

Università Federico II di Napoli

Progetto congiunto corsi Network Security e Applicazioni telematiche



Remote Hacking With Docker



Docente: *Simon Pietro Romano*

Studente: *Vincenzo Roffo M63/813*

Introduzione

1

Remote Hacking With Docker è un'applicazione web che permette l'esecuzione di comandi di hacking mediante un'interfaccia in stile "Linux-terminal".

Sebbene all'utente sia trasparente, la computazione è in realtà demandata ad un host remoto il quale, tramite il lancio di container Docker (con l'opzione "-rm"), esegue i comandi che riceve. Il client visualizza l'output restituito dal processo server in tempo reale, e ciascuna delle esecuzioni avviata dal client può essere, da quest'ultimo, in qualsiasi istante interrotta dal client invocando un apposito comando.

1.1 Prerequisiti

Gli unici strumenti *software* che è necessario installare sulla propria macchina al fine di testare e/o utilizzare l'applicazione, sono:

- **Docker Engine**, lato server
- **Browser web** qualsiasi, lato client

I restanti moduli software, necessari a risolvere le dipendenze dei moduli sorgenti, verranno scaricati in automatico seguendo la procedura illustrata di seguito.

1.2 Getting started

Per avviare l'applicazione, eseguire le seguenti azioni:

1. installare le dipendenze del back-end ed avviarlo:

```
cd server  
sudo npm install  
sudo node server.js
```

2. installare le dipendenze del front-end ed avviarlo:

```
cd client/live-terminal  
sudo npm install  
ng -o serve
```

Verrà così automaticamente aperto il browser di default all'indirizzo **"localhost:4200/"**.

Si noti che le informazioni sulle dipendenze dei moduli implementati sono descritte nei file **"*package.json*"** che non vanno quindi spostati o rimossi, se si vuole che questa procedura abbia esito positivo.

Dal punto di vista architetturale, l'applicazione si compone di **due sotto-sistemi**, che fungono da *front-end* e *back-end*. Il primo permette l'interfacciamento dell'utente per l'utilizzo delle funzionalità implementate, mentre il secondo realizza la vera e propria esecuzione che restituisce l'output mostrato dal browser (ad eccezione dei comandi di *utility* eseguiti nel client stesso).

I dati transitano tra i due sotto-sistemi secondo il protocollo **HTTP** in formato **JSON**.

Non sono stati previsti meccanismi di protezione del canale né di autenticazione.

2.1 Back-end

Il back-end dell'applicazione è stato implementato mediante il noto framework web **Express** basato sul runtime di JavaScript Node.js molto flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web. Al fine di realizzare le funzionalità già precedentemente citate, si è fatto uso delle seguenti librerie:

- **Express** (<https://expressjs.com/it/>)
- **Path**, per la navigazione dei file tramite directory e percorsi (<https://nodejs.org/api/path.html>)

- **Socket.io**, per permettere la comunicazione real.time tra I due end-point (<https://socket.io/docs/>)
- **Http**, per implementare un server che comunicasse sfruttando questo protocollo (<https://nodejs.org/api/http.html>)
- **Child_process**, per la creazione asincrona di processi figli (https://nodejs.org/api/child_process.html)

Si noti che socket.io non è una semplice implementazione della tecnologia **Websocket**, nonostante nei fatti la utilizzi. Aggiunge, difatti, alcuni metadati a ciascun pacchetto:

- il tipo di pacchetto
- lo spazio dei nomi
- l'id ack, quando è necessario un riconoscimento del messaggio

Questo è il motivo per cui un client WebSocket non sarà in grado di connettersi correttamente a un server socket.io e neanche un client socket.io sarà in grado di connettersi a un server WebSocket. Per la specifica completa del protocollo si rimanda qui: <https://github.com/socketio/socket.io-protocol>.

Stabilita la connessione con un client, il server si mette in ascolto. Essendo socket.io **event-based** e **bi-direzionale**, non appena il client genera un evento inviando un comando da eseguire, il server estrae il valore degli argomenti dai dati JSON e “*spawna*” un processo figlio con il comando docker e il comando e le opzioni che ha appena estratto. Qualsiasi tipo di **output** prodotto dal processo figlio è disponibile al padre che può quindi generare un evento sulla connessione stabilita col client per permettergli di riceverlo in tempo reale.

Il comando “**run**” e l’opzione “**-rm**” si è scelto di non cablarli nel codice, seppur siano presenti in ciascuna richiesta di esecuzione di comando, in modo da rendere più riusabile questo script.

