

Build Week 3 - ES BONUS 1

Analisi Forense di SQL Injection e Esfiltrazione

Dati tramite DNS su Security Onion

Executive Summary

Nel presente laboratorio è stata condotta un'attività di threat hunting attraverso l’analisi dei log HTTP e DNS utilizzando Kibana su piattaforma Security Onion.

L’indagine ha evidenziato due tecniche di compromissione distinte:

- 1. SQL Injection via HTTP**, utilizzata per accedere in modo illecito al database dell’applicazione web ed estrarre dati sensibili (credit card, credenziali).
- 2. Esfiltrazione dati via DNS**, mediante l’uso di sottodomini contenenti stringhe esadecimali codificate, successivamente decodificate per ricostruire un documento riservato.

L’analisi ha permesso di identificare l’IP dell’attaccante, il server bersaglio, il contenuto delle richieste malevoli e il meccanismo di esfiltrazione.

Il laboratorio dimostra come sia possibile individuare attività malevole analizzando i log di rete e correlando informazioni HTTP e DNS.

Introduzione

L’obiettivo del progetto è **analizzare traffico di rete sospetto per individuare comportamenti anomali e tecniche di attacco** comunemente utilizzate in scenari reali.

Nel primo scenario è stata analizzata un’attività HTTP che mostrava parametri sospetti all’interno dell’URL, riconducibili a un tentativo di SQL Injection finalizzato all’estrazione di dati dal database.

Nel secondo scenario è stata investigata una serie di query DNS contenenti **sottodomini** insolitamente lunghi. Attraverso l’esportazione e la decodifica delle stringhe esadecimali, è stato possibile dimostrare un caso di esfiltrazione dati tramite **DNS tunneling**.

L’analisi è stata condotta utilizzando:

- **Security Onion**

- **Kibana (Zeek Hunting)**
- **capME! per ricostruzione sessione HTTP**
- **xxd per decodifica dati esadecimali**

Obiettivi / Scenario

Usare Kibana (Security Onion) per investigare esfiltrazione dati via HTTP (SQLi) e via DNS (sottodomini lunghi / esadecimale).

- Pagina/intro del laboratorio in Kibana (home/dashboard) con titolo “Zeek Hunting” visibile.

Parte 1 (SQL Injection) — Passo 1: cambiare intervallo di tempo

1) Avvio VM + check servizi

1. Avvia **Security Onion VM**
2. Login: **analyst / cyberops**
3. Apri terminale ed esegui:
 - `sudo so-status`

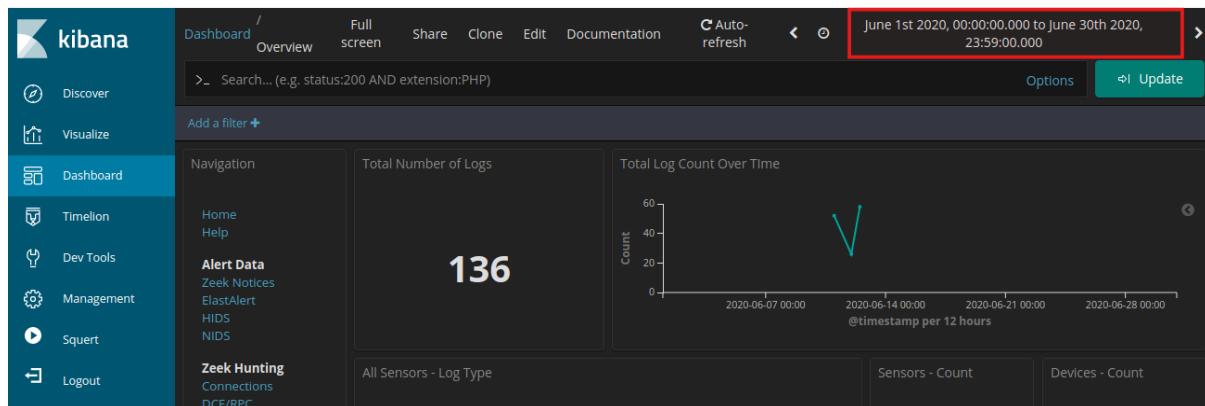
- Verifica che tutti i servizi siano **[OK]** (Elastic stack incluso).

```
analyst@SecOnion:~] sudo so-status
Status: securityonion
  * sguil server
Status: seconion-import
  * pcap_agent (sguil)
  * snort_agent-1 (sguill)
  * barnyard2-1 (spooler, unified2 format)
Status: Elastic stack
  * so-elasticsearch
  * so-logstash
  * so-kibana
  * so-freqserver
```

