NoSql Injection

NoSQL injection è una vulnerabilità di sicurezza che consente a un utente malintenzionato di inserire comandi arbitrari in una db NoSQL. Questo può essere fatto inserendo input dannoso.

Molte moderne applicazioni utilizzano database NoSql per la loro capacità di immagazzinare dati di diverso tipo e per la loro facile scalabilità.
È molto più facile imbattersi in una vulnerabilità di tipo injection in un sistema che utilizza un db nosql rispetto a uno che utilizza un db sql, le nosql injection non sono molto note tra gli sviluppatori.

NoSql Injection

Esempi di possibile payload malevolo
$gt
{"$gt":""}
{"$gt":-1}
$ne
{"$ne":""}

esempi di exploit

- ***username[$ne]=1$password[$ne]=1*** *#<Not Equals>*
- ***username[$regex]=^adm$password[$ne]=1*** *#Check a <regular expression>, could be used to brute-force a parameter*
- ***username[$regex]=.{25}&pass[$ne]=1*** *#Use the <regex> to find the length of a value*

esempi di body di una chiamata post di login

- {"username": {"$ne": null}, "password": {"$ne": null} }
- {"username": {"$ne": "foo"}, "password": {"$ne": "bar"} }
- {"username": {"$gt": undefined}, "password": {"$gt": undefined}

NoSql Injection

Mitigation

Utilizzare ODM (Object Document Mapper) può dalle NoSQL injection. Gli ODM forniscono un livello di astrazione tra la logica applicativa e il database NoSQL, il che rende più difficile per gli aggressori sfruttare le vulnerabilità NoSQL injection.

- **Validazione dei dati in ingresso:** Gli ODM possono essere utilizzati per validare i dati in ingresso prima che vengano salvati nel database.
- **Filtraggio dei dati:** Gli ODM possono essere utilizzati per filtrare i dati in ingresso prima che vengano salvati nel database.
- **Sicurezza delle query:** Gli ODM possono essere utilizzati per implementare misure di sicurezza sulle query. Questo può aiutare a impedire agli aggressori di eseguire query dannose sul database.

Gli ODM però non possono fornire una protezione completa dalle NoSQL injection. Gli aggressori possono ancora tentare di sfruttare le vulnerabilità NoSQL injection utilizzando tecniche avanzate, come l'uso di payload offuscati.

Alcuni suggerimenti per proteggere le applicazioni dalle NoSQL injection:

- Utilizzare un ODM aggiornato. Gli ODM vengono costantemente aggiornati per correggere le vulnerabilità note.
- Applicare le patch di sicurezza alle librerie NoSQL. Le librerie NoSQL vengono anche costantemente aggiornate per correggere le vulnerabilità note.
- Utilizzare un firewall per filtrare il traffico in entrata. Un firewall può aiutare a bloccare gli attacchi NoSQL injection provenienti da fonti sconosciute.

NoSql Injection

Mitigation

```
@Autowired
public NoSqlInjection(MongoTemplate mongoTemplate) {
    this.mongoTemplate = mongoTemplate;
}

@GetMapping("/nosql-injection")
public NoSqlSecretInfoDocument stored(@RequestParam String secretKey) {
    Query query = new Query(Criteria.where("secretKey").is(secretKey));
    return mongoTemplate.findOne(query, NoSqlSecretInfoDocument.class);
}
```

