Università degli Studi di Salerno

Penetration Testing Report

CTF "Opensource" su hackthebox.com

Ciro Maione mat. 0522500977 | Corso di PTEH | A.A. 2021/2022

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Indice

1 Executive Summary

Nell'ambito del corso di *Penetration Testing And Ethical Hacking*, tenuto dal professor Arcangelo Castiglione presso l'Università degli Studi di Salerno, è stata svolta come attività progettuale un penetration testing sulla macchina *OpenSource* messa a disposizione dalla piattaforma *HackTheBox* (1).

L'approccio utilizzato è di tip *Black Box*, in quanto la piattaforma fornisce esclusivamente un accesso alla rete in cui è presente la macchina ed il suo indirizzo.

Il test ha messo in luce numerose vulnerabilità *critiche* che possono consentire ad un utente malintenzionato di provocare danni al server in sé e di minare la credibilità del servizio offerto. In primo luogo vi è un **approccio sbagliato nelle modalità di condivisione del software** come progetto *open source*, in quanto sono stati condivisi esternamente anche file che espongono informazioni riguardo alla configurazione del server oltre a contenere informazioni private dell'utente coinvolto nello sviluppo.

In secondo luogo **il software in sé presenta alcune vulnerabilità importanti** che se sfruttate possono permettere di interrompere il servizio e di ottenere pieno controllo dell'applicazione web.

Inoltre è stato rilevato un **meccanismo di backup problematico**, messo a punto dall'utente della macchina, che unito alle debolezze citate in precedenza permette di ottenere il pieno controllo del'asset con il massimo dei privilegi.

La macchina presenta un livello di rischio **molto alto**. Si suggerisce di intervenire al più presto nella mitigazione delle problematiche riscontrate, a tale scopo in questo report verranno elencati varie tecniche e suggerimenti per mettere in sicurezza l'asset.

2 Vulnerability Report

La macchina presenta numerose vulnerabilità che ne vanno a compromettere la sicurezza, in questa sezione verranno elencate tutte per ordine di gravità.

- **Vulnerabilità nella pagina di upload del file.** **(Rischio Critico)**
  La pagina di upload del file del servizio **Upcloud** permette, attraverso una vulnerabilità nel software, di modificare in modo arbitrario tutti i file presenti nel container Docker che ospita l'applicazione, compresi i file dell'applicazione stessa. Quindi sfruttando questa vulnerabilità è possibile effettuare varie operazioni dannose, che vanno dal manomettere l'applicazione fino ad ottenere il pieno controllo del container.

- **Approccio errato nella distribuzione del codice sorgente.** **(Rischio Alto)**
  Il modo in cui viene condiviso il codice sorgente dell'applicazione presenta diverse falle:

  – dall'archivio è possibile ricavare delle credenziali dell'utente che non dovrebbero essere accessibili pubblicamente;

  – l'archivio contiene, oltre al sorgente, anche diversi file relativi alla messa in produzione dell'applicazione, che rilevano numerose informazioni relative all'asset e che non hanno motivo di essere distribuite insieme al codice.

- **Isolamento inefficace tramite container Docker.** **(Rischio Alto)**
  Il container Docker utilizzato per la messa in produzione dell'applicazione non è completamente isolato dal sistema originale. Infatti, dal container è possibile accedere a servizi sulla macchina host che non dovrebbero essere acceduti al di fuori di questa.

- **Sistema di backup insicuro.** **(Rischio Alto)**
  Il meccanismo messo a punto dall'utente per effettuare il backup periodico della propria cartella *home* presenta numerose debolezze:

  – viene effettuato da un programma che ha permessi ingiustamente elevati;

  – non viene utilizzato nessun meccanismo di cifratura, quindi i dati sensibili presenti nella cartella vengono salvati in chiaro.

- **Libreria jQuery non aggiornata.** **(Rischio Medio)**
  La versione della libreria *jQuery* utilizzata nell'applicazione risulta essere vulnerabile ad alcuni attacchi.

- **Errata configurazione del server Werkzeug.** **(Rischio Medio)**
  Il server espone una console di debug protetta da un pin, che dovrebbe essere utilizzata solo durante lo sviluppo e non in produzione. Ottenendo abbastanza informazioni sulla macchina è possibile bypassare il pin ed eseguire comandi python arbitrari.

3 Remediation Report

L'attività di penetration testing sull'asset ha mostrato un livello di sicurezza molto basso, a tal proposito di seguito vengono suggerite alcune contromisure che possono essere applicate per rendere più sicura l'infrastruttura.

- Patch dell'applicazione web che includa una modifica nella funzione che provvede al salvataggio del file. In particolare si consiglia di effettuare una sanificazione più efficace del nome del file, oppure, ancora meglio, evitare di utilizzare il nome del file estratto dalla richiesta per salvare l'elemento, e utilizzare piuttosto un nome nuovo.
- Evitare di condividere con il codice sorgente dettagli sulla messa in produzione e sulla struttura interna del server.
- L'utente `dev01` dovrebbe cambiare le proprie credenziali compromesse (password e chiave ssh).
- Il container Docker dovrebbe essere configurato in modo che non gli sia possibile effettuare richieste ai servizi interni della macchina host.
- Il programma che effettua il backup dovrebbe essere eseguito con privilegi più bassi (non di amministratore). Inoltre sarebbe consigliabile che i dati del backup venissero crittografati per evitare il salvataggio in chiaro di informazioni sensibili come le chiavi ssh. In alternativa si consiglia di utilizzare uno dei tanti servizi di backup disponibili anche gratuitamente.
- Aggiornare la libreria `jQuery` ad una versione uguale o superiore alla `3.5.0`.
- Configurare il server Werkzeug in modo che non sia attiva la console di debug.