- Terminale con **sudo so-status** e righe **[OK]** visibili.

2) Apri Kibana e imposta Giugno 2020 (Absolute)

- Apri **Kibana** dal collegamento sul Desktop
- Login: **analyst / cyberops**
- In alto a destra clic su **Last 24 hours**
- Seleziona **Absolute**
- Imposta:**
 - From:** 1 Giugno 2020 (00:00)
 - To:** 30 Giugno 2020 (23:59)
- Clic su **Go**.

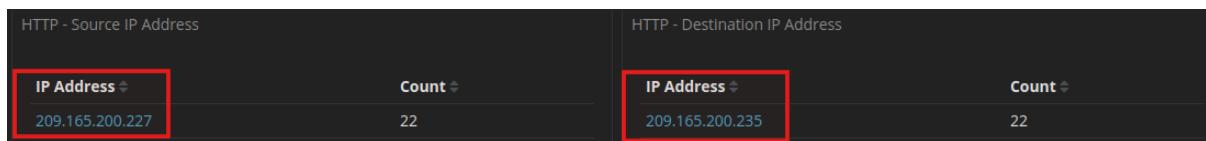
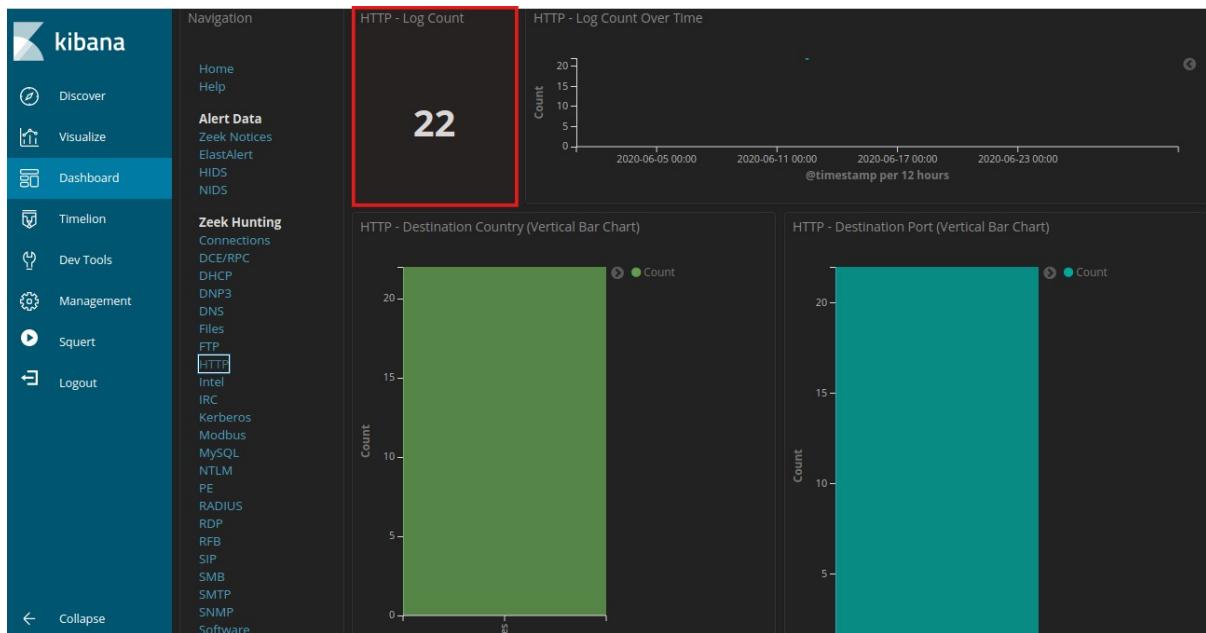


- Pop-up Time Range con **Absolute** e date di giugno 2020 visibili.