NoSql Injection

Mitigation

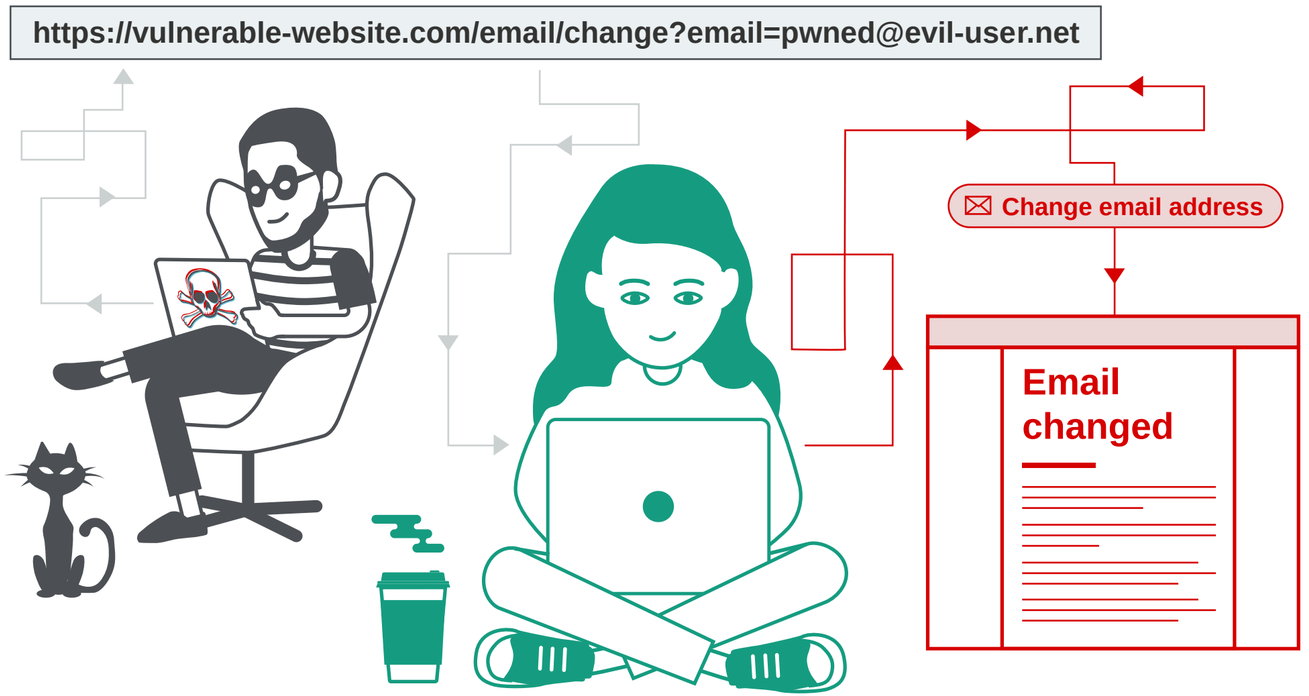
Sanitization
La sanitizzazione delle NoSQL injection è un processo che consiste nella rimozione o nella sostituzione di caratteri pericolosi dai dati in ingresso. Questo processo può essere utilizzato per impedire agli aggressori di sfruttare le vulnerabilità NoSQL injection.

Owasp mette a disposizione un cheat sheet per “combattere” le nosql injection

https://cheatsheetseries.owasp.org/cheatsheets/Java_Security_Cheat_Sheet.html#nosql

Cross-site request forgery (CSRF)

è una vulnerabilità che consente a un aggressore di indurre gli utenti a eseguire azioni che non intendono eseguire. Consente a un aggressore di aggirare in parte la same origin policy, che è progettata per impedire a siti web diversi di interferire l'uno con l'altro.



Cross-site request forgery (CSRF)

Affinché un attacco CSRF sia possibile, devono essere presenti tre condizioni fondamentali:

Un'azione rilevante. Potrebbe trattarsi di un'azione privilegiata (come la modifica dei permessi per altri utenti) o di un'azione su dati specifici dell'utente (come la modifica della sua password).

Gestione della sessione basata su cookie. L'esecuzione dell'azione comporta l’esecuzione di una o più richieste HTTP e l'applicazione si basa esclusivamente sui cookie di sessione per identificare l'utente che ha effettuato le richieste. Non ci sono altri meccanismi per tracciare le sessioni o convalidare le richieste degli utenti.