4 Findings Summary

In questa sezione viene mostrata una sintesi delle vulnerabilità rilevate durante l'attività di penetration testing, in termini della loro gravità.

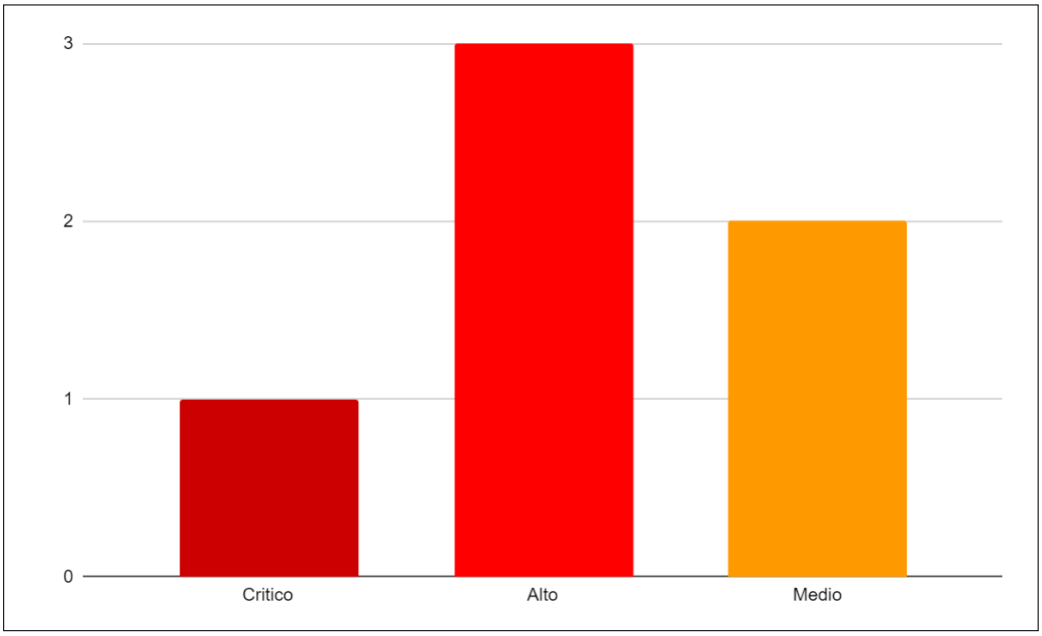

Figura 1: Istogramma

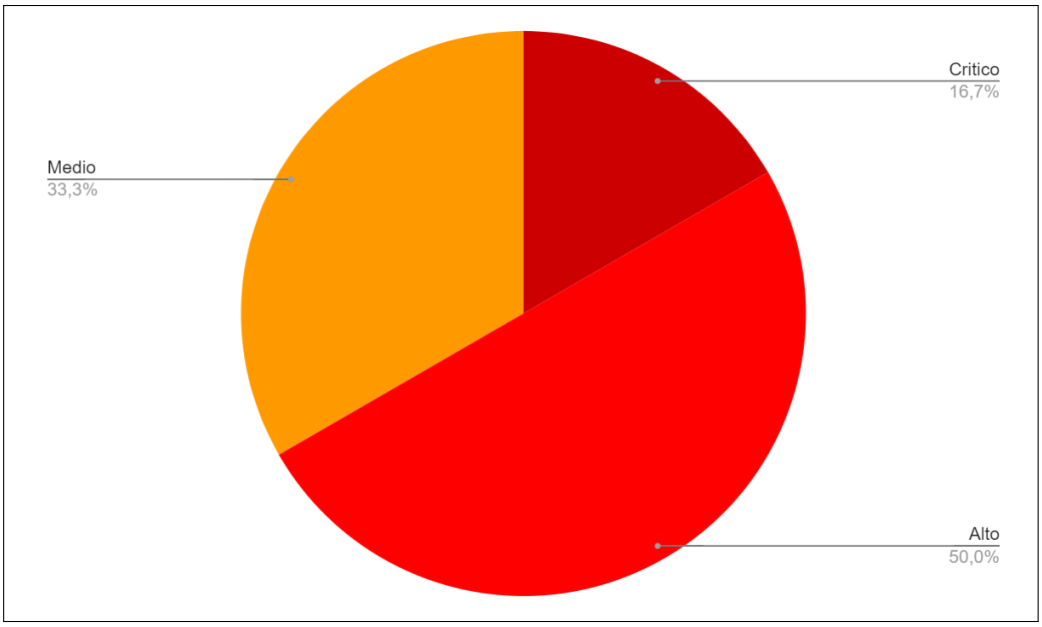

Figura 2: Diagramma a torta

5 Detailed Summary

In questa sezione verrano descritte in modo dettagliato le vulnerabilità trovate, enfatizzando in modo particolare il modo in cui la debolezza è stata rilevata, i rischi associati e le mitigazioni possibili per la messa in sicurezza.

5.1 LFI

Local File Inclusion (LFI) tramite form di upload

Rischio Critico

Descrizione: Una vulnerabilità di tipo Local File Inclusion si verifica quando un'applicazione crea un percorso attraverso una variabile controllata dall'attaccante, in modo tale che quest'ultimo possa sovvertire il modo in cui l'applicazione carica il codice per l'esecuzione.

Spiegazione: La vulnerabilità è stata trovata analizzando il codice sorgente distribuito in modo open-source dalla piattaforma, ed è stata verificata attraverso l'invio di richieste POST tramite il web proxy *Burpsuite*.

