Attacco basato su richieste malformate e errori di parsing JSON in Node.js: un caso studio

Negli ultimi anni, il numero di attacchi mirati a endpoint di applicazioni web è aumentato esponenzialmente. Uno scenario comune vede gli aggressori inviare richieste malformate o corrotte per sfruttare debolezze nel parsing dei dati o per sovraccaricare il server, provocando un'interruzione del servizio. In questo articolo esploriamo un caso studio di attacco contro un'applicazione Node.js, evidenziando il ruolo chiave giocato dai log di nginx e dagli errori di parsing JSON nel backend.

Nel nostro caso, l'applicazione Node.js implementa un endpoint `/api/contact`, accessibile tramite richieste **POST**. Questo endpoint è progettato per ricevere dati, molto probabilmente in formato **JSON**, e processare le informazioni di contatto inviate dall'utente. Tuttavia, nel file di log di **nginx**, abbiamo rilevato numerosi tentativi di accesso a questo endpoint da parte di un singolo IP, con richieste ripetute che spesso hanno ricevuto un codice di risposta **400 Bad Request** o **403 Forbidden**.

Ecco un estratto del log di nginx che evidenzia il comportamento dell'attacco:

```
IP - - [21/Oct/2024:13:22:43 +0200] "POST /api/contact HTTP/1.1" 200 11
"https://gabrieleromanato.com/contatti/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.0 Safari/537.36"
IP - - [21/Oct/2024:13:22:50 +0200] "GET /contact HTTP/1.1" 404 1237 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.4844.0 Safari/537.36"
IP - - [21/Oct/2024:13:23:17 +0200] "POST /api/contact HTTP/1.1" 400 138
"https://gabrieleromanato.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.0 Safari/537.36"
IP - - [21/Oct/2024:13:24:16 +0200] "POST /api/contact HTTP/1.1" 200 85
"https://gabrieleromanato.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.0 Safari/537.36"
```

Le richieste provenienti dall'indirizzo IP sopra menzionato mostrano un comportamento anomalo: sono numerose e si verificano in rapida successione. Notiamo inoltre che molte di esse ricevono un codice di errore 400. Questo suggerisce che il server stia ricevendo dati malformati, probabilmente durante il processo di parsing del corpo della richiesta.

Contemporaneamente, nel log di **journalctl** dell'applicazione Node.js, sono stati osservati diversi errori di parsing JSON, che indicano problemi nella lettura dei dati ricevuti da queste richieste. Gli errori di parsing si verificano quando il server tenta di interpretare i dati ricevuti come JSON ma questi sono corrotti o malformati. In scenari come questo, le richieste inviate al server non seguono il formato previsto, causando un errore:

```
SyntaxError: Unexpected token in JSON at position 0
    at JSON.parse (<anonymous>)
    ...
```

Questo errore è strettamente correlato alle risposte 400 Bad Request osservate nei log di nginx: il server rifiuta di processare i dati malformati. Questo tipo di attacco è spesso una tecnica utilizzata per sovraccaricare il server o sfruttare debolezze nel sistema di validazione e parsing.

Il comportamento osservato suggerisce che l'attaccante stia tentando di eseguire un attacco **flooding** o **brute force** contro l'endpoint `/api/contact`. L'obiettivo potrebbe essere:

- **Flooding**: Inviare un'enorme quantità di richieste malformate nel tentativo di sovraccaricare il server e causare un'interruzione del servizio (Denial of Service).
- **Brute Force**: Tentare di sfruttare una vulnerabilità nell'endpoint o nel processo di parsing del JSON per aggirare la sicurezza del sistema.

Nel nostro caso, l'attacco non ha avuto successo, poiché le risposte 400 Bad Request e 403 Forbidden indicano che il server sta respingendo correttamente le richieste malformate o non autorizzate. Tuttavia, è importante considerare che un attacco di questo tipo potrebbe causare un sovraccarico del server se il numero di richieste dovesse aumentare ulteriormente.

Per prevenire questo tipo di attacchi, ci sono diverse contromisure che possono essere adottate:

1. **Validazione rigorosa dei dati**: È importante implementare una robusta validazione del corpo della richiesta nel backend Node.js. L'uso di middleware come `express-validator` o `Joi` può aiutare a garantire che i dati siano nel formato corretto prima che il server tenti di parsarli come JSON.

2. **Gestione degli errori di parsing**: Implementare un meccanismo di gestione degli errori di parsing in Node.js che risponda immediatamente con un messaggio di errore chiaro e un codice di stato appropriato (es. 400 Bad Request), riducendo così l'impatto sul server.

3. **Rate Limiting**: L'implementazione di un sistema di rate limiting può prevenire che un singolo IP invii una quantità eccessiva di richieste in un breve periodo. Pacchetti come `express-rate-limit` possono essere utili per limitare il numero di richieste accettate da un singolo IP in un intervallo di tempo definito.

4. **Firewall applicativo (WAF)**: L'uso di un firewall applicativo può rilevare e bloccare automaticamente le richieste sospette, come quelle che contengono JSON malformati o che mostrano un comportamento anomalo.

Conclusione

Questo caso studio evidenzia l'importanza di monitorare i log sia a livello di web server (nginx) che di backend (Node.js) per rilevare attacchi potenziali come quelli basati su richieste malformate e errori di parsing JSON. Sebbene in questo caso il sistema abbia correttamente respinto le richieste

malevole, è cruciale adottare misure preventive per proteggere ulteriormente l'applicazione da attacchi futuri.

Implementando meccanismi di validazione, rate limiting e gestione degli errori, è possibile rafforzare la sicurezza dell'applicazione e ridurre il rischio di interruzioni causate da attacchi flooding o brute force.

Applicazioni Correlate

- 

  ### Node.js Placeholder Image

  Applicazione per la generazione con Node.js di immagini segnaposto.
  DockerDocker ComposeNode.jsJavaScriptExpressJS

- 

Node.js URL Shortener

Implementazione in Node.js di un sistema per l'abbreviazione degli URL.
DockerDocker ComposeNode.jsJavaScriptExpressJSMongoDB

- ### JavaScript App Hash Change

  Applicazione che sfrutta gli hash degli URL per gestire contenuto dinamico in JavaScript. DockerDocker ComposeNode.jsJavaScriptExpressJSMySQL