Nessun parametro di richiesta imprevedibile. Le richieste che eseguono l'azione non contengono parametri i cui valori l'attaccante non possa determinare o indovinare. Ad esempio, quando si spinge un utente a cambiare la password, la funzione non è vulnerabile se un aggressore ha bisogno di conoscere il valore della password esistente.

Cross-site request forgery (CSRF)

Ad esempio, supponiamo che un'applicazione contenga una funzione che consente all'utente di modificare l'indirizzo e-mail del proprio account. Quando un utente esegue questa azione, effettua una richiesta HTTP come la seguente:

POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

email=wiener@normal-user.com

Questo soddisfa le condizioni richieste per il CSRF:

L'azione di modificare l'indirizzo e-mail dell'account di un utente è interessante per un aggressore. In seguito a questa azione, l'aggressore sarà in genere in grado di attivare una reimpostazione della password e di assumere il pieno controllo dell'account dell'utente.

L'applicazione utilizza un cookie di sessione per identificare l'utente che ha emesso la richiesta. Non esistono altri token o meccanismi per tracciare le sessioni degli utenti.

L'aggressore può facilmente determinare i valori dei parametri della richiesta necessari per eseguire l'azione.

Cross-site request forgery (CSRF)

Con queste condizioni, l'aggressore può costruire una pagina Web contenente il seguente codice HTML:

```
<body>
  <form action="https://vulnerable-website.com/email/change" method="POST">
     <input type="hidden" name="email" value="pwned@attacker-domain.net" />
  </form>

  <script> document.forms[0].submit(); </script>
</body>
```

Quando un utente visita la pagina malevola:

- La pagina fa partire una richiesta HTTP al sito Web vulnerabile.
- Se l'utente ha effettuato l'accesso al sito Web vulnerabile, il suo browser includerà automaticamente il suo cookie di sessione nella richiesta.
- Il sito Web vulnerabile elaborerà la richiesta nel modo normale, la considererà come effettuata dall'utente vittima e modificherà il suo indirizzo e-mail.

Cross-site request forgery (CSRF)

GET usando *HTML tags*

**

Una best practice è non usare mai il metodo http GET per creare e/o variare lo stato dell'applicativo.

Come si verifica un attacco csrf
Un attaccante crea un sito web dannoso che contiene un codice che invia una richiesta HTTP autenticata a un sito web di fiducia. La richiesta HTTP autenticata chiede al sito web di fiducia di trasferire denaro da un conto bancario a un altro.

L'aggressore quindi invia un messaggio di posta elettronica a una vittima che contiene un collegamento al sito web dannoso. Quando la vittima fa clic sul collegamento, il codice dannoso viene eseguito nel browser della vittima. Il codice dannoso invia la richiesta HTTP autenticata al sito web di fiducia, che esegue la richiesta come se fosse stata inviata dalla vittima.

In genere, l'attaccante inserisce il codice HTML dannoso in un sito web che controlla e poi induce le vittime a visitare quel sito web. Ciò può avvenire inviando all'utente un link al sito web, tramite un messaggio di posta elettronica o un messaggio sui social media. Oppure, se l'attacco viene inserito in un sito web popolare (ad esempio, in un commento di un utente), potrebbe aspettare che gli utenti visitino il sito web.

Cross-site request forgery (CSRF)

Mitigation

Token CSRF - Un token CSRF è un valore unico, segreto e imprevedibile che viene generato dall'applicazione lato server e condiviso con il client. Quando si tenta di eseguire un'azione sensibile, come l'invio di un modulo, il client deve includere il token CSRF corretto nella richiesta. Questo rende molto difficile per un aggressore costruire una richiesta valida per conto della vittima.

Alcune problematiche

La convalida del token CSRF dipende dal metodo di richiesta
Alcune applicazioni convalidano correttamente il token quando la richiesta utilizza il metodo POST, ma saltano la convalida quando viene utilizzato il metodo GET.