In particolare modificando il valore dell'attributo *filename* del *multipart/form-data* è possibile specificare un path assoluto che non viene controllato dall'applicazione e che quindi consente di sovrascrivere qualsiasi file sul server con quello che carichiamo tramite form.

La vulnerabilità si trova nel file *views.py*, **linea 13**:

```
file_name = get_file_name(f.filename)
```

La funzione *get_file_name* (scritta dallo sviluppatore) non effettua una sanificazione efficace del filename, in quanto effettua solo la rimozione delle sotto-stringhe *"../"*, ma è consentita l'iniezione di path assoluti.

Rischi: Poiché, come è stato detto, lo sfruttamento di questa vulnerabilità consente di sovrascrivere idealmente qualunque file presente sul file system del server, è possibile dunque sovrascrivere i *"template"* con estensione html che vengono renderizzati dall'applicazione all'occorrenza e che possono contenere codice Python, che viene eseguito per costruire la pagina di risposta.

Sfruttando questo tipo di exploit è possibile ottenere una Remote Code Execution (RCE) che permette di ottenere il controllo remoto del container (Maggiori informazioni sul payload utilizzato sono nell'appendice A).

Mitigazioni: Il modo migliore per mitigare questa vulnerabilità consiste nell'evitare di utilizzare il campo filename preso dalla richiesta POST, e piuttosto generare un nome univoco per ogni file caricato, estrapolando dalla richiesta solamente l'estensione. In alternativa è possibile correggere la funzione *get_file_name* in modo che rimuova tutti i pattern che possano avere influenza nel path.

5.2 Information Leakege

Information Leakege nella distribuzione del codice sorgente

Rischio Alto

Descrizione: L'information leakege si verifica quando informazioni sensibili relative all'asset vengono inconsapevolmente o meno esposte ad attori esterni.

Spiegazione: La vulnerabilità è stata riscontrata analizzando il sorgente fornito dalla web application, in particolare sono presenti:

- la cartella *.git* che consente di avere accesso allo storico del controllo versione, spostandosi su uno dei commit è possibile trovare delle credenziali relative all'utente *dev01*, all'interno del file *settings.json* (linea 3):

```
"http.proxy": "http://dev01:Soulless_Developer#2022@10
   .10.10.128:5187/",
```

- vari file relativi al deploy dell'applicazione: *Dockerfile,build-docker.sh*. Tali file espongono informazioni relative alla struttura del file system del server, le porte esposte, la working directory del server.

Rischi: Un attaccante può sfruttare le informazioni sul deploy per condurre un attacco. Il pericolo maggiore in questo caso risiede nelle credenziali che hanno permesso di accedere al backup dell'utente dev01 e di ottenere la sua chiave ssh (come mostrato in seguito).

Mitigazioni: Per correggere questa vulnerabilità si consiglia condividere solo il codice, evitando i dettagli del deploy. Inoltre si consiglia all'utente dev01 di cambiare la sua password e la sua chiave ssh, in quanto quelle attuali potrebbero essere compromesse irrimediabilmente. Infine si raccomanda agli sviluppatori di prestare attenzione riguardo i file che si decide di pubblicare sul repository, evitando di inserire credenziali o informazioni sensibili in modo hard-coded e usare piuttosto file di configurazione appositamente estromessi dal controllo versione tramite il *.gitignore*.

5.3 Docker mal configurato

Isolamento non efficace del container Docker
Rischio Alto
Descrizione: L'isolamento del container dal sistema host non è corretto.
Spiegazione: La vulnerabilità è stata scoperta provando a interrogare tramite il container un servizio che è presente sull'host ma che era filtrato esternamente. Questo ha reso possibile attraverso port forwarding di accedere al servizio in questione dall'esterno.
Rischi: I rischi principali sono legati al fatto che avendo accesso al container è possibile avere accesso ai servizi dell'host filtrati all'esterno, nel caso specifico è stato possibile accedere al servizio *Gitea* (2). Maggiori dettagli nell'appendice B
Mitigazioni: Per correggere la vulnerabilità è necessario effettuare un filtraggio appropriato che tratti le richieste che arrivano dal container allo stesso modo di quelle che vengono dall'esterno.

5.4 Sistema di backup

Esecuzione con privilegi elevati
Rischio Alto
Descrizione: La vulnerabilità si presenta quando dei programmi vengono eseguiti con privilegi ingiustamente elevati, consentendo a un utente base di effettuare una privilege escalation.
Spiegazione: La vulnerabilità è stata individuata osservando i processi in esecuzione tramite lo strumento *pspy* (7), è stato scoperto che a intervalli regolari viene lanciato lo script *git-sync*, presente nella cartella */usr/local/bin/*. Lo script è di proprietà di *root* ed è eseguito da *root*. Un utente con i permessi di *dev01* può sfruttare questo meccanismo per eseguire codice arbitrario come *root* utilizzando i git hooks (8). Maggiori dettagli sono nell'appendice C.
Rischi: Poichè un utente che ha i permessi di scrittura nella cartella */home/dev01/.git/hooks* riesce ad eseguire comandi come root, questi può effettuare qualsiasi operazione sul sistema.
Mitigazioni: Si consiglia di eseguire lo script come utente *dev01* o di cambiare sistema di backup.