In ogni caso, ciascun container una volta terminata la propria esecuzione verrà automaticamente ripulito e il suo file system verrà rimosso.

I processi generati per l'esecuzione di comandi docker possono essere interrotti (**killed**) su richiesta asincrona del client.

2.2 Front-end

Il front-end dell'applicazione è stato sviluppato mediante la nota piattaforma di sviluppo open-source **Angular**. Questo ha favorito la modularità e semplificato l'implementazione, grazie al forte disaccoppiamento tra logica di business e presentazione fornito dal modello architetturale **Model-View-Whatever** a cui Angular è conforme.

L'architettura software dell'applicativo client implementato prevede:

- un componente **app**, presente in qualsiasi applicazione Angular. Che rappresenta l'involucro per tutti gli altri componenti e viene richiamato dall'**app-module** per il bootstrap dell'applicazione
- un componente **terminal**, rappresentante il terminale "*linux-style*"
- un servizio **websocket**, per la creazione di una socket-client finalizzata alla gestione di qualsiasi interazione tra i due sotto-sistemi implementati tramite emissione e ricezione di eventi

2.2.1 Terminal.component

Il *terminal.component* Angular è il fulcro dell'applicazione client. In sostanza ne realizza tutte funzionalità, escluse quelle relative alla connettività, che sono demandate al *websocket.service*. Per implementare un terminale all'interno del browser si è utilizzato un componente “**ng-terminal**” importato, di cui si riporta la documentazione: <https://www.npmjs.com/package/ng-terminal>.

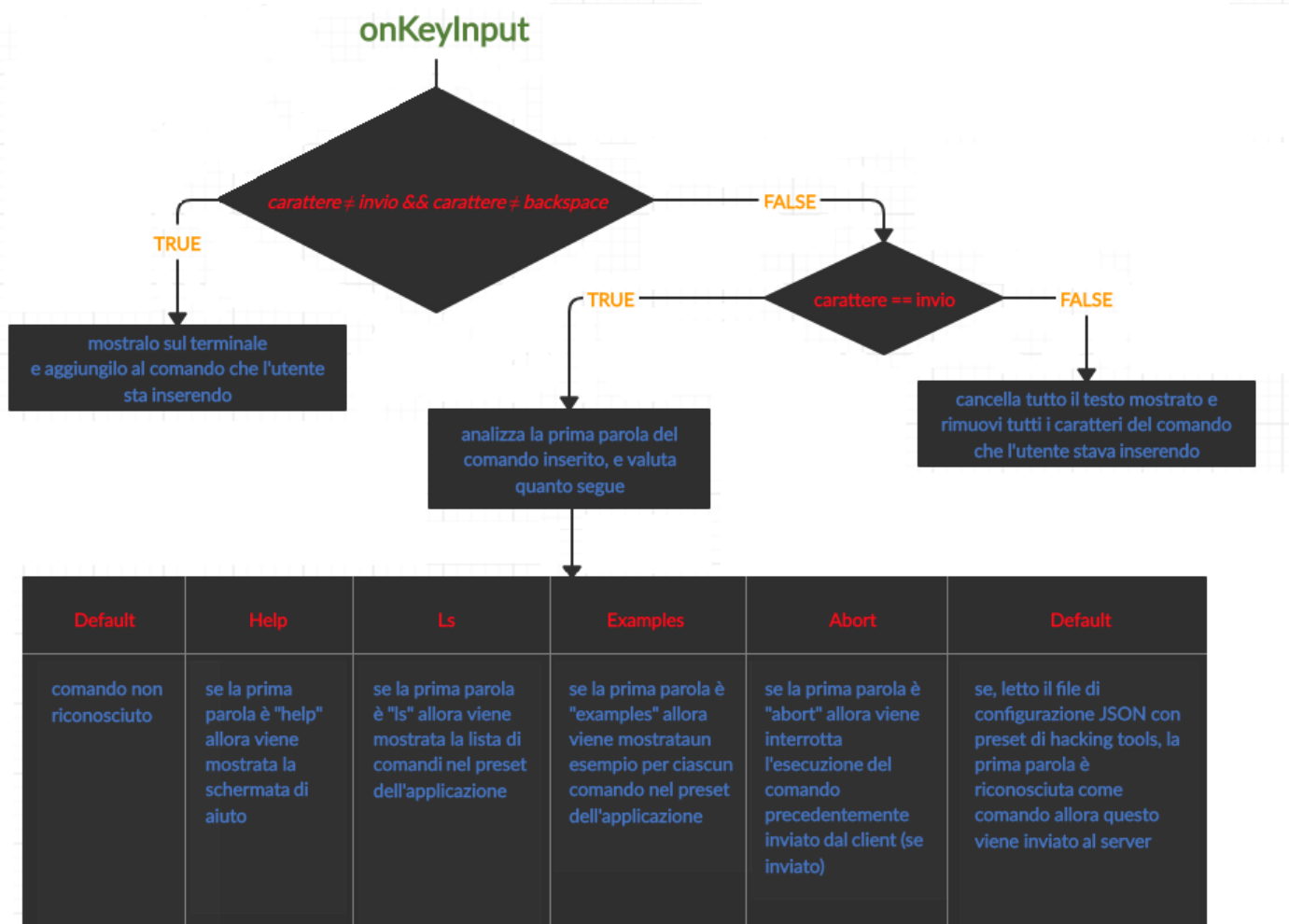
La presentazione del componente è stata quindi del tutto ereditata utilizzando il tag HTML **<ng-terminal>**, a cui sono associati: una sorgente dati , per l'inserimento di testo; un evento (e un handler), associato all'inserimento di qualsiasi carattere da tastiera; le opzioni di display.