Il token CSRF non è legato alla sessione dell'utente
Alcune applicazioni non convalidano che il token appartenga alla stessa sessione dell'utente che effettua la richiesta. Invece, l'applicazione mantiene un pool globale di token emessi e accetta qualsiasi token presente in questo pool.
In questa situazione, l'attaccante può accedere all'applicazione utilizzando il proprio account, ottenere un token valido e poi fornirlo all'utente vittima nel suo attacco CSRF.

Cross-site request forgery (CSRF)

Mitigation

Cookie SameSite - è un meccanismo di sicurezza del browser che determina quando i cookie di un sito web sono inclusi nelle richieste provenienti da altri siti web. Poiché le richieste per l'esecuzione di azioni sensibili richiedono un cookie di sessione autenticato, le restrizioni SameSite appropriate possono impedire a un aggressore di attivare queste azioni cross-site.

in pratica i cookie sameSite sono un attributo dei cookie HTTP che consente ai siti web di indicare se i cookie devono essere inviati solo con richieste dello stesso sito web o anche con richieste di altri siti web.

- **Strict:** I cookie sameSite=Strict non vengono inviati con richieste di altri siti web.
- **Lax:** I cookie sameSite=Lax vengono inviati con richieste di altri siti web, ma solo se la richiesta è un'azione di primo livello, come un clic su un collegamento. Le chiamate POST sono considerate di secondo livello, eccezioni alla regola: se una richiesta POST è una richiesta di reindirizzamento, il cookie SameSite Lax verrà inviato con la richiesta POST.
- **None:** I cookie sameSite=None vengono inviati con richieste di altri siti web, anche con richieste di terze part

Dal 2021, Chrome applica le restrizioni SameSite Lax per impostazione predefinita. Essendo questo lo standard proposto, ci aspettiamo che gli altri principali browser adottino questo comportamento in futuro.

SameSite != SameOrigin

Cross-site request forgery (CSRF)

Mitigation

Convalida basata sul referer - Alcune applicazioni utilizzano l'intestazione HTTP Referer per cercare di difendersi dagli attacchi CSRF, di solito verificando che la richiesta provenga dal dominio dell'applicazione. Questo metodo è generalmente meno efficace della convalida del token CSRF.

La convalida del Referer può essere aggirata
Alcune applicazioni validano l'intestazione Referer in un modo ingenuo che può essere aggirato. Ad esempio, se l'applicazione controlla se il dominio nel Referer inizia con il valore previsto, all'aggressore può inserirlo come sottodominio del proprio dominio:

http://vulnerable-website.com.attacker-website.com/csrf-attack

Allo stesso modo, se l'applicazione valida semplicemente che il Referer contenga il proprio nome di dominio, l'aggressore può inserire il valore richiesto in un altro punto dell'URL:

http://attacker-website.com/csrf-attack?vulnerable-website.com

Intestazione Referer
L'intestazione HTTP Referer è un'intestazione di richiesta opzionale che contiene l'URL della pagina web che ha collegato alla risorsa richiesta. In genere viene aggiunta automaticamente dai browser quando un utente attiva una richiesta HTTP. Esistono vari metodi che consentono alla pagina di collegamento di nascondere o modificare il valore dell'intestazione Referer. Ciò avviene spesso per motivi di privacy.

Cross-site request forgery (CSRF)

Mitigation

Utilizzo di un token di sessione invece di cookie di sessione.

Il token di sessione non è gestito direttamente dal browser, ma dall'applicativo.
Può essere salvato nel localStorage, l'applicativo deve espressamente aggiungere il token (a livello di codice) per ogni richiesta http che necessità di un session token.

Server-Side Request Forgery (SSRF)

È un tipo di vulnerabilità di sicurezza che consente a un utente malintenzionato di costringere un'applicazione a effettuare una richiesta HTTP a un server arbitrario, anche se il server non è accessibile direttamente dall'utente.