Backup non cifrato
Rischio Medio
Spiegazione: Una volta ottenuto l'accesso al repository su Gitea, contenente il backup dell'utente *dev01*, è stato subito rilevato che i dati erano salvati in chiaro, ed era quindi possibile leggere la chiave ssh dell'utente.
Rischi: Un utente che ha in qualche modo avuto accesso all'account di *dev01* su Gitea riesce a leggere la chiave ssh dell'utente e quindi ad accedere al server con i suoi privilegi.
Mitigazioni: è consigliabile utilizzare un qualche meccanismo di cifratura dei dati in modo che qualora qualcuno avesse accesso al repository non sarebbe in grado di leggere i dati, in alternativa è possibile cambiare meccanismo di backup, affidandosi per esempio a programmi specializzati e consolidati nel settore.

5.5 Versione vulnerabile della libreria jQuery

Cross Site Scripting
Rischio Medio
Descrizione: Nelle versioni di jQuery maggiori o uguali a 1.2 e precedenti alla 3.5.0, il passaggio di HTML da fonti non attendibili, anche dopo averlo sanificato, a uno dei metodi di manipolazione DOM di jQuery (es. .html(), .append() ecc.) può portare all'esecuzione di codice arbitrario.
Spiegazione: La vulnerabilità è stata individuata tramite lo strumento *Nessus*, in particolare è stato effettuato un basic network scan.
CVE collegati: CVE-2020-11022 (3), CVE-2020-11023 (4).
Rischi: Al momento non è stato rilevato codice vulnerabile, ma dato il fatto che l'applicazione è ancora in sviluppo, è possibile che in futuro possa essere introdotta qualche debolezza. Eventuali vulnerabilità potrebbero portare all'esecuzione di codice arbitrario sul browser di una vittima sfruttando l'applicazione in questione.
Mitigazioni: Si consiglia di usare una versione di jQuery superiore o uguale alla 3.5.0.

5.6 Server mal configurato

Il server espone una debug console
Rischio Medio
Descrizione: La debug console è esposta nell'applicazione in produzione, essa dovrebbe essere utilizzata solo durante lo sviluppo.
Spiegazione: La vulnerabilità è stata rilevata tramite lo strumento *nikto*, che ha comunicato la presenza della route */console*.
Rischi: Con la console attiva, un attaccante potrebbe riuscire a bypassare il pin e ad ottenere una Remote Command Execution, sfruttando Python, come spiegato qui (5).
Mitigazioni: Come suggerito dalla documentazione (6), la console va disattivata in ambiente di produzione e utilizzata solo in ambiente di sviluppo.

Riferimenti

1. https://app.hackthebox.com
2. https://gitea.io/en-us/
3. https://nvd.nist.gov/vuln/detail/CVE-2020-11022
4. https://nvd.nist.gov/vuln/detail/CVE-2020-11023
5. https://github.com/wdahlenburg/werkzeug-debug-console-bypass
6. https://werkzeug.palletsprojects.com/en/2.0.x/debug/#debugger-pin
7. https://github.com/DominicBreuker/pspy
8. https://git-scm.com/book/it/v2/Customizing-Git-Git-Hooks

Appendice

A LFI

Per effettuare la Local File Inclusion e l'esecuzione di comandi arbitrari sul target è stato utilizzato il web proxy *Burpsuite*, per modificare la richiesta POST relativa all'upload. In particolare è stato modificato l'attributo filename del multipart-form data inviato, per fare in modo di sovrascrivere il template `success.html`, ed è stato inserito come contenuto del file caricato il seguente codice:

```
{{ request.application.__globals__.__builtins__.__import__('os').popen('id
    ').read() }}
```

Il codice in questione permette di eseguire un comando arbitrario passato come argomento a `popen` e stampa l'output nella pagina. È possibile sostituire a `id` un qualsiasi comando di shell.
In figura 4 è mostrato l'invio di un payload che consente di ottenere una reverse shell.

