire filtri (ad esempio, tracciare solo specifici PID);
- leggere il formato dati degli eventi generati.

Un classico esempio è il tracepoint `sys_enter_openat`, che registra le chiamate di apertura file. Agganciandosi a questo evento con eBPF è possibile rilevare tentativi di apertura di file sensibili come `/etc/passwd` e reagire, ad esempio inviando un segnale di terminazione al processo interessato.

## 12.7 Sviluppo e caricamento di programmi eBPF con libbpf-bootstrap

Per sviluppare eBPF in C in modo moderno e coerente con le nuove API, si utilizza il framework `libbpf-bootstrap`. Questo ambiente offre uno scheletro preconfigurato che integra:

- esempi di utilizzo di tracepoint, kprobe, uprobe e USDT;
- il supporto a BTF (BPF Type Format), necessario per il CO-RE (*Compile Once, Run Everywhere*);
- strumenti di compilazione, linking e generazione skeleton automatico.

Il toolchain include inoltre `bpftool` per l'ispezione di mappe, programmi e oggetti caricati nel kernel.

## 12.8 Laboratorio: Tracepoint `openat`

In questo laboratorio si realizza una semplice forma di *runtime enforcement* utilizzando un tracepoint del kernel e un programma eBPF. L'idea è quella di agganciare un programma eBPF al tracepoint `sys_enter_openat`, che viene attivato ogni volta che un processo invoca la syscall `openat()` per aprire un file. Intercettando questo evento direttamente nel kernel, è possibile ispezionare in tempo reale quale file si sta tentando di aprire e decidere se permettere o meno l'operazione.

Il programma eBPF, caricato nel kernel tramite lo skeleton generato da `libbpf-bootstrap`, viene eseguito ad ogni ingresso nella syscall: legge in modo sicuro, con le primitive eBPF, il percorso del file passato come argomento alla `openat()`, e confronta la stringa ottenuta con il percorso sensibile `/etc/passwd`. Se il confronto ha esito positivo, il programma:

- registra l'evento nel sistema di tracing del kernel, ad esempio tramite `bpf_printk()`, in modo che possa essere osservato in `trace_pipe`;
- invia al processo chiamante un segnale di terminazione (SIGKILL) utilizzando l'helper `bpf_send_signal()`, impedendo di fatto l'accesso al file.

Dal lato *user space*, un piccolo loader scritto in C utilizza le API di `libbpf` per:

1. aprire lo skeleton del programma eBPF e verificarne il bytecode;
2. caricare il programma nel kernel;
3. attaccare automaticamente il programma eBPF al tracepoint `sys_enter_openat`;
4. rimanere in esecuzione per tutta la durata dell'esperimento.

Questo laboratorio mostra in pratica come eBPF consenta non solo di *osservare* il comportamento del sistema (monitorando tutte le aperture di file), ma anche di *far rispettare* dinamicamente una politica di sicurezza direttamente nel kernel, senza modificare il codice sorgente del kernel stesso e con un overhead molto contenuto.

Il loader user-space si occupa di aprire, caricare e agganciare lo skeleton al tracepoint, quindi rimane in attesa indefinita, mentre l'output del programma viene visualizzato tramite `trace_pipe`.

## 12.9 User Statically Defined Tracing (USDT)

Le applicazioni possono definire tracepoint personalizzati, permettendo analisi non invasive anche senza modificare la logica applicativa. Un esempio classico è l'inserimento di un probe USDT su un'applicazione di login: ogni password errata genera un evento USDT, che un programma eBPF agganciato può intercettare per registrare tentativi falliti o analizzare le stringhe inserite.

## 12.10 Kprobes e analisi dinamica del kernel

I kprobe permettono di osservare l'esecuzione interna del kernel in punti non esposti da tracepoints. Un esempio didattico è l'aggancio alla funzione `do_unlinkat` per monitorare le operazioni di cancellazione file. Monitorando questa funzione si può generare un evento ogni volta che un processo rimuove un file dal filesystem.

## 12.11 Uprobes: analisi dinamica di applicazioni user-space

Gli uprobes consentono di interceptare funzioni user-space a runtime. Il laboratorio prevede l'aggancio alla funzione `readline()` di Bash, per tracciare i comandi digitati dagli utenti. Ogni comando viene memorizzato in una mappa `BPF_MAP_TYPE_PERCPU_HASH`, evitando race condition multi-core. Questo consente di creare un contatore per ciascun comando eseguito nel sistema.

## 12.12 Progetti reali basati su eBPF

Numerosi progetti di livello enterprise utilizzano eBPF per sicurezza e osservabilità:

- **Cilium**: CNI per Kubernetes, networking e sicurezza L3-L7.
- **Hubble**: osservabilità del networking nei cluster Cilium.
- **Tetragon**: security observability e runtime enforcement.
- **Symbiote, BPFDoor, Dewdrop, Bvp47**: esempi di impiego malevolo di eBPF o filtri BPF per nascondere traffico, abilitare backdoor o attivare accessi tramite pacchetti “magici”.

## 12.13 Abuso dei BPF filter: il caso BPFDoor

BPFDoor è una backdoor avanzata che utilizza filtri BPF applicati ai socket per intercettare pacchetti contenenti “magic numbers” specifici. Grazie al fatto che i filtri BPF vengono eseguiti prima dello stack TCP/IP, la backdoor può attivarsi anche se un firewall blocca la comunicazione. Una volta ricevuto il pacchetto corretto, il malware apre una connessione inversa e fornisce una shell privilegiata all'attaccante. La complessità dei filtri aumenta con le versioni e supporta combinazioni multiple di TCP, UDP e ICMP.

## 12.14 Conclusioni

eBPF rappresenta uno strumento potentissimo che fonde osservabilità e sicurezza in un unico paradigma. Grazie all'aggancio efficiente a eventi kernel e user-space, eBPF permette di analizzare il comportamento dei sistemi in tempo reale e, allo stesso tempo, di agire proattivamente per bloccare attività malevoli. Le sue applicazioni spaziano dalla visibilità nei cluster Kubernetes, al debugging profondo, fino alla rilevazione di malware avanzati.