In un tipico attacco SSRF, l'attaccante potrebbe indurre il server a connettersi a servizi esclusivamente interni all'infrastruttura dell'organizzazione. In altri casi, potrebbe essere in grado di forzare il server a connettersi a sistemi esterni arbitrari. Questo potrebbe far trapelare dati sensibili, come le credenziali di autorizzazione.

Impatto
Un attacco SSRF riuscito può spesso portare ad azioni non autorizzate o all'accesso ai dati all'interno dell'organizzazione. Ciò può avvenire nell'applicazione vulnerabile o in altri servizi back-end con cui l'applicazione può comunicare. In alcune situazioni, la vulnerabilità SSRF può consentire a un aggressore di eseguire comandi arbitrari.

Un exploit SSRF che provoca connessioni a sistemi esterni di terze parti può dare luogo ad attacchi successivi malevoli, che possono sembrare provenire dall'organizzazione che ospita l'applicazione vulnerabile.

Server-Side Request Forgery (SSRF)

Attacchi SSRF contro il server
In un attacco SSRF contro il server, l'aggressore fa sì che l'applicazione effettui una richiesta HTTP al server che ospita l'applicazione, tramite la sua interfaccia di rete di loopback. Praticamente si tratta di inviare un URL con un host come 127.0.0.1 o localhost.

Ad esempio, un'applicazione ecommerce che consenta all'utente di vedere se un articolo è disponibile in un determinato negozio. Per fornire le informazioni sulle scorte, l'applicazione deve interrogare varie API REST di back-end. Per farlo, passa l'URL all'endpoint dell'API di back-end tramite una richiesta HTTP di front-end. Quando un utente visualizza lo stato delle scorte di un articolo, il suo browser effettua la seguente richiesta:

POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1

Server-Side Request Forgery (SSRF)

Attacchi SSRF contro il server
Un utente malintenzionato però potrebbe sostituire l'url con un altro

http://localhost/admin

di base questo servizio non è disponibile all'esterno e/o a utenti non autorizzati, ma se la request dovesse provenire da un servizio autorizzato, il servizio admin accetterebbe la richiesta.

POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://localhost/admin

Server-Side Request Forgery (SSRF)

Attacchi SSRF contro altri sistemi back-end
In alcuni casi, il server delle applicazioni è in grado di interagire con sistemi back-end non direttamente raggiungibili dagli utenti. Questi sistemi hanno spesso indirizzi IP privati non instradabili. In molti casi, i sistemi back-end interni contengono funzionalità sensibili a cui si può accedere senza autenticazione da chiunque sia in grado di interagire con i sistemi.

POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://192.168.0.68/admin

Possiamo pensare ad architetture moderve come i microservizi di una VPC e/o reti interna di kubernetes.

Server-Side Request Forgery (SSRF)

Mitigation

- ***Validazione dell'input***: La prima e più importante difesa contro gli attacchi SSRF è la validazione dell'input. È importante validare tutti gli URL forniti dagli utenti prima di effettuare una richiesta HTTP. La validazione può essere eseguita utilizzando un filtro o una regex.

- ***Limitazione dell'accesso***: È importante limitare l'accesso alle risorse interne. Ciò può essere fatto utilizzando controlli di accesso granulari o configurando una rete interna segmentata.

- ***Architettura sicura***: che non permette al client di decidere arbitrariamente quale servizio chiamare, esempio invece di lasciare la visibilità dell'url del serevizio al client, esporre esclusivamente determinati endpoints per eseguire singole azioni, il check di un ecommerce potrebbe avere un metodo GET check che si aspetta l'id del prodotto, sarà poi la business logic del servizio gateway a chiamare il servizio responsabile.