Passo 2: filtrare per traffico HTTP

3) Seleziona filtro HTTP (Zeek Hunting)

- Nel pannello (Zeek Hunting) clic su **HTTP**
- Scorri i risultati e rispondi alle domande.



- Kibana con filtro **HTTP** attivo + tabella log visibile (campi/colonne o dettagli con IP e porta).

Domande + Risposte (valori da leggere su Kibana)

D1. Qual è l'indirizzo IP sorgente?

Risposta: È l'IP del client/attaccante che avvia la richiesta.

D2. Qual è l'indirizzo IP destinazione?

Risposta: È il server web bersaglio.

D3. Qual è il numero di porta destinazione?

Risposta: 80/443 per web)

Dettaglio primo log HTTP (expand)

4) Espandi il primo risultato (freccia accanto al timestamp)

1. Scorrere fino a **HTTP Logs**
2. Espandi il **primo** risultato cliccando la freccia vicino al timestamp

HTTP - Logs											
Time ▾	source_ip	destination_ip	destination_port ^x «»	resp_fuids	uid	_id					
June 12th 2020, 21:30:09.445	209.165.200.227	209.165.200.235	80	FEvWs63HqvCqth3LH1	CuKeR52aPJRN7PfqDd	ZzjrzXIBB6Cd- OSD_iW					
View surrounding documents View single document											
Table		JSON									
○ @timestamp	Q R D * June 12th 2020, 21:30:09.445										
t @version	Q R D * 1										
t _id	Q R D * ZzjrzXIBB6Cd- OSD_iW										
t _index	Q R D * seconion:logstash-import-2020.06.12										
# _score	Q R D * -										
t _type	Q R D * doc										
t destination_geo.city_name	Q R D * Monterey										

- Log HTTP espanso dove si vedono chiaramente:
 - **timestamp**
 - **event type**
 - **message** (con riferimento a GET e soprattutto **uri**)

Domande + Risposte

D4. Qual è il timestamp del primo risultato?

Risposta: il valore del campo **@timestamp**

D5. Qual è il tipo di evento?

Risposta: il valore del campo **event_type** / **zeek.event** / **event.dataset** **D6. Cosa è incluso nel campo message? (focus su uri)**

Risposta:

Nel campo **message** sono riportati i **dettagli della richiesta HTTP GET** (client → server): metodo, host/destinazione e soprattutto l'**URI**, cioè la **risorsa richiesta** e spesso anche **parametri** (query string) utili a capire cosa stava tentando di fare l'attaccante.

D7. Qual è il significato di queste informazioni?

Risposta:

L'**URI** e i parametri permettono di identificare l'**azione** svolta (pagina/endpoint colpito), e se contiene pattern anomali (es. **union select**, payload, stringhe sospette) indica **tentativo di SQL injection** o abuso applicativo per accedere a dati sensibili.