Per quanto concerne la prima, tutto il testo che l'utente digita tramite tastiera viene mostrato sul terminale. Le principali caratteristiche relativamente all'inserimento di caratteri sono tre:

1. la sostituzione di un carattere tramite posizionamento del cursore genera una stringa che non corrisponde a quella mostrata a video, quindi probabilmente il comando eseguito non verrà riconosciuto
2. gli spazi tra parole possono essere anche più di uno, questo non causa errori
3. la pressione del tasto **Backspace** causa la cancellazione di tutto il testo corrente sul terminale e non solo dell'ultimo carattere, il che comporta l'inserimento *ex-novo* di tutto il comando che si stava inserendo

Si noti inoltre che **non** è possibile l'operazione “*scroll*” sul terminale.

In relazione al secondo invece, all'inserimento di un carattere da tastiera sul terminale è associato un evento gestito da un apposito metodo, che esegue secondo il seguente diagramma di flusso:



Per quanto riguarda le opzioni di display, queste sono statiche e cablate nel codice, nonostante il componente importato prevedesse meccanismi di modifica dinamici.

La formattazione del comando inserito in formato JSON è implementato da un apposito metodo contenuto anch'esso nel file *typescript* del componente.

2.3 File di configurazione

L'implementazione di un file di configurazione in formato JSON è stata necessaria (e sensata) solamente per il front-end dell'applicazione. Difatti non ci sono settaggi possibili per quanto riguarda il server, essendo stato pensato per questa applicazione specifica.

Il file contiene per ciascuno dei tool supportati dall'applicazione:

- il **nome** del comando linux
- l'**immagine docker** ad esso associata
- un **esempio** di utilizzo

Seppur Docker permette l'esecuzione di immagini non presenti in locale (scaricandole in automatico), i comandi relativi ad immagini non presenti nel preset di questo file non si espliciteranno in richieste al server.

Esempi di utilizzo

3

In questo paragrafo vengono riportati alcuni esempi di utilizzo dell'applicazione.

Questi ultimi mirano esclusivamente a dare un'idea dell'applicazione.