Fuzzing Enumeration

È una tecnica di ***enumeration*** che consente di trovare subdomains e endpoints. Il fuzzing consiste nell'invio di input non validi a un sistema o un'applicazione per cercare di provocare un errore o un comportamento anomalo. In questo caso, il fuzzer invia una serie di domini o percorsi di endpoint alla destinazione e analizza la risposta per trovare eventuali corrispondenze.

Spesso si utilizza una wordlist per il fuzzing di subdomains e endpoints.

- Utilizzare una wordlist di nomi di dominio comuni.
- Utilizzare una wordlist di percorsi di endpoint comuni.
- Utilizzare una wordlist di parole o frasi casuali.

Fuzzing Enumeration

Esistono diversi framework e tools per automatizzare le valutazioni della sicurezza delle applicazioni web scansionandole e trovando le vulnerabilità.

- https://github.com/ffuf/ffuf
- https://wfuzz.readthedocs.io/en/latest/
- https://github.com/OJ/gobuster

Fuzzing Enumeration

Mitigation

- Utilizzare un firewall web può aiutare a bloccare le richieste non valide. Un firewall web può essere configurato per bloccare le richieste che provengono da indirizzi IP o domini noti per essere utilizzati per attacchi di fuzzing.

- Monitorare il traffico web può aiutare a identificare eventuali anomalie che potrebbero indicare un attacco di fuzzing. Questo può essere fatto utilizzando uno strumento di monitoraggio del traffico web o un servizio di sicurezza cloud.

Rate Limiter a livello applicativo e infrastratturale

- Nginx rate limiting
- Apache mod_ratelimit
- AWS WAF rate limiting
- Cloudflare rate limiting

Librerie

- https://github.com/bucket4j/bucket4j
- https://github.com/google/guava ***Guava RateLimiter***

Brute force

Un attacco brute force contro le applicazioni web è un tipo di attacco informatico in cui un utente malintenzionato tenta di accedere a un account o a una risorsa utilizzando un numero elevato di combinazioni di nome utente e password.

In questo tipo di attacco, l'utente malintenzionato utilizza un software automatizzato per provare una serie di combinazioni di nome utente e password fino a quando non ne trova una valida. Questo può essere un processo lento e laborioso, ma può essere molto efficace se l'applicazione web non ha una buona protezione.

Gli attacchi brute force contro le applicazioni web possono essere utilizzati per accedere a una varietà di risorse, tra cui:

- Account di posta elettronica
- Account di social media
- Account bancari
- Account di gioco
- Account di servizi cloud

Brute force

Spesso gli attacchi brute force utilizzano una ***wordlist***.

Le wordlists sono liste di parole o di combinazioni di parole che vengono utilizzate negli attacchi brute force per tentare di trovare una password valida.

Le wordlists possono essere utilizzate per aumentare la probabilità che un attacco brute force riesca. Tuttavia, è importante notare che le wordlists non sono infallibili. Se la password è abbastanza complessa, può essere difficile da indovinare anche con l'uso di una wordlist.

- **Wordlist di base:** Queste wordlists contengono parole comuni, come nomi, cognomi, date di nascita e numeri di telefono.
- **Wordlist di dizionario:** Queste wordlists contengono parole che sono raccolte da dizionari e altre fonti.
- **Wordlist personalizzate:** Queste wordlists vengono create da un utente malintenzionato e possono contenere informazioni specifiche, come nomi di utenti, indirizzi e-mail o nomi di aziende.

- RockYou https://github.com/ohmybahgosh/RockYou2021.txt
- https://github.com/kkrypt0nn/wordlists/tree/main

Tool per testare l'attacco bruteforce
https://github.com/vanhauser-thc/thc-hydra

Brute force

Mitigation

- **Controllo della password strength** password alfanumeriche con caratteri speciali obbligatori
- **Abilitare l'autenticazione a due fattori (2FA)** Dare possibilità all'utente di avere un 2fa, renderlo molto esplicito a livello di ui
- **Password rotation** È importante cambiare la password periodicamente (esempio ogni 60 giorni)
- **Rate limiter** sia a livello applicativo sia a livello infrastratturale
- **Protezione captcha**