Passo 3: aprire capME! e trovare SQLi

5) Aprire capME! dal campo **alert_id**

1. Nel log espanso, trova il campo **alert_id** (link)
2. Cliccare il valore → si apre **capME!** in una nuova scheda
3. In **capME!**:
 - **Blu** = richieste HTTP inviate da **SRC**

- **Rosso** = risposte del server DST
4. Nella sezione iniziale “Log entry”, individua la porzione di stringa con:
- `username='+union+select+ccid,ccnumber,ccv,expiration,null+from+credit_cards...&password=`

```

209.165.200.227:56194 209.165.200.235:80-6-1022147824.pcap
Log entry:
[{"ts": "2020-06-12T21:30:09.445030Z", "uid": "CuKeR52aPjRN7PfqDd", "id.orig_h": "209.165.200.227", "id.orig_p": "56194", "id.resp_h": "209.165.200.235", "id.resp_p": "80", "trans_dept_h": "1", "method": "GET", "host": "209.165.200.235", "uri": "/multilidae/index.php?page=user-info.php&username='+union+select+ccid,ccnumber,ccv,expiration,null+from+credit_cards...+&password='&user-info-php-submit-button=View+Account+Details", "referrer": "http://209.165.200.235/multilidae/index.php?page=user-info.php", "version": "1.1", "user_agent": "Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0", "request_body_len": 0, "response_body_len": 23665, "status_code": 200, "status_msg": "OK", "tags": ["HTTP-URI_SQL"], "resp_fuids": ["FEVWws63HqvCqfh3LH1"], "resp_mime_types": ["text/html"]}

Sensor Name: seconion-import
Timestamp: 2020-06-12 21:30:09
Connection ID: CLI
Src IP: 209.165.200.227
Dst IP: 209.165.200.235
Src Port: 56194
Dst Port: 80
OS Fingerprint: 209.165.200.227:56194 - UNKNOWN [S44:64:1:60:M1460,S,T,N,W7...?:?] (up: 2829 hrs)
OS Fingerprint: -> 209.165.200.235:80 (link: ethernet/modem)

```

- Pagina capME! con testo visibile + porzione con `union select ... from credit_cards` leggibile.

Domande + Risposte

D8. Qual è il significato della stringa con `union select ... from credit_cards`?

Risposta:

È un chiaro payload di SQL injection: l'attaccante tenta di bypassare/forzare l'autenticazione e soprattutto di estrarre dati dal database (tabella `credit_cards`) usando comandi SQL (`UNION SELECT`) per ottenere campi come `ccid`, `ccnumber`, `ccv`, `expiration`.

6) Cerca `username` nella trascrizione (Ctrl+F)

1. In capME! premi **Ctrl+F**
2. Cerca `username`
3. Scorri i match con freccia giù come richiesto
4. Osserva cosa compare **più avanti** nella trascrizione.

Zeek - HTTP - Kibana capME! localhost:capme/elastic.php?esid=ZzjrZIBB6Cd-_OSD_iW

```
DST: <tr>
DST: <tr><td></td></tr>
DST: <tr>
DST: ...<td colspan="2" style="text-align:center; font-style: italic;">
DST: ....Dont have an account? <a href="#">Please register here</a>
DST: </td>
DST: </tr>
DST: <table>
DST: <form>
DST:
DST:
DST: 3a
DST: <p class="report-header">Results for . 5 records found <p>
DST:
DST: 24
DST: <b>Username=</b>4444111122223333<br>
DST:
DST: 17
DST: <b>Password=</b>745<br>
DST:
DST: 22
DST: <b>Signature=</b>2012-03-01<br><p>
DST:
DST: 24
DST: <b>Username=</b>7746536337776330<br>
DST:
DST: 17
DST: <b>Password=</b>722<br>
DST:
DST: 22
DST: <b>Signature=</b>2015-04-01<br><p>
DST:
DST: 24
```

- capME! con ricerca **username** evidenziata + almeno 1 punto dove si vedono credenziali/dati esfiltrati.

Domande + Risposte

D9. Cosa vedi più avanti nella trascrizione riguardo ai nomi utente?

Risposta:

Più avanti compaiono **valori di username non “normali”** o ripetuti come dati, e soprattutto si vedono **credenziali e/o stringhe** che indicano **estrazione/esposizione di informazioni** (non solo login legittimo). In pratica la trascrizione mostra che l’attaccante sta usando richieste per ottenere record e non semplicemente autenticarsi.