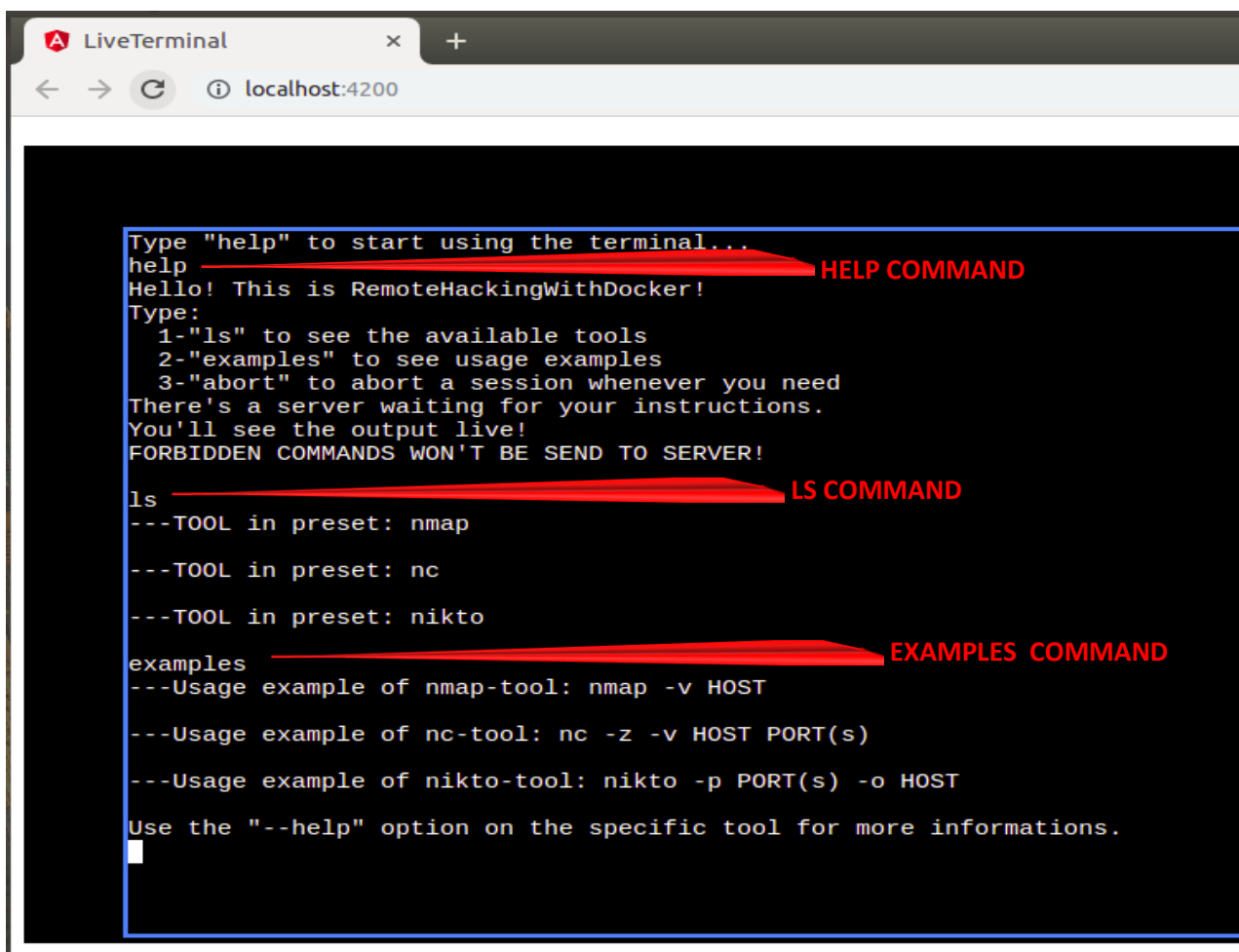
A tal proposito vengono allegati alcuni *screenshot* che mostrano sia l'interfaccia client (front-end), sia il terminale in cui è in esecuzione il processo server.

Si noti che due ulteriori prerequisiti all'utilizzo di quest'applicazione, sono la consapevolezza e praticità nell'uso dei tool supportati. Per chi, invece, fosse alle prime armi, oltre che consultare la schermata di *help* di ciascun tool invocandola esplicitamente, può consultare le documentazioni di cui si allegano i link:

- *nmap*: <https://nmap.org/docs.html>
- *nikto*; <https://cirt.net/nikto2-docs/>
- *netcat*: <https://nmap.org/ncat/guide/>

3.1 Comandi di utility

I comandi di utility sono direttive eseguite esclusivamente lato client, che assistono l'utente nell'utilizzo dell'applicazione.



The screenshot shows a web browser window with a tab titled "LiveTerminal". The address bar shows "localhost:4200". The terminal content is as follows:

```
Type "help" to start using the terminal...
help
Hello! This is RemoteHackingWithDocker!
Type:
  1-"ls" to see the available tools
  2-"examples" to see usage examples
  3-"abort" to abort a session whenever you need
There's a server waiting for your instructions.
You'll see the output live!
FORBIDDEN COMMANDS WON'T BE SEND TO SERVER!

ls
---TOOL in preset: nmap

---TOOL in preset: nc

---TOOL in preset: nikto

examples
---Usage example of nmap-tool: nmap -v HOST

---Usage example of nc-tool: nc -z -v HOST PORT(s)

---Usage example of nikto-tool: nikto -p PORT(s) -o HOST

Use the "--help" option on the specific tool for more informations.
█
```

Red annotations in the original image point to specific commands:

- HELP COMMAND** points to the `help` command.
- LS COMMAND** points to the `ls` command.
- EXAMPLES COMMAND** points to the `examples` command.