Unrestricted Upload of File

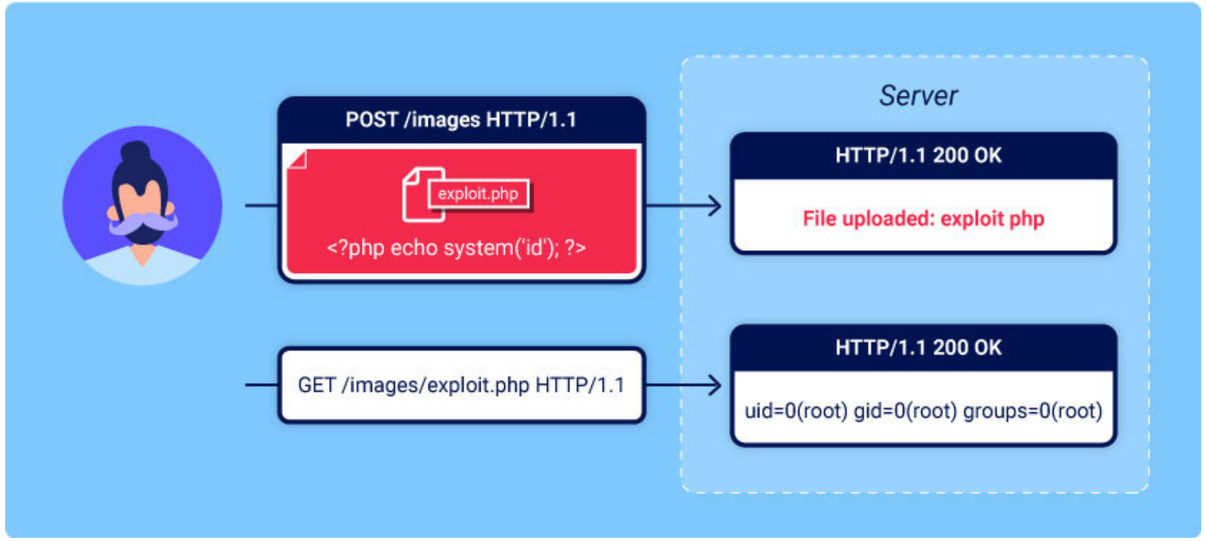
Un unrestricted upload of file è una vulnerabilità che si verifica quando un'applicazione web non controlla o non controlla correttamente il tipo di file che vengono caricati. Ciò può consentire a un utente malintenzionato di caricare un file dannoso, come un file eseguibile o uno script, che può essere utilizzato per eseguire codice arbitrario sul server o per accedere a dati sensibili.

- L'attacco può essere sfruttato caricando un file di qualsiasi tipo, ma i tipi di file più comuni che vengono utilizzati includono file eseguibili, script e file di testo.
- Il file può essere caricato utilizzando un form web, un'API o un altro meccanismo.
- Il file dannoso può essere eseguito in diversi modi, ad esempio eseguendo un comando, caricando un altro file o sfruttando una vulnerabilità nel server.

Alcuni esempi di come questa vulnerabilità potrebbe essere sfruttata

- Caricare un file eseguibile che installa un malware sul server
- Caricare uno script che ruba dati sensibili
- Caricare una reverse shell

Unrestricted Upload of File



Nel peggiore dei casi, il tipo di file non viene convalidato correttamente e la configurazione del server consente l'esecuzione di alcuni tipi di file (come .php e .jsp). In questo caso, un utente malintenzionato potrebbe caricare un file di codice lato server che funziona come una shell web, garantendo di fatto il pieno controllo del server.

Se il nome del file non viene convalidato correttamente, questo potrebbe consentire a un utente malintenzionato di sovrascrivere file critici semplicemente caricando un file con lo stesso nome. Se il server è anche vulnerabile al directory traversal, questo potrebbe significare che gli aggressori sono in grado di caricare i file in posizioni non previste.