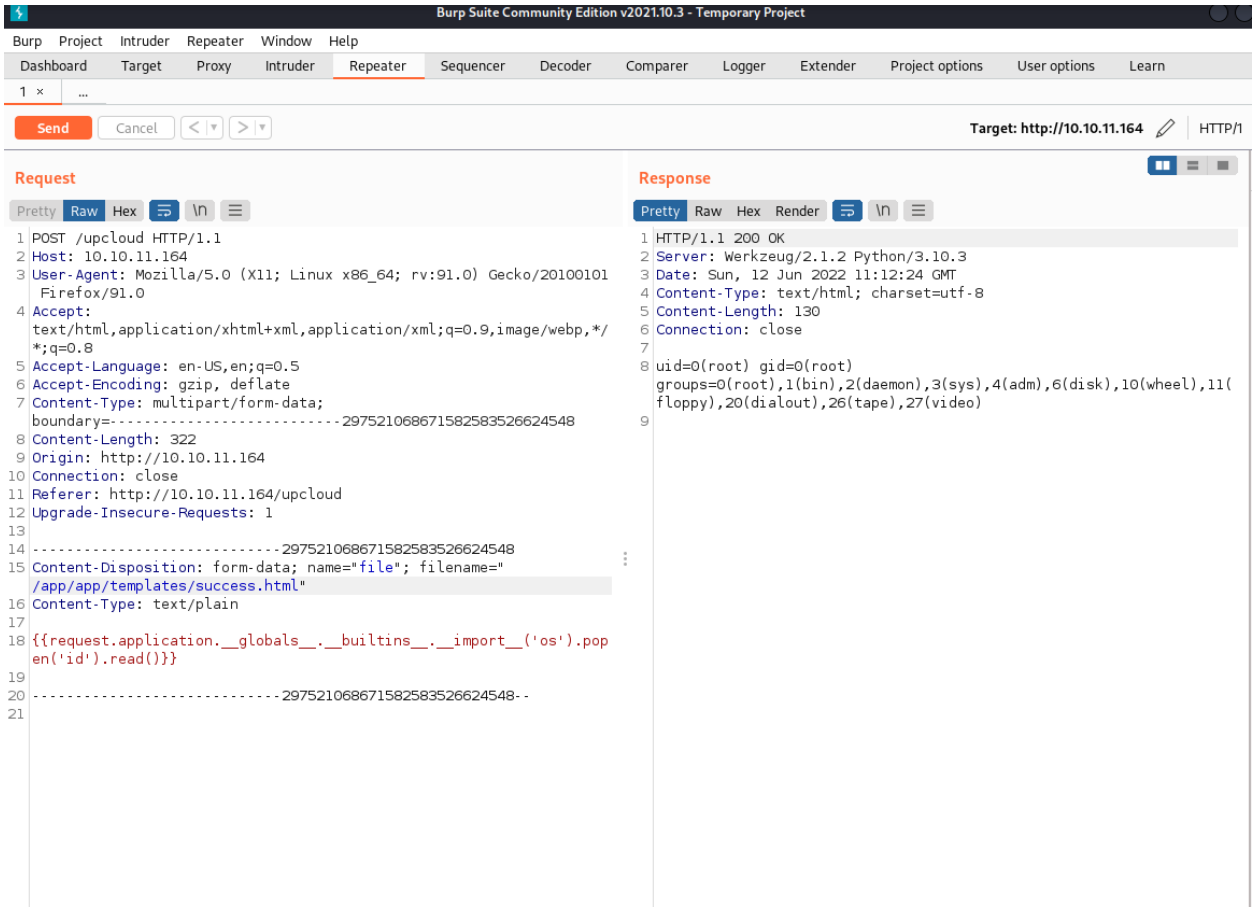

Figura 3: Schermata di Burpsuite con esecuzione del comando id tramite LFI

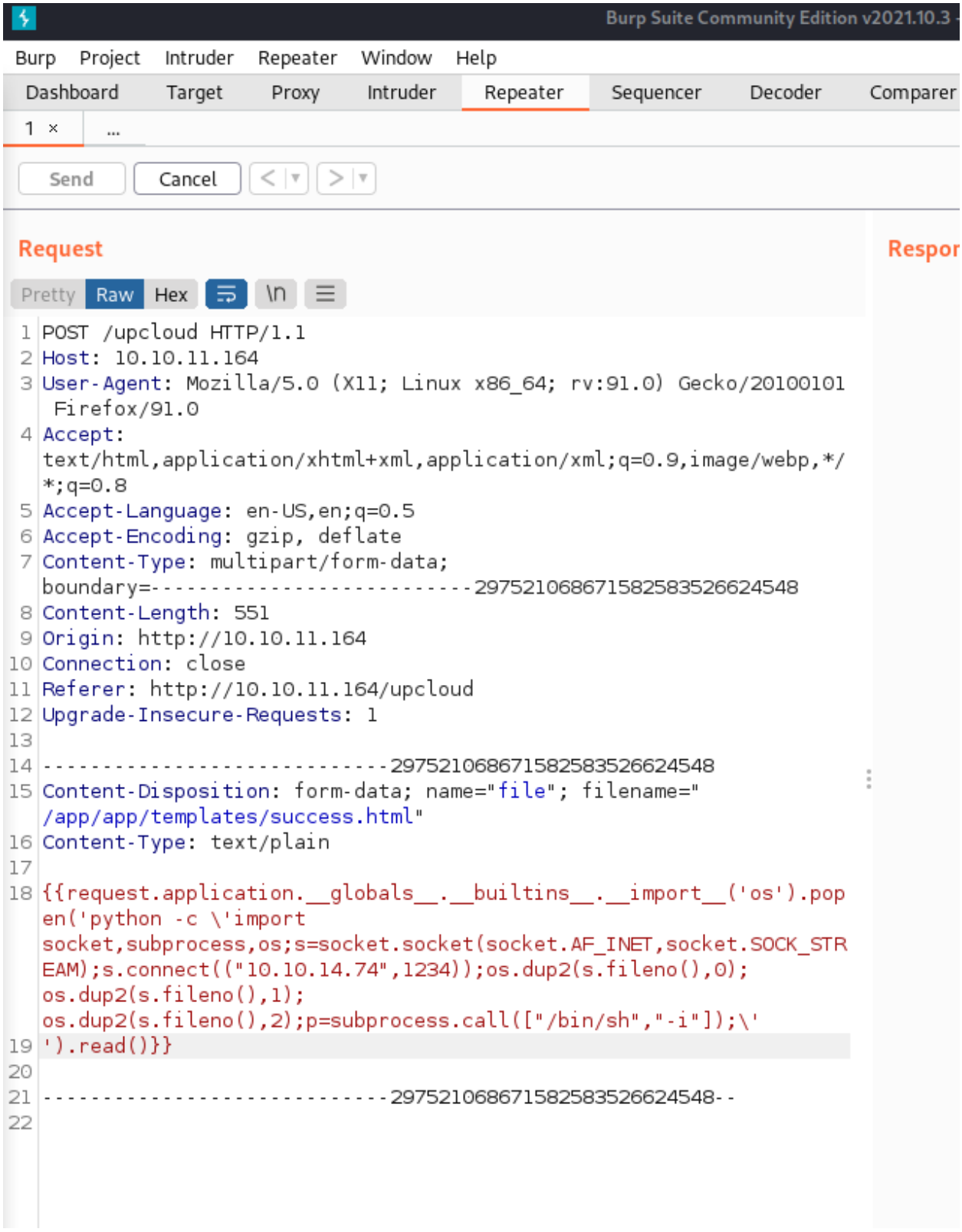

Figura 4: Schermata di Burpsuite con esecuzione della reverse shell

B Port Forwarding

Dal container Docker è possibile effettuare richieste al servizio *Gitea* sulla porta `3000` (filtrata dal server), figura 5.