3.2 Comando di hacking

Esecuzione di un comando di hacking presente all'interno del preset definito all'interno del file di configurazione del client. Dopo l'invocazione del comando **"nmap -v 127.0.0.1-2"** viene *real-time* mostrato il seguente output, che verrà quindi stampato progressivamente. Come visibile nel terminale Linux mostrato, il server ha ricevuto i 5 parametri per l'esecuzione del container docker con l'immagine associata al comando specificato.

```
LiveTerminal x +
localhost:4200

Starting Nmap 7.60 ( https://nmap.org ) at 2019-08-26 18:20 GMT
Initiating Parallel DNS resolution of 2 hosts. at 18:20
Completed Parallel DNS resolution of 2 hosts. at 18:20, 0.03s elapsed
Initiating SYN Stealth Scan at 18:20
Scanning 2 hosts [1000 ports/host]
Increasing send delay for 127.0.0.1 from 0 to 5 due to 17 out of 56 dropped probes since last increase.
Increasing send delay for 127.0.0.2 from 0 to 5 due to 18 out of 58 dropped probes since last increase.
Increasing send delay for 127.0.0.1 from 5 to 10 due to 11 out of 35 dropped probes since last increase.
Increasing send delay for 127.0.0.2 from 5 to 10 due to 11 out of 35 dropped probes since last increase.
Increasing send delay for 127.0.0.2 from 10 to 20 due to 16 out of 53 dropped probes since last increase.
Increasing send delay for 127.0.0.1 from 10 to 20 due to 17 out of 55 dropped probes since last increase.
Increasing send delay for 127.0.0.2 from 20 to 40 due to 11 out of 27 dropped probes since last increase.
Increasing send delay for 127.0.0.1 from 20 to 40 due to 11 out of 27 dropped probes since last increase.
Increasing send delay for 127.0.0.1 from 40 to 80 due to 11 out of 32 dropped probes since last increase.
Increasing send delay for 127.0.0.2 from 40 to 80 due to 11 out of 33 dropped probes since last increase.
SYN Stealth Scan Timing: About 41.08% done; ETC: 18:21 (0:00:44 remaining)
Completed SYN Stealth Scan against 127.0.0.1 in 94.94s (1 host left)
Completed SYN Stealth Scan at 18:21, 94.94s elapsed (2000 total ports)
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000078s latency).
All 1000 scanned ports on localhost (127.0.0.1) are closed

Nmap scan report for 127.0.0.2
Host is up (0.000077s latency).
All 1000 scanned ports on 127.0.0.2 are closed

Read data files from: /usr/bin/./share/nmap
Nmap done: 2 IP addresses (2 hosts up) scanned in 95.08 seconds
Raw packets sent: 2656 (116.864KB) | Rcvd: 3984 (164.672KB)
```