Unrestricted Upload of File

```
@PostMapping("/upload")
public void upload(MultipartFile file) throws IOException {
    Path path = Paths.get("uploads", file.getOriginalFilename());
    file.transferTo(path);
  }
```

Unrestricted Upload of File

```
@PostMapping("/upload")
public void upload(MultipartFile file) throws IOException {
    Path path = Paths.get("uploads", file.getOriginalFilename());
    file.transferTo(path);
  }
```

Unrestricted Upload of File

Caricamento di script dannosi lato client
È possibile caricare script per attacchi lato client. Ad esempio, se è possibile caricare file HTML o immagini SVG, si potrebbero utilizzare i tag <script> per creare payload XSS.

Se il file caricato compare in una pagina visitata da altri utenti, il loro browser eseguirà lo script quando tenterà di renderizzare la pagina. Con la same-origin policy, questo tipo di attacchi funziona solo se il file viene servito dalla stessa origine in cui è stato caricato.

Offuscamento delle estensioni dei file
Anche le black list più esaustive possono essere potenzialmente aggirate utilizzando le classiche tecniche di offuscamento. Supponiamo che il codice di validazione sia sensibile alle maiuscole e non riconosca che **exploit.pHp** è in realtà un file **.php.** Se il codice che successivamente mappa l'estensione del file in un tipo MIME non è sensibile alle maiuscole e alle minuscole, questa discrepanza consente di far passare la convalida a file PHP dannosi che potrebbero essere eseguiti dal server.

Altre difese prevedono la rimozione o la sostituzione di estensioni pericolose per impedire l'esecuzione del file. Se questa trasformazione non viene applicata in modo ricorsivo, è possibile posizionare la stringa proibita in modo tale che rimuovendola rimanga comunque un'estensione di file valida. Ad esempio, si consideri cosa succede se si toglie .php dal seguente nome di file:

exploit.p.phphp

Unrestricted Upload of File

Non fidarsi del Content Type
I metadati dei file sono facilmente sovrascrivibili, quindi un check del content type lato backend non è sufficiente come filtro.

Le tecniche più sicuri cercano di verificare che il contenuto del file corrisponda effettivamente a quanto previsto.

Nel caso di una funzione di caricamento di immagini, il server potrebbe cercare di verificare alcune proprietà intrinseche di un'immagine, come le sue dimensioni. Se si prova a caricare uno script PHP, ad esempio, questo non avrà alcuna dimensione. Pertanto, il server può dedurre che non può essere un'immagine e rifiutare il caricamento di conseguenza.

Alcuni tipi di file possono sempre contenere una sequenza specifica di byte nel loro header o footer. Questi possono essere utilizzati come un fingerprint o una firma per determinare se il contenuto corrisponde al tipo previsto. Ad esempio, i file JPEG iniziano sempre con i byte FF D8 FF.

Anche questa tecnica però potrebbe essere bypassata utilizzando strumenti di stenografia per aggiungere payload malevoli ai files.

Il tool più famoso è ***exiftool***

Unrestricted Upload of File

- filtrare e validare i file (controllo del tipo e della dimensione) come primo approccio
- utilizzare librerie e/o servizi per la protezione contro file malevoli (Spring Security, Google Cloud Storage e aws s3 offrono delle protezioni)
- Ricreare il file, in caso di immagini e video una compressione potrebbe anche beneficiare le performance dell' applicativo
- Non caricare i file sulla stessa macchina dove viene eseguito il back-end o su altre macchine potenzialmente sensibili. Per evitare l'esecuzione del file lato server è meglio caricare il file su uno storage s3

https://www.virustotal.com/gui/home/upload
https://cloud.google.com/architecture/automate-malware-scanning-for-documents-uploaded-to-cloud-storage
https://www.clamav.net/