D10. Fornisci alcuni esempi di username, password e firma esfiltrati.

Risposta:

```
DST: <b>Username</b><br>
DST:
DST: 17
DST: <b>Password</b><br>
DST:
DST: 22
DST: <b>Signature</b><br><p>
DST:
DST: 24
DST: <b>Username</b><br>
DST:
DST: 17
DST: <b>Password</b><br>
DST:
DST: 22
DST: <b>Signature</b><br><p>
DST:
DST: 24
DST: <b>Username</b><br>
DST:
DST: 17
DST: <b>Password</b><br>
DST:
DST: 22
DST: <b>Signature</b><br><p>
```

Output di 3 esempi reali direttamente dalla trascrizione capME!

Qui i valori dipendono dai log presenti nella VM

Successivamente nell'ultimo screenshot si evince il **caricamento del codice sorgente (HTML e JavaScript)** di una pagina web, .

In termini di cybersecurity ecco cosa indicano quegli elementi:

- Il target: Mutilidae

Si vede chiaramente un link a **Mutilidae** (un'applicazione web PHP deliberatamente vulnerabile, parte del progetto OWASP). È il bersaglio classico per esercitarsi in attacchi come:

- **SQL Injection**
- **Cross-Site Scripting (XSS)**.

- Versione PHP Vulnerabile

La riga **PHP Version: 5.2.4-2ubuntu5.10** è un segnale enorme per un analista. Quella è una versione estremamente vecchia e obsoleta di PHP, piena di vulnerabilità note che rendono l'attacco molto più semplice da eseguire.

- JavaScript e LocalStorage

La sezione finale con **<script type="text/javascript">** mostra dei tentativi di scrivere dati nel **LocalStorage** e **SessionStorage** del browser:

- **window.localStorage.setItem(...)**

- `window.sessionStorage.setItem(...)` Questo codice viene spesso analizzato per capire se un utente malintenzionato sta cercando di rubare **token di sessione** o iniettare script malevoli per tracciare l'utente.

Nello screenshot si nota anche un comando `alert(e)`. In un contesto reale, questo è tipico di un attacco **XSS**, dove l'attaccante verifica se può far eseguire codice JavaScript arbitrario nel browser della vittima.

```
DST:  
DST: 38  
DST: <div class="footer">PHP Version: 5.2.4-2ubuntu5.10</div>  
DST:  
DST: 13b  
DST:  
DST: .<div class="footer">  
DST: ..The newest version of  
DST: ..<a href="http://www.irongeek.com/i.php?page=security/mutillidae-deliberately-vulnerable-php-owasp-top-10" target="_blank">  
DST: ...Mutillidae  
DST: ..</a>  
DST: ..can downloaded from <a href="http://irongeek.com" target="_blank">Irongeek's Site</a>  
DST: .</div>  
DST: </body>  
DST: </html>  
DST:  
DST: 116  
DST: <script type="text/javascript">  
DST: ...try{  
DST: ....window.localStorage.setItem("LocalStorageTarget", "This is set by the index.php page");  
DST: ....window.sessionStorage.setItem("SessionStorageTarget", "This is set by the index.php page");  
DST: ...}catch(e){  
DST: ....alert(e);  
DST: ...};  
DST: ..</script>  
DST:  
DST: 0  
DST:  
DST:
```

DEBUG: Using archived data: /nsm/server_data/securityonion/archive/2020-06-12/seconion-import/209.165.200.227:56194_209.165.200.235:80-6.raw
 QUERY: SELECT sid FROM sensor WHERE hostname='seconion-import' AND agent_type='pcap' LIMIT 1
 CAPME: Processed transcript in 1.30 seconds: 0.41 0.76 0.00 0.13 0.00

[209.165.200.227:56194_209.165.200.235:80-6-1022147824.pcap](#)

In sintesi: Nell'ultimo screenshot è visibile il codice sorgente HTML e JavaScript della pagina Mutillidae. È esposta la versione PHP 5.2.4-2ubuntu5.10, obsoleta e nota per vulnerabilità pubbliche.