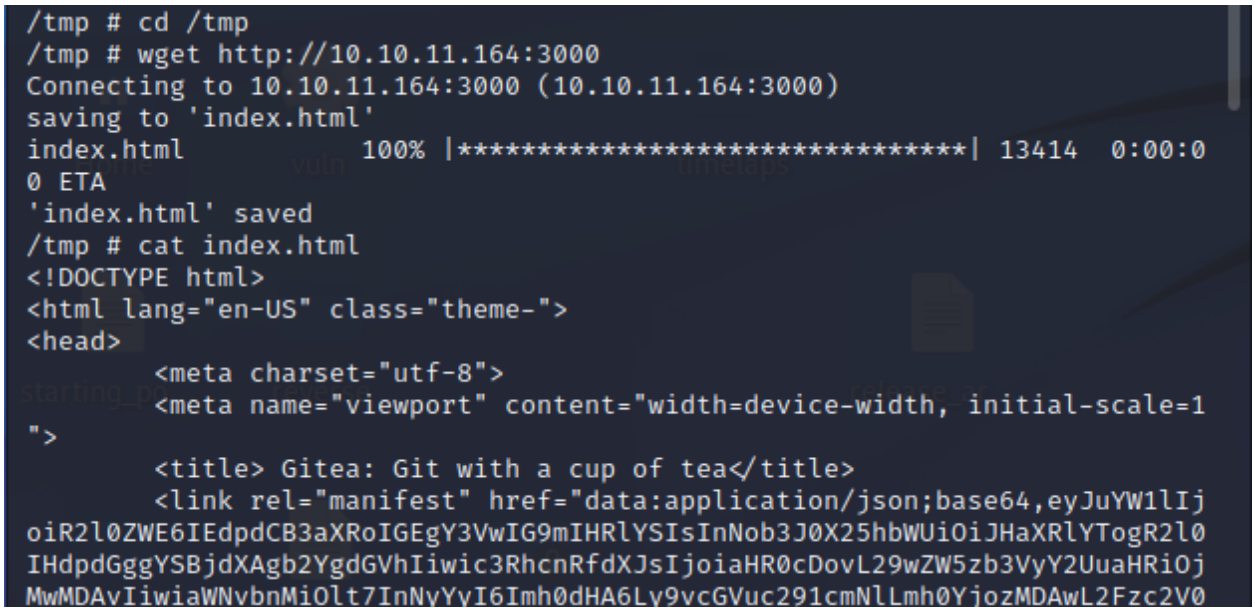

Figura 5: Esecuzione di wget sul servizio filtrato

Per poter accedere al servizio dal browser della macchina attaccante è necessario effettuare un port forwarding, per fare questo è stato usato lo strumento `socat`, che è stato installato manualmente sul container. È stato stoppato il servizio in ascolto sulla porta `80` sul conainer ed è stato poi lanciato il comando socat nel seguente modo:

```
socat tcp -listen:80,reuseaddr,fork tcp:10.10.11.164:3000
```