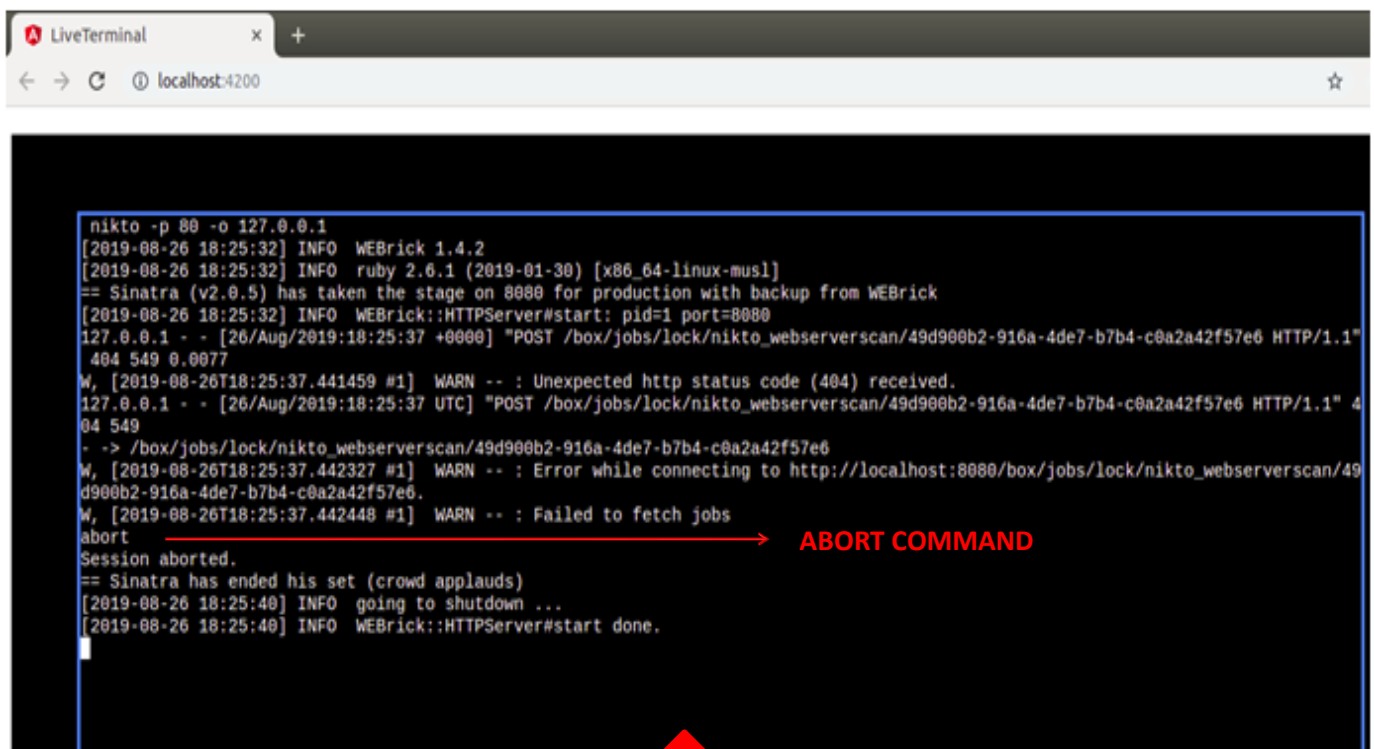


```
Listening on port 3000
Client to server: HEAR ME?
{"arg0":"run","arg1":"--rm","arg2":"uzyexe/nmap","arg3":"-v","arg4":"127.0.0.1-2"}

arg0
arg1
arg2
arg3
arg4
```


3.2 Comando abort

Il comando *abort* è utile ad interrompere l'esecuzione di un processo precedentemente avviato, qualora il client lo richieda.



```
nikto -p 80 -o 127.0.0.1
[2019-08-26 18:25:32] INFO WEBrick 1.4.2
[2019-08-26 18:25:32] INFO ruby 2.6.1 (2019-01-30) [x86_64-linux-musl]
== Sinatra (v2.0.5) has taken the stage on 8080 for production with backup from WEBrick
[2019-08-26 18:25:32] INFO WEBrick::HTTPServer#start: pid=1 port=8080
127.0.0.1 - - [26/Aug/2019:18:25:37 +0000] "POST /box/jobs/lock/nikto_webserverscan/49d900b2-916a-4de7-b7b4-c0a2a42f57e6 HTTP/1.1"
404 549 0.0077
W, [2019-08-26T18:25:37.441459 #1] WARN -- : Unexpected http status code (404) received.
127.0.0.1 - - [26/Aug/2019:18:25:37 UTC] "POST /box/jobs/lock/nikto_webserverscan/49d900b2-916a-4de7-b7b4-c0a2a42f57e6 HTTP/1.1" 4
04 549
- -> /box/jobs/lock/nikto_webserverscan/49d900b2-916a-4de7-b7b4-c0a2a42f57e6
W, [2019-08-26T18:25:37.442327 #1] WARN -- : Error while connecting to http://localhost:8080/box/jobs/lock/nikto_webserverscan/49
d900b2-916a-4de7-b7b4-c0a2a42f57e6.
W, [2019-08-26T18:25:37.442448 #1] WARN -- : Failed to fetch jobs
abort
Session aborted.
== Sinatra has ended his set (crowd applauds)
[2019-08-26 18:25:40] INFO going to shutdown ...
[2019-08-26 18:25:40] INFO WEBrick::HTTPServer#start done.
```

ABORT COMMAND



```
Listening on port 3000
Client to server: HEAR ME?
{"arg0":"run","arg1":"--rm","arg2":"securecodebox/nikto","arg3":"-p","arg4":"80","arg5":"-o","arg6":"127
.0.0.1-4"}
arg0
arg1
arg2
arg3
arg4
arg5
arg6
KILLED PROCESS!
child process exited with code 0
```

SERVER KILLED THE CHILD PROCESS