È inoltre presente codice JavaScript che utilizza localStorage e sessionStorage per memorizzare dati lato client.

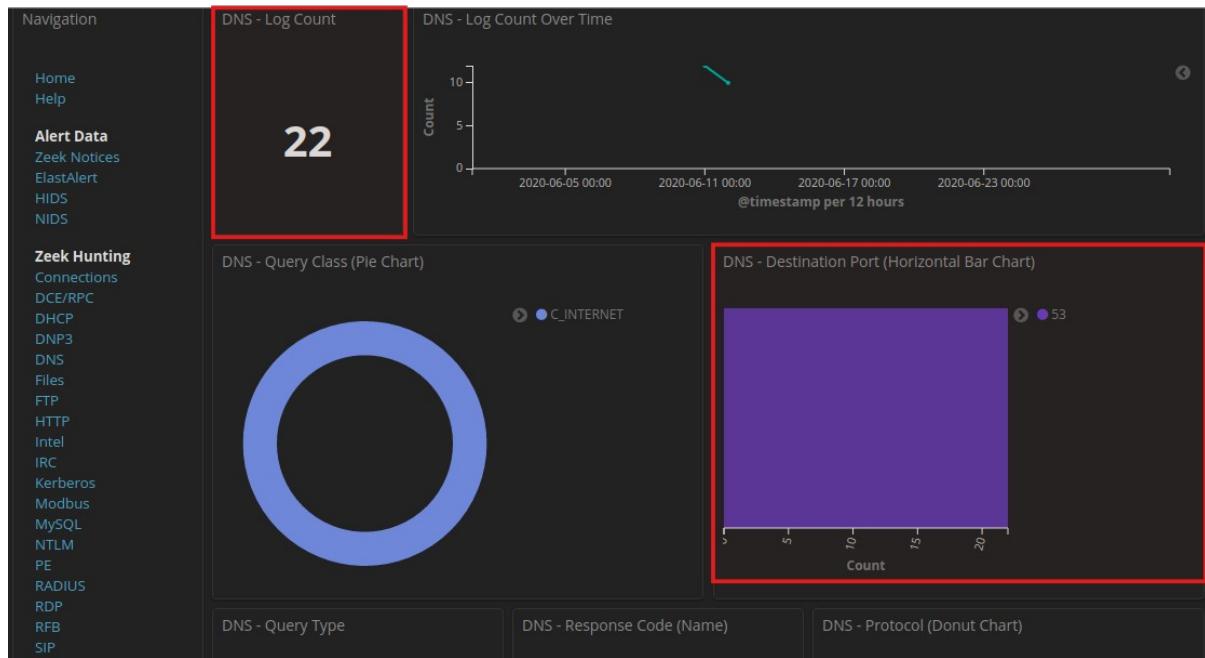
Il blocco try/catch con alert(e) indica gestione di errori JavaScript e non rappresenta di per sé un attacco XSS attivo.

Particolarmente rilevante è la presenza di informazioni di debug che espongono query SQL interne, configurazione del sistema e percorsi file, configurando un caso di Information Disclosure.

Parte 2 (DNS Exfiltration) — Passo 1–2: filtro DNS + analisi

7) Torna a Kibana e filtra DNS (Zeek Hunting)

1. Chiudi capME! e torna a Kibana
2. In alto **rimuovi filtri/ricerche** (clear)
3. Clic su **Home** (Navigation)
4. In Zeek Hunting clic su **DNS**
5. Nota “DNS Log Count” e grafico “Destination Port”.



- Kibana con **DNS** selezionato + metriche/grafici visibili.

8) Rivedi voci DNS e nota sottodomini lunghi verso **ns.example.com**

1. Scorri: tipi query (A, AAAA, NB, PTR), codici risposta
2. Scorri: top client DNS e server DNS
3. Scorri: top query DNS: nota query con sottodomini **insolitamente lunghi** collegati a **ns.example.com**
4. Devi investigare **example.com**.