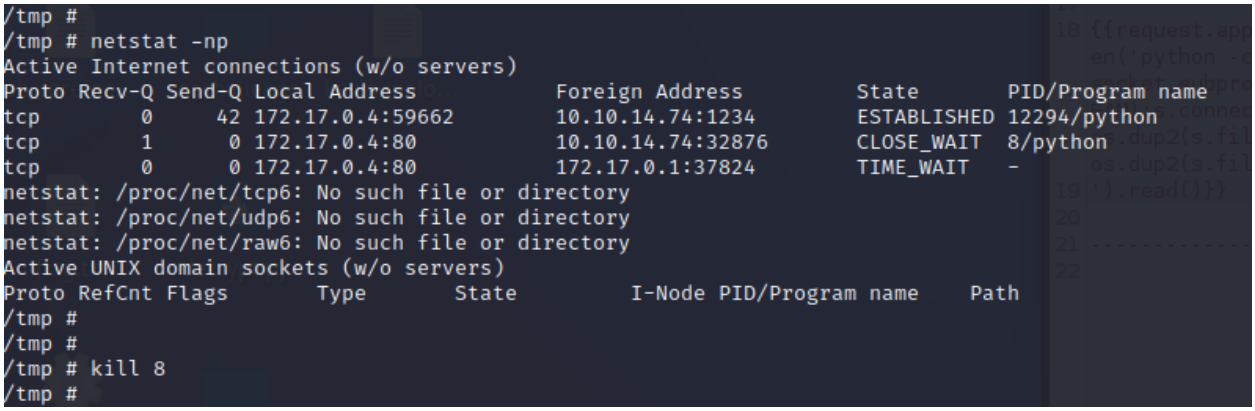

Figura 6: Il processo in ascolto sulla porta 80 viene fermato

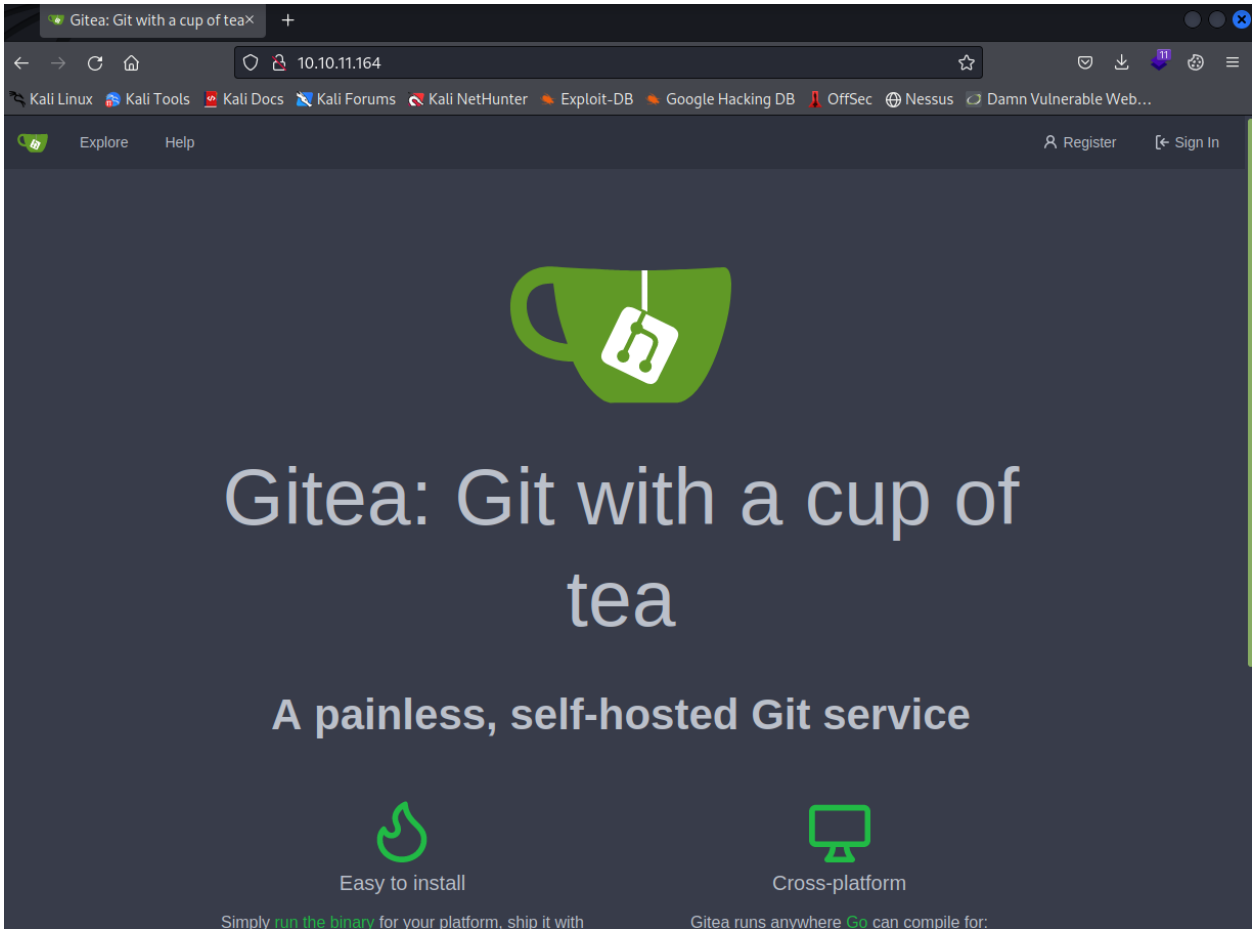

Figura 7: Il servizio Gitea è ora accessibile esternamente

C Script di backup

In figura 8 è mostrato lo screen dell'output dello strumento `pspy` che ha permesso di rilevare l'esecuzione intermittente dello script `git-sync`.

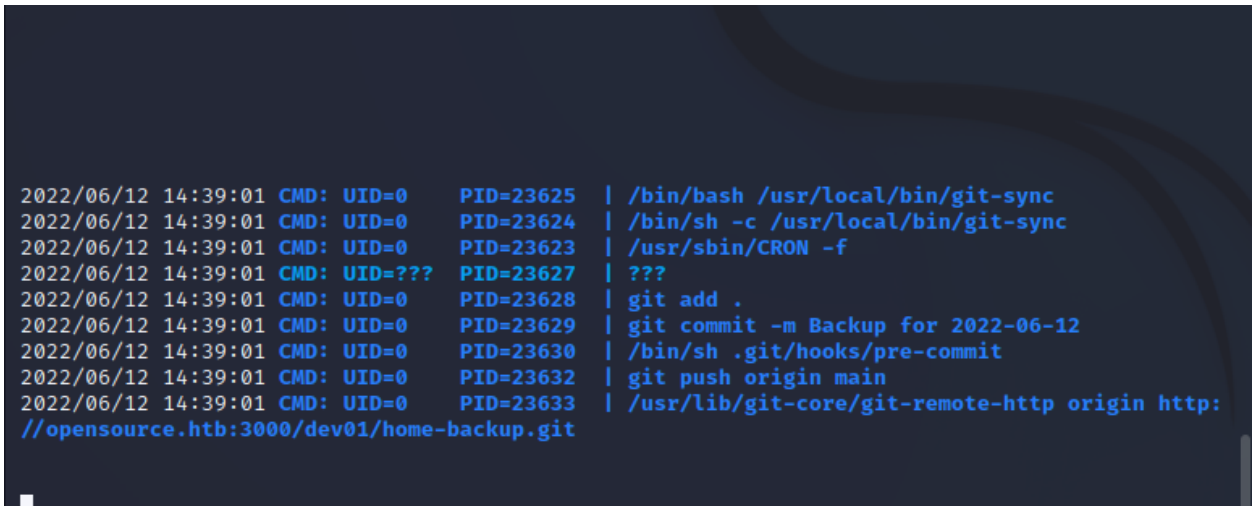

Figura 8: Output di pspy

Per ottenere l'elevazione dei privilegi è stato creato un file chiamato `pre-commit` all'interno della cartella `/home/dev01/.git/hooks/`, il cui contenuto è mostrato in figura 9.