The screenshot shows the Kibana Discover interface. On the left sidebar, under the 'Discover' section, there is a red box highlighting the 'DNS - Queries.csv' tab. The main panel displays a 'Query' section with several log entries. Below the query, it says 'No results found'. At the bottom, there are 'Raw' and 'Formatted' export options, with 'Raw' being highlighted by a red box.

- Sezione con query a ns.example.com e sottodomini lunghi ben visibili.

Filtro example.com + IP client/server DNS

9) Filtrare per example.com

1. Tornare in alto nella finestra DNS
2. Nella barra di ricerca inserisci: **example.com**
3. Clic su **Update**
4. Registrare IP in “DNS – Client” e “DNS – Server”.

The screenshot shows the Kibana Dashboard interface. In the top navigation bar, the search bar contains the text 'example.com' with a red box around it. The main dashboard area features a card titled 'Total Number of Logs' with the value '4' displayed prominently. To the right is a chart titled 'Total Log Count Over Time' showing a single data point at 4. Below these are three cards: 'All Sensors - Log Type' (Log Type(s): pro_dns, Count: 4), 'Sensors - Count' (0), and 'Devices - Count' (0). The left sidebar has a red box around the 'Discover' tab.

- Barra ricerca con example.com + (DNS Log Count ridotto)

Client	Count
209.165.200.235	18
192.168.0.11	4

Server	Count
8.8.4.4	6
173.36.131.10	6
173.37.87.157	6
209.165.200.235	4

- Sezione “DNS – Client” e “DNS – Server” con IP visibili

Domande + Risposte

D11. Registra gli indirizzi IP del client e del server DNS.

Risposta:

- **DNS Client IP:**
209.165.200.235
192.168.0.11
- **DNS Server IP:**
8.8.4.4.
173.36.131.10
173.37.87.157
209.165.200.235

Passo 3: determinare dati esfiltrati (Export Raw → hex → xxd)

10) Export Raw delle query DNS sospette

1. Scorri verso il basso: dovresti vedere **4 voci uniche** di query DNS verso example.com
2. Nota: sottodomini lunghi sembrano **esadecimale** (0-9, a-f)
3. Clic su **Export: Raw** per scaricare CSV (in [/home/analyst/Downloads](#))

The screenshot shows the Kibana interface with a sidebar on the left containing links like Discover, Visualize, Dashboard, Timeline, Dev Tools, Management, Squirt, and Logout. The main area has two panels: 'DNS - Queries' and 'DNS - Answers'. The 'DNS - Queries' panel contains a table with several rows of DNS queries. A red box highlights the 'Raw' export button at the bottom of this panel. To the right, a message says 'No results found' with a smiley face icon.

- Area dove si vede **Export: Raw** + query con sottodomini lunghi.

11) Pulisci il CSV (lascia solo l'hex)

1. Vai in [/home/analyst/Downloads](#)
2. Apri il CSV con editor (es. [gedit](#))
3. Elimina tutto tranne la parte **esadecimale** dei sottodomini
4. Rimuovi anche **virgolette**
5. Salva con lo **stesso nome** (come nel PDF).

The screenshot shows a CSV file titled 'DNS - Queries.csv' located in the 'Downloads' folder. The file contains the following hex data:
434f4e464944454e5449414c20444f43554d454e540a444f204e4f542053
484152450a5468697320646f63756d656e7420636f6e7461696e7320696e
666f726d6174696f6e2061626f757420746865206c617374207365637572
697479206272656163682e0a

- gedit con il file “ripulito” dove si vedono solo righe hex (come esempio nel PDF).

12) Decodifica con [xxd](#) e leggi il contenuto

Da terminale in Downloads:

```
xxd -r -p "DNS - Queries.csv" > secret.txt  
cat secret.txt
```

```
2 analyst@SecOnion:~} cd Downloads  
analyst@SecOnion:~/Downloads} ls  
DNS - Queries.csv  
analyst@SecOnion:~/Downloads} xxd -r -p "DNS - Queries.csv" > secret.txt  
analyst@SecOnion:~/Downloads} cat secret.txt  
CONFIDENTIAL DOCUMENT  
DO NOT SHARE  
This document contains information about the last security breach.
```

- Terminale con comando **xxd -r -p "DNS - Queries.csv" > secret.txt**
- **cat secret.txt** + output **cat secret.txt** visibile.

Domande finali

D12. I sottodomini delle query DNS erano sottodomini? Se no, qual è il testo?

Risposta:

No: non sono sottodomini “legittimi”, ma **dati codificati in esadecimale**. Decodificando, il testo è:

CONFIDENTIAL DOCUMENT

DO NOT SHARE

This document contains information about the last security breach.

D13. Cosa implica questo risultato riguardo a queste particolari richieste DNS? Qual è il significato più ampio?

Risposta:

Implica che le richieste DNS sono state usate come **canale di esfiltrazione**: i “sottodomini” contenevano in realtà **contenuto di un documento** spezzettato e trasmesso all'esterno tramite query DNS. A livello più ampio, dimostra una tecnica di **DNS tunneling/esfiltration**, spesso usata per bypassare controlli perché il DNS è quasi sempre consentito in uscita dalle reti.

D14. Cosa potrebbe aver creato queste query DNS codificate e perché è stato scelto il DNS come mezzo per esfiltrare dati?

Risposta:

Probabilmente uno **script/tool malevolo** (o malware) sul sistema compromesso che:

- prende dati (testo/PII/documenti),
 - li converte in **hex**,
 - li “impacchetta” in sottodomini e genera query verso un dominio controllato dall'attaccante (es. **ns.example.com**).
- Il DNS viene scelto perché:
- è spesso **permesso** anche quando HTTP/FTP sono filtrati,

- può passare inosservato tra traffico “normale”,
- consente di inviare piccoli chunk di dati in modo ripetuto.

Conclusioni

L'attività svolta ha dimostrato come un'analisi strutturata dei log di rete su **Security Onion** consenta di individuare e ricostruire attacchi reali combinando fonti differenti (HTTP e DNS) e strumenti di correlazione come Kibana (Zeek Hunting) e capME!.

Nel primo scenario è stata identificata una SQL Injection mirata all'estrazione di dati sensibili da un'applicazione web vulnerabile (Mutillidae). L'analisi dell'URI, del payload **UNION SELECT** e della trascrizione HTTP ha evidenziato un chiaro tentativo di accesso illecito al database e di esfiltrazione di credenziali e dati di carte di credito. Questo conferma come la semplice osservazione dei parametri nelle richieste GET possa rivelare attività di compromissione applicativa.

Nel secondo scenario è stata rilevata una tecnica più sofisticata: esfiltrazione dati tramite DNS tunneling. Le query apparentemente legittime verso `example.com` contenevano in realtà stringhe esadecimali che, una volta decodificate con `xxd`, hanno rivelato il contenuto di un documento riservato. Questo dimostra come il DNS, spesso considerato traffico “innocuo” e lasciato libero in uscita, possa essere abusato come canale covert per l'esfiltrazione di informazioni.

Dal punto di vista difensivo, il laboratorio evidenzia tre aspetti fondamentali:

- L'importanza della **visibilità centralizzata dei log**.
- Il valore della **correlazione tra protocolli diversi** (HTTP + DNS).
- La necessità di monitorare anche protocolli considerati “di servizio”, come il DNS.

In conclusione, l'esercitazione dimostra come un SOC analyst, attraverso attività di threat hunting metodico, possa identificare sia attacchi applicativi diretti (SQLi) sia tecniche di esfiltrazione stealth (DNS tunneling), rafforzando la capacità di detection e risposta agli incidenti in ambienti reali.