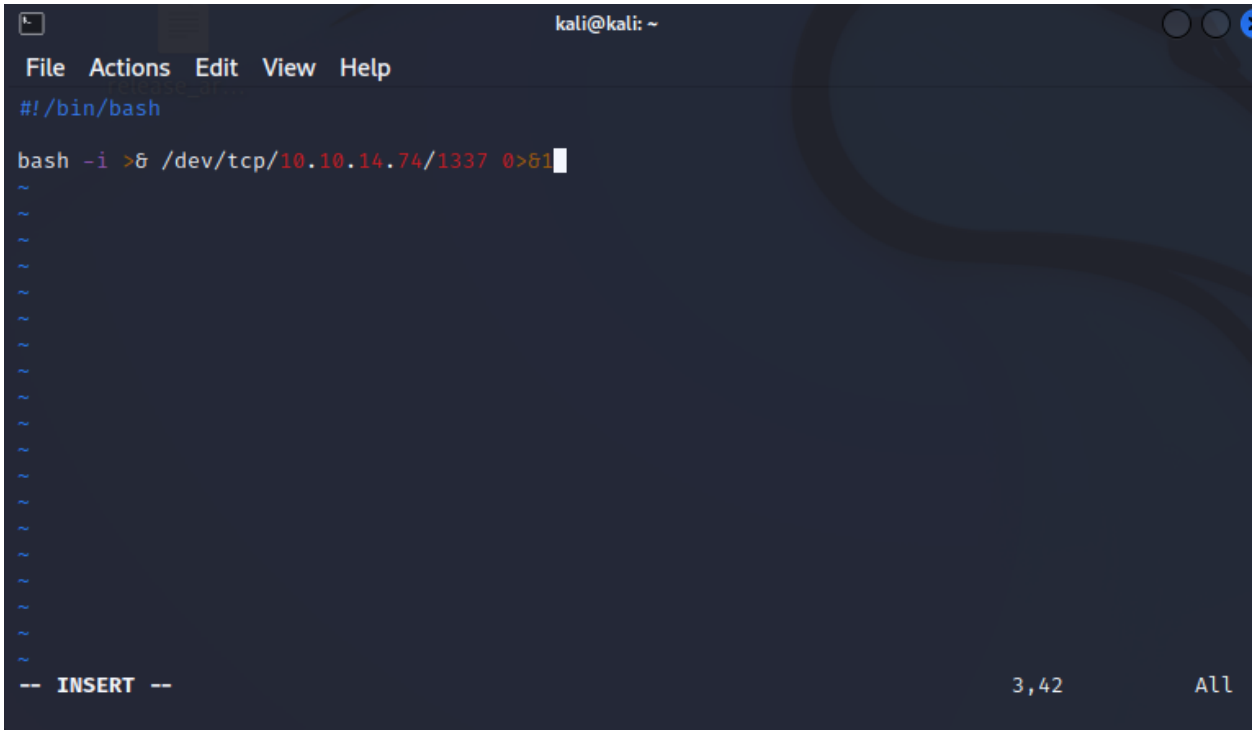

Figura 9: File pre-commit

Lo script consente di creare una reverse shell. Il contenuto può essere sostituito con qualsiasi comando di shell. (Bisogna assicurarsi che il file abbia i permessi di esecuzione). La figura 10 mostra l'ottenimento della reverse shell nel terminale in cui eravamo in ascolto con il tool `netcat`.

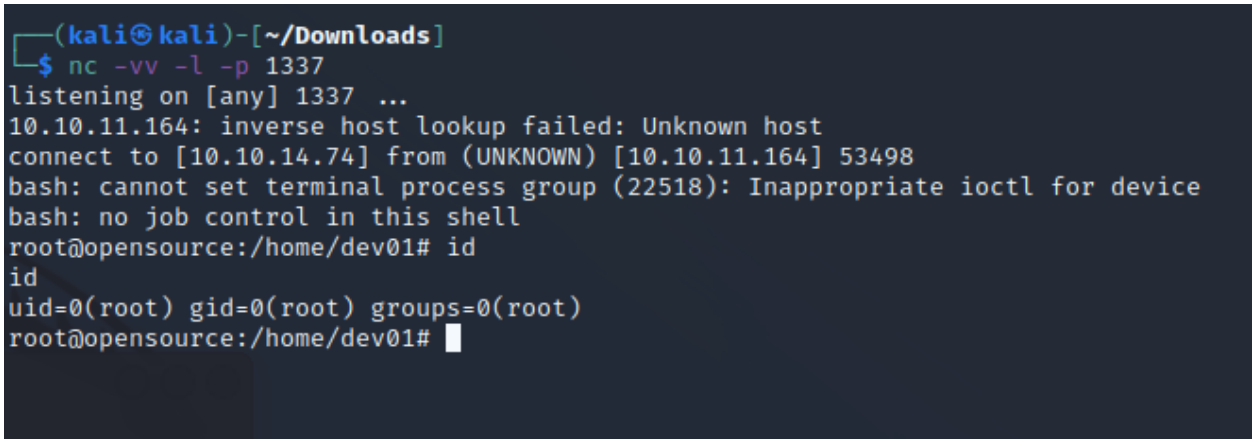

Figura 10: Ottenimento reverse shell